Suman Dahal

Sensor Platform for IoT Training

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Thesis
24 November 2015
Abstract

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Suman Dahal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Sensor Platform for IoT Training</td>
</tr>
<tr>
<td>Number of Pages</td>
<td>37 pages + 1 appendix</td>
</tr>
<tr>
<td>Date</td>
<td>24 November 2015</td>
</tr>
<tr>
<td>Degree</td>
<td>Bachelor of Engineering</td>
</tr>
<tr>
<td>Degree Programme</td>
<td>Information Technology</td>
</tr>
<tr>
<td>Specialisation option</td>
<td>Smart systems</td>
</tr>
</tbody>
</table>
| Instructor(s) | Joseph Hotchkiss, Project Manager  
|             | Keijo Länsikunnas, Principal Lecturer |

The goal of the project was to design and develop a sensor platform board which would be used to train new students on embedded programming. The students would be able to program the Pyboard and access all its timer, serial buses and various other embedded topics. They would be able to learn the procedure and functionality of Internet of Things (IoT) and design a sensor platform that could be used in the greenhouse environment.

The board would use a microcontroller platform where all the sensors would be interfaced. The data manipulation, calibration and packet formation would take place in the microcontroller. Once the data was ready to be transmitted, it could be sent wirelessly through Bluetooth Low energy protocol which could be received by BLE client devices. The data received in the client device can be used to analyze environmental conditions and act accordingly remotely.

The goal of the project was achieved successfully. The sensor platform board is being used at Helsinki Metropolia University of Applied Sciences to train the students on IoT and embedded programming. The students are using the Python programming to get the reading from the sensors integrated on the sensor platform board. In addition to this, they are connecting additional sensors through the RJ12 ports and pin headers available on the board. The available sensors are mostly used to study the greenhouse environment.

Keywords | sensor, sensor platform, Bluetooth Low Energy, PSoC, PSoC BLE
Contents

1 Introduction 1

2 Theoretical Background 3

2.1 Pyboard 3
   2.1.1 Features 3
   2.1.2 Powering Options 4
   2.1.3 Control the Pyboard 5

2.2 Sensors 5
   2.2.1 Temperature Sensor 6
   2.2.2 Ambient Light Sensor 6
   2.2.3 Humidity Sensor 8
   2.2.4 Soil Moisture Sensor Module 9
   2.2.5 PH Electrode 10
   2.2.6 Carbon Dioxide Sensor Module 10

2.3 DC Relay 12
   2.3.1 Electromechanical Relays 12
   2.3.2 Solid State Relays 13
   2.3.3 MPDCD3 14

2.4 Step-up Circuit 15

2.5 Bluetooth Low Energy 16
   2.5.1 BLE Frequencies and Hopping 16
   2.5.2 BLE Device Roles 16
   2.5.3 BLE Protocol 17

3 System Design and Implementation 19

3.1 Analog Sensors Design and Implementation 19
   3.1.1 Temperature Sensor Circuit 19
   3.1.2 PH Probe 20
   3.1.3 Humidity Sensor Circuit 22

3.2 Digital Sensors 23
   3.2.1 Ambient Light Sensor 23
   3.2.2 Soil Moisture and CO2 Sensor Connector Module 25

3.3 PSoC BLE 25
   3.3.1 BLE Stack 26
   3.3.2 Connecting with Pyboard 27
3.3.3 Getting Data from Pyboard 28
3.3.4 Interrupt and Sending Data 29
3.3.5 Data Reception and Conversion 31
3.3.6 Raspberry Pi BLE Connection 32
3.4 Measurement Board 33
3.4.1 Schematic Design 33
3.4.2 PCB Layout 35

4 Result 36

5 Drawbacks and Solutions 37
5.1 PH Probe Error and Solution 37
5.2 PsoC BLE Data Error and Solution 37
5.3 Humidity Sensor Error and Solution 37
5.4 Raspberry Pi data Error and Solution 39

6 Conclusion 40

References 41

Appendices
Appendix 1. C-code for PsoC BLE
<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Alternating Current</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog to Digital Convertor</td>
</tr>
<tr>
<td>ARM</td>
<td>Acorn RISC Machine</td>
</tr>
<tr>
<td>BLE</td>
<td>Bluetooth Low Energy</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analog Convertor</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>GAP</td>
<td>Generic Access Profile</td>
</tr>
<tr>
<td>GATT</td>
<td>Generic Attribute Profile</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input and Output</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>SPI</td>
<td>Service Provider Interface</td>
</tr>
<tr>
<td>REPL</td>
<td>Read-eval-print Loop</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver and Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
</tbody>
</table>
1 Introduction

The advancement in technology has made life much easier for people in their daily work. People want to know, monitor and control the devices or things being in another part of the world. The devices, sensors or things in a network that can exchange data are called the Internet of Things (IoT). Embedded devices are being installed in many environments to monitor space through the network. With the same principle the greenhouse environment can also be monitored and controlled over the network.

With the development in technology and the Internet of Things (IoT), the development in the education system has been an integral part of this generation. Thus, a Sensor board for IoT training project was designed to train new students about the embedded systems and the Internet of Things. The goal of the project was to create a sensor platform board that could be interfaced with micro Python platform called the Pyboard. Further the data collected on the Pyboard are sent through a universal asynchronous receiver and transmitter (UART) to the PsoC BLE and wirelessly using Bluetooth Low Energy (BLE) protocol. The final product after the project was aimed at being used by the first year students in the new course called Devices which could be programmed to monitor the greenhouse environment.

The board consists of two major parts, a Pyboard interface and PsoC BLE. The Pyboard part of the device consists of three sensors, Ambient light, Temperature and Humidity sensor, three RJ12 ports consisting of 3.3V, ground and 4 GPIO pins, one 10 pin header which also consists of 7 GPIOs, 3.3V and a ground and two relays all interfaced with Pyboard. The sensors data could be collected and processed using their respective protocol programming through Pyboard. Further the data collected was sent serially to the BLE dongle. The data was a frame of 20 bytes with a fixed identifier of each sensor. Meanwhile the PsoC BLE part was designed and programmed to receive the data from the micropython and to send it to other BLE devices.

The project was carried out by three people, namely Kyoungwon Kim, Hyewon Jeon and me. The project was supervised by Keijo Länsikunnas and managed by Joseph Hotchkiss. Kim was assigned to program the PsoC BLE dongle, CO₂ sensor and PH Probe. Hyewon programmed the temperature sensor and humidity sensor. I was responsible for
designing the printed circuit board (PCB) for the sensor platform and programming the ambient light sensor as well as designing the circuit for all the sensors.

Successful design and implementation of the board would be a good learning environment for the students of embedded systems. In addition to this, this board can also be used to create automation in the greenhouse environment.
2 Theoretical Background

The board consists of two parts: the microcontroller unit and the BLE unit. The Pyboard is the heart of the board where all the sensors and components are interfaced. The Pyboard can be programmed using the micropython programming. It controls all the GPIOs and the Relays. It also acts as the transmitter for the UART to send the data collected from the sensors to the PsoC BLE. The PsoC BLE is responsible for collecting the data from the Pyboard and send it wirelessly through the Bluetooth Low Energy protocol. In this chapter the working principle and the physics behind the components will be explained.

2.1 Pyboard

Pyboard is the electronic circuit board which uses micropython to run electronic development projects. Micropython is the optimized Python 3 programming language that can run on a microcontroller. The board consists of “pyb module”, which contains all the Pyboard specific classes and functions. All the functions and API’s to program the board can be imported from “pyb module”. It can be connected to a PC using the micro USB cable.

2.1.1 Features

The Pyboard is built in with a 168 MHz Cortex M4 CPU with ARM cortex based STM32F405RG microcontroller. It consists of 1024 KB flash memory with 192 KB of RAM. The board also consists of a Micro SD card slot, which can be used as the primary memory to program the board. The board also consists of an accelerometer and LEDs which are simpler to program and test at the same time without any additional components.
As shown in figure 1, the board has 29 GPIOs, which consists of three 12-bit ADCs, two 12-bit DACs, UART, I2C and SPI buses. It also consists of the pins for real time clock backup battery, one reset and one user switch. The board works with input voltage range from 3.6 volts to 10 volts. The board provides 3.3v-regulated output with up to 300mA current. With its all the powerful features, this electronic board is easier to build, train and design projects relating to smart systems. (Micropython.org, 2015)

The board consists of six UART channels, two SPI channels, two I2C channels, two CAN bus channels and DAC and ADC channels. Further it also consists of timer that can perform PWM, output compare and input capture.

2.1.2 Powering Options

The Pyboard can be powered by two different ways. It can be powered through a USB cable from a computer or a power source to its micro USB terminal. However it can also be connected through batteries in its $V_{IN}$ pins. While powering the device we need to keep in mind the voltage that it requires to power on the device in both cases. It can
power from 3.6V to 10V DC. Thus while suppling the power through USB the power supply will be 5V as USB nominal voltage is 5V however while suppling through batteries we need to make the supply voltage in the range of 3.3V to 10V.

2.1.3 Control the Pyboard

The Pyboard can be controlled in three different ways. The Pyboard can be connected to the computer using the USB which will appear as a com port in the computer. It can then be used on terminal or any serial program to get to the python REPL. This helps us to execute python programs or commands through the REPL prompt.

In addition to this the board can be controlled by changing the REPL to raw REPL mode by typing CTRL+A from the keyboard. In raw REPL mode we can order the Pyboard to execute python script immediately.

However the most efficient way of using or controlling the board is through the file system of the Pyboard. The Pyboard has built in flash memory which can be extended connecting a SD card on the SD card slot of the board. Then, we can than make a main.py file and write our script on the file. The script on each reboot or boot executes the script. (Micropython.org, 2015)

2.2 Sensors

According to the Oxford Dictionary, “Sensors are the devices which detects or measures a physical property and records, indicates, or otherwise responds to it.” (Oxforddictionaries.com, 2015) There are various kinds of environmental phenomenon in the ecosystem. In order to sense that phenomenon and convert that to the electronic format, sensors were designed. The output from the sensors are basically electrical signals converted from the environmental factors like heat, light, motion, pressure, moisture, etc.

Depending on the output of the sensors the sensors are divided into two different categories, analog and digital sensors. The sensors that output the electrical signals in the form of voltage without modifying the signals are the analog outputting sensors. The analog signals vary in the range with slight change in the input. However the digital sensors are the sensors that discrete signal meaning the change in value in steps which is within the output range of the sensor. The digital signals preceding value can be known as it changes in steps.
Due to the demand of sensors in the greenhouse ecosystem and specification of the project we chose the six different sensors. In a greenhouse it is necessary to know the temperature, ambient light level, humidity, soil moisture, CO2 level and PH value of the liquid fertilizers used. Thus the required sensors were ordered and will be discussed in this section.

2.2.1 Temperature Sensor

As temperature measurement is the essential part in any kind of environment, and especially talking about the greenhouse sensor platform, it was essential to integrate a temperature sensor in the sensor platform board. It helps to study the temperature readings in addition we can also analyze the performance of other sensors connected on the board depending on the temperature readings as the output of those sensors are temperature variant.

Thus KTY81-210 temperature sensor was used due to its capability to measure wide range of temperature variant. It can measure from $-55^\circ C$ to $155^\circ C$. As the sensor is resistive, it is easier to design and integrate the circuit to the board. The High accuracy, fail-safe behavior, and linear characteristics makes the sensor reliable to use. (KTY81 Series, 2008)

2.2.2 Ambient Light Sensor

The board also consists of an ambient light sensor. The TSL2250 ambient light sensor produced by Texas Advanced Optoelectronic Solutions, converts light to the digital output voltage. The sensor consists of a 12 bit ADC convertor which helps us to read the digital output. This makes us easier to integrate the sensor output directly to the microcontroller. Further the sensor has the ADC and two photodiodes integrated in a single complementary metal oxide semiconductor (CMOS) as shown in figure 2.
The TSL2250 operates with the single supply voltage of 2.7v to 5.5v. It can be programmed as the two wired SMBus interface. SMBus is the subset of I2C Bus where the only difference is that the SMBus operates between 10 KHz to 100 KHz clock rate. As this sensor is designed for broad wavelength lights, it has two channels. One of the channel is responsible to sense visible and infrared light and the other is designed to sense only the infrared. The output of both the sensor are used to calculate the ambient light in the surrounding in the standard unit of lux.

This sensor is mostly used in the electronic devices where ambient light data is used to control the functionality of the devices. For example they are used in the laptop computers, GPS systems etc. where the backlighting is controlled using the ambient light readings. (TSL2250 Ambient light sensor with SMBus interface, 2003)

The integrated ADC integrates a photodiode current. Initially it integrates photodiode current in channel 0 followed by channel 1. At the end of the operation, the digital value converted from the diode or the current from the photodiode is transferred to the respective channel ADC register. As the transfer of the data is saved in multiple buffers, no any wrong data is read during the transfer. After one cycle is completed, the sensor initiates another cycle automatically. The final data stored in the ADC registers can be read and controlled using the SMBus interface.
The sensor has an 8 bit-command register that can be controlled through the SMBus. The slave address for the sensor is 0111001 which is wired internally so we have only one device per bus. There are different functions that can be used to control the sensor SMBus protocol. The sensor can be read in different extended mode and standard mode. While the sensor is operating in the standard mode, the time for the full cycle is 800 mS, 400 mS for each channel whereas the use of extended mode takes less time by a factor of 5 while decreasing the sensing capability by 5 times. However the extended range helps to measure the higher light levels.

2.2.3 Humidity Sensor

Due to its cheap price and application on various places like air conditioner, humidifier, hygrometer, etc. we used the HCZ-D5-A sensor produced by Multicom. The electrodes are made with lead free solder. The sensor behaves as a capacitive sensor along with its resistive properties. The actual sensor is put in a case with minimal effect in the humidity reading.

The sensor changes its impedance with change in the humidity level. As the environment gets humid, the impedance decreases or in other words humidity level is inversely proportional to the impedance of the sensor. The impedance varies from 6.3 MΩ to 2 KΩ from dry to wet humidity. This sensor can measure between 20% RH to 90% RH humidity level. However in office environment RH very seldom goes below 40%.
HCZ-D5-A operates at 0.5 to 2 KHz at $1V_{\text{rms}}$ or 2.8V peak to peak at maximum. However the readings varies with the difference in temperature in the surrounding. Figure 3 shows the relative humidity at different temperatures measured at 1 KHz, $1V_{\text{rms}}$. As shown in the figure the impedance decreases rapidly with the change in relative humidity however it also depends on the different temperature level. (Humidity Sensor Type HCZ-D5-A, 2011)

2.2.4 Soil Moisture Sensor Module

The soil moisture sensor module that could be used to measure the soil's moisture in the green house was ordered from an online shopping site, eBay. According to the little documentation provided on the site, the soil moisture sensor is a water sensor that detects the amount of water present in the soil. In other words it measures the moisture level in the soil.

The module operates with the voltage range between 3.3V to 5V. It consists of a LM393 chip for comparator circuit. It consists of two outputs, Digital and Analog output. The
digital output is high if the soil is wet and low when it is dry which depends on the threshold value which can be adjusted using the potentiometer available in the module. In the meantime the analog output gives the AD converted value in voltage which can be used to get precise value of the soil moisture. (EBay, 2015)

2.2.5 PH Electrode

PH Electrode is a sensor that is used to measure the PH value of a chemical or any fluid. It measures if the chemical is either acidic or basic. In the greenhouse it can be used to measure the acidic or basic content of the liquid fertilizers that can be used for the plants in order to know if they are harmful for the plants.

Produced in china this sensor is accurate and gives reliable reading instantly. The sensor has the internal resistance of less the 250 MΩ with the response time of less than 1 minute. The sensor works in the temperature range between 0-60°C. The sensor is connected with the BNC connector plug where the readings can be taken from. The PH probe works with the reaction or activity of the hydrogen ion in the chemical. (EBay, 2015)

2.2.6 Carbon Dioxide Sensor Module

Produced by Sandbox electronics, the CO2 sensor module uses MG-811 sensor to sense the amount of CO2 present in the air. The MG-811 sensor is used in air quality measurement and control applications. Thus this sensor was selected to be use in the greenhouse project.
As shown in figure 4, the module works with the supply voltage (VCC) of less than 5.5V. The sensor must be heated with the heating power supply (HEAT) of 7.5 to 12V. The heating current is about 200mA and heating power 1200mW. The operating temperature range of the sensor is from -20°C to 50°C.

The MG-811 sensor is a cell structured 6 pins module that has a very small output range with the change in CO2 level, i.e. 100mV to 600mV. As the output signal is very small, it needs to be amplified in order to have clear readings. Thus the sensor module consists of signal conditioning circuit and the heating circuit as shown in figure 4. The signal conditioning circuit amplifies the output voltage or signal whereas the heating circuit keeps the sensor heated for reliable and accurate readings.

The sensor’s output at 400ppm in clean air is from 200mV to 600mV which is known as zero point voltage (V₀). The voltage drops with the increase in the amount of CO2 concentration. The amount of CO2 can be calculated using the common logarithm formula as shown in equation 1:

$$V_s = V_0 + \Delta V_s/\left(\log_{10}400 - \log_{10}1000\right) \times \left(\log_{10}CO_2 - \log_{10}400\right)$$ \hspace{1cm} (1)

Where, $\Delta V_s = sensor \ output \ at \ 400ppm - sensor \ output \ at \ 1000ppm$
The gain of the signal conditioning circuit is 8.5. Hence the output range of 100mV to 600mV will be magnified to 850 mV to 4.8V which is easier to read using a microcontroller. (Sandboxelectronics.com, 2014)

2.3 DC Relay

Relay is a kind of electronic circuit that behaves as a switch which controls a high load with a small signal. DC Relay controls the higher potential DC components connected to its terminal with a small voltage signal. There are two different types of relays with their own advantages and disadvantages.

2.3.1 Electromechanical Relays

The relay where magnetic force switches the load are electromechanical relays. These relays are also called coil based relays as they consist of coil for energizing circuit or to create magnetic effect in the circuit when powered. The electromechanical relays consists of two different circuits: energizing circuit and contact circuit. The coil in the energizing circuit creates magnetic field when current flows through the coil. Once magnetic field is created in the coil, it attracts the metal ferrous plate from the armature as shown in figure 5. Once the armature makes contact with the contact plate, it switches the circuit. (Galco.com, 2015)
The electromechanical relays are capable of switching wide range of loads up to their maximum voltage rating. They do not require heatsinking as the contact resistance decreases with the increase in load. They require a large amount of coil power to switch however they can operate at high amount of load at different temperatures. (Electronicdesign.com, 2015)

In contrast to this, electromechanical relays have a fixed lifetime. They wear out with time and the major failure is coil failure. The contact and the piston in the armature also gets damaged after some time if huge load is contacted to it for a longer period of time. In addition to this these relays also has high amount of interference. The electromagnetic sensitive devices will be affected by the relays in the circuit. (Electronicdesign.com, 2015)

### 2.3.2 Solid State Relays

The relays that switch the loads with the electrical circuit are solid state relays. It has the input circuit, control circuit and output circuit integrated in the relay. The input circuit behaves as a coil in the electromechanical relays. The circuit turns on when enough
operating voltage is applied through the input terminals of the relay. Similarly the circuit switches off when the voltage is low than the specified minimum voltage. In addition to this, the output circuit switches on the load when the control circuit is energized and turns it back off when it is not. (Galco.com, 2015)

The advantage of solid state relay is that it doesn’t require external circuit to integrate it in an electronic platform. In addition to this these relays do not have interference to other components compared to the electromechanical relays. The life span of this relay ranges from 100 thousand to 500 thousand cycle and gets worn out when its contact pin gets burned out. (Galco.com, 2015)

However it is difficult to find out if the solid state relay is operating well depending on its performance with the load. It has to be replaced when its contact pins burn out. (Electronicdesign.com, 2015)

2.3.3 MPDCD3

As MPDCD3 solid state relay produced by Crydom, can switch higher DC loads with a small signals without any additional circuits, it will be better be used in the sensor platform board.

As shown in figure 6, MPDCD3 relay we don’t need any additional components wiring. It can operate from 3V to 60 V with a maximum load current up to 3A_{rms} and minimum 20mA_{rms}. (PCB Mount MP Series, 2014) DC load is also inductive load, hence it has to
be suppressed with diode. The current flowing through the inductive load, it stores energy in the magnetic flux. Once the relay is off, the magnetic flux fails and the energy stored creates electro-magnetic force with polarity in the direction of existing current flow. If there is no way for the inductive low current to flow it develops a high voltage that breaks the component in the circuit including the relay. Thus, it is safe to use a diode to allow this current to flow.

2.4 Step-up Circuit

As the board was required to be powered by 2AA batteries or 2.4V and the components in the board required 5V to be powered on, we had to step up the supply voltage to 5V. Hence we chose LM2621MM low input DC to DC step up regulator to step up from 2.4 to 5V.

![Application circuit for LM2621mm](Modified from LM2621 Low Input Voltage, Step-Up DC-DC Converter, 2005)

LM2621mm can operate with the operating voltage between 1.2V to 14V and regulates it to required output voltage. It has high switching frequency which is up to 2 MHz that can be adjusted by varying the R_{FQ} as shown in figure 7. As the datasheet suggests to use the frequency between 300 KHz to 2MHz, a resistor of 150KΩ was used to create a frequency close to 900 KHz. The output voltage can also be adjusted by varying the R_{F1} and R_{F2} in the circuit which can be calculated by using the formula as in equation 2.

\[ RF2 = RF1 / [(VOUT / 1.24) - 1] \]  

(2)
The output voltage varies between 1.24V to 14V depending on the value of the resistors connected in feedback resistive divider consisting of $R_{F1}$ and $R_{F2}$. Typically $R_{F1}$ is selected to be 150KΩ as suggested by Texas Instruments and $R_{F2}$ can be calculated using the equation. (LM2621 Low Input Voltage, Step-Up DC-DC Convertor, 2005)

2.5 Bluetooth Low Energy

Bluetooth low energy (BLE) or Bluetooth Smart is the Bluetooth 4.0+ version of Bluetooth that is designed to be used in the electronics devices in IoT. The Bluetooth low energy devices are power efficient or in other words they can operate with little power for long period of time. In sleep mode a BLE device uses around 4uA current. It helps the developers to create a device that can communicate with every Bluetooth low energy enable devices. (Bluetooth.com, 2015)

BLE was developed by Nokia which was named as Wibree during its development. It wasn’t initially included in Bluetooth 4 version because of its new protocol and new architecture varying with Bluetooth architecture. However because of its compatibility with classic Bluetooth on hardware level it was included in Bluetooth version 4 and named as Bluetooth Low Energy. (Vesanen, 2015)

2.5.1 BLE Frequencies and Hopping

BLE makes the use of adaptive frequency hopping meaning they do not overlap with other frequencies like Wi-Fi frequencies decreasing the interference thus limiting connection setup time and collisions probabilities. It uses predefined advertisements channels on 2.4GHz band which is license free. The data transfer channel is selected from the channel that has the least interference. The BLE does the frequency hopping from channel to channel with slower speed than Bluetooth classic. (Vesanen, 2015)

2.5.2 BLE Device Roles

The BLE devices act themselves as broadcaster, observer, peripheral or central devices. The broadcaster doesn’t accept any connections however transmits data constantly on its surrounding range. In the other hand the observer listens to BLE devices passively. However the peripheral devices advertises, connects and transmits data on each request. In addition to this the central BLE devices connects to peripheral devices, requests
and gets the data from the devices. Mostly the device with low powered or battery powered are broadcaster or peripheral devices and the rest are observer or central devices for example, smartphones and Linux devices.

The broadcaster or peripheral devices send advertisements containing the mac address and its name at intervals that is pre-defined. The interval can be adjusted while programming the BLE device which can vary from milliseconds to minutes. However when the broadcaster and observer involve themselves, the broadcaster as described earlier send advertisement on all the advertisement on the defined interval. The observer scans the channels and tries to find out on which channel the broadcaster is broadcasting depending on the received advertisements.

In the context of connection between peripheral and central, the central receives the advertisements and requests for the connection with the information about the channels to be used and the hopping sequence. Once the peripheral device connects with the central device, it stops advertising and remains connected and stays awake until the timeout defined on the central device. (Vesanen, 2015)

2.5.3 BLE Protocol

At a central device is connected with only one peripheral device. Once a connection is made, the standard Bluetooth low energy protocol is used in order to make connection and exchange data.
As shown in figure 8, the lowest unit is the radio frequency unit which is named as base-band. Above it lies link layer that is responsible for switching the channels or hopping the channels. L2CAP layer gets the data from GAP and GATT profile and provides it to the link layer. The GAP profile or generic access profile is used to configure the device mode, advertisement packets and timing for the advertisements and broadcasts them. ATT or attribute protocol connects or pairs two BLE devices. The most important part in BLE protocol is the GATT or generic attribute profile that handles the communication between two low energy devices. GATT profile is used to set the list of attributes which is provided by the peripheral devices to the central devices which specifies what can be read and written through the BLE device. Depending on the BLE device, it provides various types of attributes which simplifies to create a GATT service on the Bluetooth device. (Vesanen, 2015)
3 System Design and Implementation

As the main goal of the project was to create a sensor platform that could be programmed using the Pyboard and the PsoC BLE for the radio communications, all the components discussed in section 2, had to be designed and tested electronically. First all the circuit were designed and tested for all the components and finally integrated in a board to meet the goal of the project. The board only consists of Ambient light sensor, Temperature sensor, PsoC BLE and relays. All the other components will be connected through add on boards in order to make the measurement efficient and reliable. Under this section we will discuss all the procedures and design of each sensors and part of the board.

The temperature sensor, humidity sensor and the PH probe were the analog sensor which required designing the circuit for each sensors. However the Digital output sensors CO2, Soil Moisture and Ambient light did not require any external circuit except the ambient light sensor required a pull up resistor.

3.1 Analog Sensors Design and Implementation

The analog sensors require the additional circuits for the output to be in range and to function properly. The input voltage for the analog sensors depend on the datasheet provided by the manufacturer. The circuit for the sensor depends on the properties of the sensor. As the Pyboard ADC cannot read the negative part of the analog signal the output has to be rectified or converted into the format it can read. Hence each of the sensors are designed with different approach depending on their respective properties.

3.1.1 Temperature Sensor Circuit

As the temperature sensor was resistive, we had to design a circuit so that we would be able to find the resistance value of the sensor at different temperatures. The found resistive value could be matched with the value in the datasheet and the actual temperature value in degree Celsius or Fahrenheit could be measured.
As the voltage divider drops some voltage across a resistor, the voltage dropped across the temperature sensor can be measured using the microcontroller ADC channel. In figure 9, the circuit is provided with the supply voltage of 3.3V. The R2 or 1.7KΩ resistor drops voltage depending on the changing resistance of the temperature sensor. The resistance value of the temperature sensor is directly proportional to the temperature. Thus with the increase in resistance value the voltage output across X1 also changes. Thus the read voltage can be converted to the corresponding resistance using the voltage divider in equation 3.

\[ X1(Vout) = \frac{P3}{(R2 + P3)} \times 3.3, \text{ where } R2 = 1.7\,\Omega \]

Hence, \[ P3 = X1 \times R2 / (3.3 - X1) \]

The temperature that correspond the acquired resistance implies the temperature value.

3.1.2 PH Probe

PH probe measures the amount of hydrogen ion and drops the voltage across it depending on the chemical. The output of the BNC connector of the PH probe can be connected to the 10 pin header ADC channel and read using ADC protocol on the micropython. As
the PH probe was supplied without the datasheet, it was needed to be tested with different chemicals and figure out the equation for using the PH probe output and convert it to the human readable standard unit PH level.

<table>
<thead>
<tr>
<th>Chemicals</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>58-66</td>
<td>53-59</td>
<td>49-59</td>
<td>57.333</td>
</tr>
<tr>
<td>Vinegar</td>
<td>180</td>
<td>206</td>
<td>195-206</td>
<td>195.5</td>
</tr>
<tr>
<td>Coffee</td>
<td>83-86</td>
<td>94</td>
<td>95</td>
<td>91.17</td>
</tr>
<tr>
<td>Salt water</td>
<td>58-62</td>
<td>54-62</td>
<td>58-66</td>
<td>60</td>
</tr>
<tr>
<td>Baking soda water</td>
<td>54-64</td>
<td>49-54</td>
<td>46-58</td>
<td>54.83</td>
</tr>
<tr>
<td>Soapy water</td>
<td>33-44</td>
<td>29-37</td>
<td>29-36</td>
<td>34.67</td>
</tr>
</tbody>
</table>

Table 1 shows different voltage output in mV of the probe at different chemicals. There were 3 readings taken and the average was taken out from the three readings for each chemical. Further the graph was plotted using the average value of the output.

Figure 10. PH probe voltage output in mV vs Typical PH value
As shown in figure 10, the output is not as linear as it has to be because it was difficult to get the chemical of all the PH value. With the chemicals we had and studying the curve we figured out that the curve is linear. Using linear regression, we have figured out a formula to calculate the PH value with the data output. By using the linear regression taking the data obtained from the experiment, we were able to get equation 4.

\[ Y = 12.5 - 0.0561X \]  

*Where \( X \) is the value from PH probe and \( Y \) is the expected PH value.*

### 3.1.3 Humidity Sensor Circuit

The humidity sensor had to be created in a breakout or add-on board because the design and experiment of the sensor was delayed in comparison with the main sensor board platform. As explained in section 2 the humidity sensor consists of capacitive and resistive features. It works with \( 1 \text{V}_{\text{rms}} \) at 1 KHz. Hence we need to use DAC from the micropython to create a sine wave equal to the required power supply.

![Humidity sensor circuit for J1 Connector](image.png)

As shown in figure 11, the DAC channel 1 pin X5 of the micro python will be connected to the circuit through RJ12 connector. However the signal produced from the Pyboard contains DC level which is removed using the C1 capacitor with the value of 100nF. The signal is than applied to the circuit through R2. The voltage divider equation can be used to calculate the impedance across the humidity sensor C2. The TLC2262CP because of
its low input bias current of typical value of almost 1pA, amplifier with the diode is used as a half wave rectifier which along with R4 and C4 converts it to the DC voltage that can be read and converted into the impedance in the micropython. The resistor R1 of 10KΩ helps to minimize the effect of negative bias current created by the amplifier.

The sensor is tested with the input voltage of 3.2V peak to peak sine wave. Hence when the sensor is dry, the output at the input of the op-amp is 1.2V peak. However the output is sinusoidal which needs to be converted using the half wave rectifier. The output at Y11 is 1.2V DC. In addition to this when the sensor is wet the output voltage is measured to be as low as 54mV.

The impedance of the sensor can be calculated using the voltage divider equation in 5.

\[ V_{out} = \left( \frac{C2}{C2 + R2} \right) \times VIN \]  
\[ \text{Where } R2 = 590K\Omega, VIN = 1.6V \]  

In addition to this, the breakout board also consists of 3 pin header with the Pin out of 5V, Y12 ADC and GND Pin. The main aim of these pins are to connect CO2 sensor along with the Humidity sensor.

3.2 Digital Sensors

Digital sensors give the output in the form of discrete signals. These kinds of sensors have the integrated circuitry to convert the analog signals into the digital output. The MG811 sensor module and the soil moisture consists of all the circuitry so we can get the data output in digital format. Further the data from the ambient light sensor can be read using the Smbus protocol.

3.2.1 Ambient Light Sensor

As the ambient light sensor follows the smbus protocol which resembles the I2C circuit design, the pinout from the ambient light sensor had to be connected following the I2C protocol. The Vcc pin was connected to 3.3V and pin 4 was grounded. The data and the clock pin from the sensor had to be connected to the respective pins on the microcontroller.
In figure 12, the SCL pin 5 from the sensor is connected to the SCL pin X9 of the microcontroller and SData pin 8 from the sensor is connected to the SData pin X10 of the Pyboard because the pins X9 and X10 are the I2C channel 1 pins.

Once the protocol is followed and the data received, the ADC data can be converted to the standard unit lux format. The received byte consists of seven data bits and 1 valid bit.

Table 2. ADC data format (Modified from (TSL2250 Ambient light sensor with SMBus interface, 2003))

<table>
<thead>
<tr>
<th>Valid</th>
<th>Chord Bits</th>
<th>Step Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>B7</td>
<td>B6</td>
<td>B5</td>
</tr>
<tr>
<td>Valid</td>
<td>C2</td>
<td>C1</td>
</tr>
</tbody>
</table>

Table 2 shows the data format of the ADC which represents the photo detector current in its 7 bit and 8th bit explains if the data is written in the ADC or not. The valid B7 bit has a value of 1 when the ADC has completed writing the value on the channel data register. Bits B6 to B4 together represent the chord number and the B3 to B0 represent step number which are used later during the conversion of data to lux.
Further the ADC count value needs to be calculated for each channel using the equation 6.

\[ \text{ADC Count Value} = (\text{INT} (16.5 \times (2c - 1)) + (S \times 2c)) \]  

Where \( c \) is the chord number from B6 to B4 and \( S \) is the step number from B0 to B3.

The chord number value varies from 0 to 7 and step number varies from 0 to 15. Hence the amount of light in lux can be calculated using the formula in equation 7.

\[ \text{Light level (lux)} = \frac{\text{Ch0 Counts} \times (0.46) \times (e^{-3.13R})}{\text{Ch1 Counts}} \]  

Where \( R = \frac{\text{Ch1 Counts}}{\text{Ch0 Counts}} \)

Ch0 counts in the above equation means the ADC count value of Channel 0 and Ch1 counts means the ADC count value of channel 1.

Thus at the completion of the sensor board user can directly write a program to read the sensor data following the Smbus protocol and convert it to lux using the above equations. (TSL2250 Ambient light sensor with SMBus interface, 2003)

3.2.2 Soil Moisture and CO2 Sensor Connector Module

The soil moisture and the CO2 sensor both have 4 pin out. Hence a breakout pin with all 6 pin out connecting through pin header was created for the sensors. This helps to connect the sensor to the RJ12 ports J1, J2 or J3. If we need to connect the CO2 sensor with this breakout board we need to connect it with the J3 connector in the measurement board.

All the RJ12 connectors have ADC pins which can be used to read the voltage value out of the sensor. The digital pins can be connected to any of the GPIO pins in order to read the digital output of the sensor. The conversion can be done using the datasheet of the CO2 sensor and the soil moisture sensor respectively. Furthermore sensors that can be powered with either 3.3V or 5V can be connected and programmed through the board connected via RJ12 ports.

3.3 PsoC BLE

CY8CKIT-142 PsoC 4 BLE Module is designed and developed by Cypress semiconductors that can help developers integrate the BLE stack in IoT. It consists of a PsoC 4 BLE Device, two crystals, an antenna and other GPIOs.
It is an ARM Cortex –M0 based single-chip embedded solution that can be integrated in electronic projects with the help of programmable analog and digital peripherals. (Cypress.com, 2015)

As discussed in section 2.10 of this documentation, the PsoC BLE follows all the BLE protocol for communication. It supports all the broadcaster, observer, peripheral and central roles. It can act as a master or a slave device.

3.3.1 BLE Stack

The PsoC BLE uses the Bluetooth 4.1 Low energy functionality to implement the BLE stack. The stack is integrated inside the component. The BLE stack combines the BLE protocol introducing some other more layers as shown in figure 13.

![BLE Stack](image)

*Figure 13.PsoC BLE stack*

The lowest layer is the physical layer which corresponds to the radio frequency unit layer in the BLE protocol. Similarly the Link layer organizes BLE connections among the devices. The host controller interface allows the user to access the upper layer from the link layer. In addition to this the L2CAP layer is responsible to provide the communication in a free channel. It provides or manages three channels using the channel multiplexing where ATT and SMP uses two channels and one by the protocol for its own purpose. The security manager protocol (SMP) provides the security features to the BLE device. It encrypts the data, controls the pairing with the fixed device and creates a unique key for the identification of the device. However the connection option with the pair code or
without the code depends on the GAP profile of the central and peripheral devices. Contradicting most of the devices the GATT profile is set as a server or a client where the attributes are broadcasted or exposed using the ATT profile.

All the attributes and configuration of the GATT and GAP profile can be performed using the PsoC creator. The PsoC consists of custom characteristic and pre-defined characteristics for the BLE component which makes it easier to be programmed as a server. (Bluetooth Low Energy, 2014)

In this project, we have used PsoC 4, in order to use low-powered wireless systems. We use PsoC for receiving data from Pyboard through UART. Using Bluetooth low energy, data packets sent from Pyboard can be forwarded to other devices. In our case, that other device is Raspberry Pi.

PsoC creator was used to program the BLE component following the API’s provided on the datasheet. The BLE protocol was followed in order to configure the advertisement packets, characteristics and other attributes for the BLE network. Once the BLE component was dragged and put in the design, the configuration dialogue box appears. The GAP profile can be configured and the advertisements packets can be configured. The GATT profile can be edited from the GATT tab. The device can be chosen to be either peripheral or central or both. Further it can also be configured as either GATT server or GATT client. As our device is responsible for sending the data to other device we configure it as server. In addition to this we can configure the MTU size or Maximum transmit unit from 0 bytes to 512 bytes. However we need to consider the MTU size the client can handle to receive or transmit during the BLE communication.

The PsoC BLE is powered using the 3.3V on all its VDDD, VDDR and VDD pins through the Pyboard. Pyboard is connected to the P0.0 and P0.1 of PsoC board to the UART 6 of the board. (Cypress.com, 2015)

3.3.2 Connecting with Pyboard

In UART communication, PsoC is in charge of receiving and the pin for RX on PsoC is P0[0]. Once the correct TX pin is connected on the Pyboard side with the RX pin on the PsoC, UART is set and data packets can be sent.
3.3.3 Getting Data from Pyboard

The data packets sent from Pyboard has fixed forms and with PSoC, those packets are filtered to check if there is any wrong data. PSoC is continuously waiting for any data packets to be sent from Pyboard and when it gets a data packet, it first checks if the indicator is there at the first byte. Then, there is temp1 [20], temp2 [20], temp3 [20] arrays, each of them for each data packet with three different indicator, to store rest of the data bytes.

The figure shown below is the code for the data packet with indicator 1. It is checking if the packet has the CO2 and PH sensor data, which are the sensor data that should be sent through packet number 1.

Lastly, every data packet ends with ‘+’ and using this indicator PSoC decides to stop reading the data from Pyboard. Before PSoC decides to send the data, it checks whether the data packet contained all the sensor data that it was supposed to have and check indicator array is used for this purpose. If the packet that has been received does not contain all the sensor data that it was supposed to have then variable send becomes 1. The variable is used when sending the data from PSoC to the connected device. It does not occur the interrupt to send the data to the device unless the value of send is 1 as shown in the code in figure 14. (Cypress.com, 2015)
3.3.4 Interrupt and Sending Data

Once, the received data is ready to be sent, UART buffer in between PsoC and Pyboard gets cleared. Hence delay has to be introduced. In order to collect and calculate all the data, there needs to be delay between packets. Giving delay can be done with the .delay (milliseconds) function in “pyb” library. If we give too small delay, we won’t get any packets from PSOC since it takes some time to do calculate on the Pyboard side. We will not be able to get the data also when delay is big.

In this project, we have the limitation with the packet length of 20 bytes. It is possible to send more than 20 bytes, however, since we are using Raspberry Pi as the device that receives the data from PsoC, we could not have data packet with more than 20 bytes. It
is the problem with reading data from Raspberry Pi side. However, with CySmart application that Cypress provides, we can get the result of getting more than 20 bytes. Hence the data that needs to be sent through the Pyboard has to be formatted on the Pyboard in the format discussed below as the data send from the Pyboard to PsoC will be sent as it is through the PsoC BLE.

As the 6 sensor data cannot be sent all at once and need to be divided into several packets. There are sensors that give output with long decimal numbers and for our convenience, we need to modify our sensor output so that it will have 2 decimal digits at most. Considering these conditions, we could only fit at most three sensors in one packet.

There are two kinds of indicators, one for the packet itself and the other kind is for each sensor. The packet indicator is a number. It is from 1 to 3 and it is written in the first byte of each packet. For the sensors, the indicator is ‘/*’ and the * is replaced with the first letter of each sensor. For instance, CO2 sensor’s indicator is ‘/c’ and soil moisture sensor’s indicator is ‘/s’ as shown in table 3.

**Table 3. Sensors and their identifiers.**

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>Sensor Indicator</th>
<th>Corresponding Packet Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO2</td>
<td>/c</td>
<td>1</td>
</tr>
<tr>
<td>PH</td>
<td>/p</td>
<td>1</td>
</tr>
<tr>
<td>Soil Moisture</td>
<td>/s</td>
<td>2</td>
</tr>
<tr>
<td>Temperature</td>
<td>/t</td>
<td>2</td>
</tr>
<tr>
<td>Humidity</td>
<td>/h</td>
<td>3</td>
</tr>
<tr>
<td>Ambient Light</td>
<td>/a</td>
<td>3</td>
</tr>
</tbody>
</table>

PsoC is already programmed to only pass the correct data that it gets from Pyboard. In this project, the correct data means the data packet containing correct indicator for the packet and the sensor data that matches the packet number. The following table shows the indicator for each sensor and where each of those sensors belong.

It is not possible to put the sensor data into a packet that does not correspond to its identifier packet. To make it clear, only data from soil moisture sensor and temperature
sensor can be sent through packet number 2. The order of those two sensors does not matter, however, we need to follow the exact rule for writing the data.

Finally, there is an indicator that shows the end of the packet. After writing all the data in the Pyboard code, we need to add an indicator which is ‘+’ at the very last byte of packet. This indicator will be used in PSOC to figure out the end of each packet.

The order of the data in one packet is shown in figure 15 below.

![Data packet format](image)

Figure 15. Data packet format.

In figure 15, blue part is where the packet number goes, orange part is for the sensor indicator and the yellow part is for the value that we get from the corresponding sensor for the indicator that we need to write on the left orange box and green part is for the end indicator. As mentioned above, longest packet length is 20 bytes. As we are using 6 bytes already for the indicators, we need to make sure to round off the sensor value to 2. Below is an example of packet number 1.

3.3.5 Data Reception and Conversion

We will receive the packets on the devices that connect to with PsoC BLE. The data type that PsoC handles is uint8 and those data will appear on devices in form of hexadecimal number. The letters that PsoC receives from Pyboard are handled in the way that each letter are received in form of uint8_t, which is same with handling char, then cast the letter in form of uint8. From PsoC to device, the letter will be sent in form of uint8 and will be received in form of hexadecimal therefore, we need to convert the value using ASCII table as shown in figure 16.
One thing that we should not be confused is the value that we receive is originally char data therefore, from the ASCII table we need to reference the ‘Char/Chr’ for the ‘Hx(hexadecimal)’ value that we receive to have the final result. For example, for the packet indicator 1, 2, 3, the result that we get from the connected device would be 31, 32, 33.

### 3.3.6 Raspberry Pi BLE Connection

Raspberry Pi was used to pair with the BLE module and collect the data from the sensor. Further the data could be used to convert to the human readable format and manipulate the data to be used in the database or any other applications.

The Raspberry Pi would require a Bluetooth dongle version 4.0 or higher for the BLE protocol to perform normally. The Raspberry Pi required raspbian operating system and bluez packet and its dependencies installed.
After all the necessary packets are installed we can begin Bluetooth communication and gatttool to receive the data as follows.

- **Lsusb**: Checks the buses or devices connected in the serial port of the Raspberry Pi.
- **hciconfig hci0 up**: Makes the serial port on.
- **hcitool dev**: Ensures and prints the address of the Bluetooth device connected on the port.
- **hcitool lescan**: Bluetooth low energy scanning

After the lescan the available BLE devices with their respective address will be displayed: We need to note our specific Bluetooth device address as we will be able to identify it depending on the name of our BLE device.

```
xx:xx:xx:xx:xx:xx       Board1
```

Where xx represents the Bluetooth address and Board1 represents name of our Bluetooth device.

Now the gatttool command can be used to connect and retrieve the data from the BLE device and its respective channel. The gatttool commands used are listed below:

- **sudo gatttool -b xx:xx:xx:xx:xx --interactive** (where xx represents Bluetooth address)
- **connect** (Connect/pair to the BLE device)
- **char-write-req** 0x000f 0100 (Enable the push notification in the 0x000f channel. The following displayed output is the data)
- **Disconnect**

The received data is in hexadecimal which has to convert to integer in order to get correct reading of the sensor on the Pyboard. (Saunby, 2013)

### 3.4 Measurement Board

After designing and building all the circuit and programming the PsoC board, the final board was designed. As discussed earlier, the board contained 3 RJ12 ports, one 10 pin header, ground loop, DC Relays, Ambient light sensor, temperature sensor and step up circuit. The task was carried out in printed circuit board (PCB) designing application KiCad.

#### 3.4.1 Schematic Design
The symbol for the Pyboard, PSoC module, ambient light were built from the library editor in Ki-cad. Further the board was circuit was designed through the schematic design on Ki-cad. After the completion of the schematic the modules were designed using the module editor and assigned to respective component on the netlist file through the PCBnew on the Ki-cad.

Figure 107. Schematic for measurement board.
In figure 17, Pin 1 and Pin 2 in all the RJ 12 ports are connected with Ground and 3.3V respectively. In RJ12 port J1, pin 3 is connected to the X3 pin, pin 4 to Y9, pin 5 to pin Y10 and pin 6 to pin X8. Similarly, port J2 has pin 3, 4, 5 and 6 connected to X4, X7, X21 and X22 respectively. However J3 has pin 6 connected to 5 volts and has three programmable pins Y12, Y11 and X5 from Pyboard connected to pins 5, 4 and 3 respectively.

In the other hand the temperature sensor is connected to X1 pin on the Pyboard. The ambient light sensor has its smbus pins connected to I2C 1 pins on the Pyboard. Similarly, DC relay U4 is connected to pin X12 and relay U6 is connected to pin X11 on Pyboard. The power supply pin is connected to the 27th pin in the Pyboard which is the Vin pin.

3.4.2 PCB Layout

After designing the schematic, netlist file was created and each component were assigned with the modules. The netlist was read and the final board was designed.

![Figure 118. PCB Board for measurement board.](image)

Figure 18 explains the final board designed using the Ki-cad. It has the footprint for all the components. The traces are drawn according to the schematic. The traces also consists of vias thus the board is a two layered PCB board. Both the layers are filled with ground so that no traces were required to be made for the ground pins. The 3.3V to the sensors are connected through the regulated 3.3V of the Pyboard.
Once the design or the PCB layout was ready, the gerber files were created. The gerber files for the front layer, back layer, board outline and mask for both the sides were created using Ki-cad. In addition to this the drill file was also created and thus all the files were sent for production through Cridix Oy.

4 Result

After all the process and design, the board was milled and tested. The board was further sent for production. Once the board was received, all the required components were soldered on the board as shown in figure 15.

In figure 19, the sensor board consists of three RJ12 ports on the left side and a step up power circuit. It also consists of the BLE module, Pyboard, relays, temperature sensor and ambient light sensor.
5 Drawbacks and Solutions

The benefit of a drawback in a project is the learning curve that helps to teach the troubleshooting process. During the project there were a few errors and problems. The problems were troubleshooting and solved with the help of supervisors, teachers and team work.

5.1 PH Probe Error and Solution

The PH electrode provided had no datasheet. Hence we had to design and implement the way of using it. The BNC connector also did not have the ground pin and it took us some time to get a new one. The only way to test the PH electrode was to test the output voltage of the electrode in different kinds of chemicals. However it was a big challenge to get all the typical PH value chemicals from PH value 1 to 14. Hence 6 available chemicals were selected and tested numerous times. With the help of mathematics and test results we were able to finally design our own equation to use and get the result from the sensor.

5.2 PsoC BLE Data Error and Solution

In the other hand the PsoC BLE contained a large number of APIs which were to be studied and because of time management we had to go through them quickly. In addition to this the PsoC was able to transmit through the BLE and we could get the data on the CYsmart application provided by Cypress. However there was a trouble connecting it with the Raspberry Pi. The Raspberry Pi could not receive the data packet of more than 20 bytes. As the specification of the project was to send and get the data on the Raspberry Pi, we had to create the data packets of 20 bytes. Further the data was lost once in a while which made it difficult to send the uncorrupted data. Hence a lot of filtering was required to solve the problem. As explained earlier in chapter 3, each packet of data was assigned with fixed identifier and each sensor was provided with an identifier in order to make the filtering efficient and reliable.

5.3 Humidity Sensor Error and Solution

The first phase of the board was designed and a humidity sensor was integrated in the board. As the sensor had resistive properties, a voltage divider was used followed by a
voltage follower through LM358 in order to make the sensor reading reliable and calculate the resistance of the sensor. One of the sensors was experimented and finalized. However after the production we came to know that the sensor was not responding as it should have with the humid environment.

After checking the capacitance and resistive properties of the sensor, the sensor was provided with 2.8V peak to peak sine wave and the voltage across the sensor was measured. However, the sensor again did not respond and the reading each time varied from the other.

Taking into account the capacitive and resistive properties of the sensor, we calculated the impedance of the sensor to be high and hence we again decided to use the sensor in a voltage divider with varying resistors supplying with 3.3V DC voltage. However the voltage output of the sensor was very low and the sensor didn’t react to the humidity in the environment.

Finally we decided to supply the sensor with $1V_{\text{rms}}$ at 1 KHz as suggested by the datasheet. We had to remove the DC level of the signal out from the microcontroller as it affected the reading of the sensor. As 100nF sensor converted the signal without the DC component, the signal was supplied to the voltage divider, where the voltage output of the sensor was effective and ranged between 1.2V peak to 54 mV peak. However a new challenge was that the microcontroller ADC would find it difficult to read the negative voltage and thus, we had to convert the AC signal into DC.

A half wave rectifier with the help of a diode and resistor was created and connected in series at the output of the humidity sensor. However this dropped the voltage and the input and the output was different. Hence the half wave rectifier was designed with the LM358 operational amplifier. The output and input matched. However the op-amp had a high input biased current. Therefore we decided to create a negative voltage for the negative input of the op-amp. But when connected with the op-amp, the input gave us false reading.

The TCL2262 op-amp with the low input bias current of 1pA was introduced to overcome the false reading. As a result, this shifted the input voltage when the op-amp was connected to the output of the sensor. On studying the circuit, we found out that the negative bias current passed through the sensor because of the resistive property of the sensor.
This effect shifted the voltage output through the sensor and would kill the op-amp. A 10KΩ resistor in parallel with the humidity sensor was placed in the circuit, which helped to drop the negative bias current through it. We also figured out that in order to make a sensor working, we needed to design the circuit based on the datasheet provided by the manufacturer.

5.4 Raspberry Pi data Error and Solution

As the specification of the project, Raspberry Pi was used as the receiving end device of the data sent by the PsoC BLE module. The blues packet for Raspbian Wheezy was installed and the gatttool was used to get the data. The Raspberry Pi was tested numerous time to receive the data. When the process was carried out by some of the users, the BLE got connected but didnot exchange any data. So, the process was retested.

The SD card for the Raspberry Pi was flashed with Raspbian Wheezy operating system and blues packet was installed. The gatttool was performed and the Raspberry Pi was able to receive the sensor data. Then the kernel version was upgraded and the gatttool didnot work. Later on, the problem was solved by downgrading the kernel version back to 3.18+. In addition to this, a serial module connector was included in the platform board that uses the same transmitter and receiver pin as the PsoC. This would help to use the serial module as the BLE protocol and eradicate the problem of upgrading the kernel version on Raspberry Pi.
The goal of the project was to create a measurement board or sensor platform board for IoT training for the new students in the field of smart systems. The goal was achieved successfully. The temperature sensor and the ambient light sensor was integrated in the sensor platform. A user could program the Pyboard and read the sensor data without worrying about the circuit design for the sensor. We can also connect various other analog and digital sensors to the RJ12 ports and the gpios and read the data depending on the pin they are connected to. The collected data can be sent through the UART to the PSOC BLE module and wirelessly using the Bluetooth low energy protocol.

With the increase in the demand of the Internet of Things, it is necessary to give knowledge about the IoT to new students. The sensor platform will provide the student with knowledge about interfacing the sensors and functionality of the IoT. This can also help them to increase their knowledge in the Python programming skills as they will use Python program in the Pyboard and Raspberry Pi. They can study and use different kinds of bus systems, i.e. a serial bus, I2C bus and SPI bus as all the pins are available to be programmed from the Pyboard.

Furthermore the sensor platform board can also be used in a greenhouse to study the environment of the greenhouse and control the environment automatically. The relays can be used to control the door and water pipes handling. In addition to this, users can get new kinds of sensors according to their needs and program them connecting the sensors by building an external breakout board. Thus this sensor platform has multiple uses. It can be used to study a home environment, a meeting hall environment etc.

The sensor platform is being used by new information technology students at Helsinki Metropolia University of Applied Sciences in the course called Devices. Thus this board has high potential and can be developed further. Before the second phase production of the board a connector for the Bluetooth Low Energy serial module will be integrated in the board. This will help to connect the board with Raspberry Pi through BLE in the future.
References


C-code of PsoC BLE

/**************************************************************************
* Included headers
***************************************************************************/
#include <project.h>
#include <stdio.h>

/**************************************************************************
* Macros
***************************************************************************/
#define FALSE     0u
#define TRUE      1u
#define LED_OFF   1u
#define LED_ON    0u
#define WDT_COUNTER (CY_SYS_WDT_COUNTER1)
#define WDT_COUNTER_MASK (CY_SYS_WDT_COUNTER1_MASK)
#define WDT_INTERRUPT_SOURCE (CY_SYS_WDT_COUNTER1_INT)
#define WDT_COUNTER_ENABLE (1u)
#define WDT_1SEC (32767u)

/*Global Variables*/
uint8 authReq;
uint8 notificationsEnabled=FALSE;
uint16 temperature;
volatile uint32 WDTInterrupt;
uint8 temp1[20];
uint8 temp2[20];

/**************************************************************************
* Function Declaration
***************************************************************************/
void StackEventhandler(uint32 event,void * eventParam);

void StackEventhandler(uint32 event, void * eventParam)
{
    CYBLE_GATTS_WRITE_REQ_PARAM_T *wrReqParam;

    switch(event)
    {
    case CYBLE_EVT_STACK_ON:
        printf("\r\n========BLE Stack ON=========");
        printf("Greenhouse_20 Project\r\n");
        /*Start Advertising*/
        CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
        printf("\r\nStarted to Advertise\r\n");
        break;
case CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP:
    if(CyBle_GetState()==CYBLE_STATE_DISCONNECTED)
        {/*Start advertising if time out happens*/
            CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
        }
    if(CyBle_GetState()==CYBLE_STATE_ADVERTISING)
        {
            Advertising_LED_Write(LED_ON);
        }
    break;

case CYBLE_EVT_GAP_DEVICE_CONNECTED:
    printf("Connected to Client\r\n");
    Connection_LED_Write(LED_ON); /*Turn ON RED LED*/
    Advertising_LED_Write(LED_OFF);
    break;

case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
    printf("Disconnected\r\n");
    /*Turn OFF GREEN LEDs*/
    Connection_LED_Write(LED_OFF);
    notificationsEnabled=FALSE;
    /*Start advertising again*/
    CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
    printf("Started to advertise\r\n");
    break;

case CYBLE_EVT_GATT_WRITE_REQ:
    wrReqParam=(CYBLE_GATT_WRITE_REQ_PARAM_T*) eventParam;
    /*If notification is enabled*/
    if(wrReqParam->handleValPair.attrHandle==
        CYBLE_CUSTOM_SERVICE_CUSTOM_CHARACTERISTIC_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_HANDLE)
    { 
        if(*wrReqParam->handleValPair.value.val==0x01)
            { 
                notificationsEnabled=TRUE;
                printf("Notification enabled\r\n");
            }
        else if(*wrReqParam->handleValPair.value.val==0x00)
            { 
                notificationsEnabled=FALSE;
                printf("Notifications disabled\r\n");
            }
    }
    CyBle_GattsWriteRsp(cyBle_connHandle);
    break;

default:
    break;
Appendix 1

3 (6)

CY_ISR(Timer_Interrupt)
{
    if(CySysWdtGetInterruptSource() & WDT_INTERRUPT_SOURCE)
    {
        /* Indicate that timer is raised to the main loop */
        WDTInterrupt=TRUE;
        /* Clears interrupt request */
        CySysWdtClearInterrupt(WDT_INTERRUPT_SOURCE);
    }
}

void WDT_Start(void)
{
    /* Unlock the WDT registers for modification */
    CySysWdtUnlock();
    /* Setup ISR */
    WDT_Interrupt_StartEx(&Timer_Interrupt);
    /* Write the mode to generate interrupt on match */
    CySysWdtWriteMode(WDT_COUNTER, CY_SYS_WDT_MODE_INT);
    /* Configure the WDT counter clear on a match setting */
    CySysWdtWriteClearOnMatch(WDT_COUNTER, WDT_COUNTER_ENABLE);
    /* Configure the WDT counter match comparison value */
    CySysWdtWriteMatch(WDT_COUNTER, WDT_1SEC);
    /* Reset WDT counter */
    CySysWdtResetCounters(WDT_COUNTER);
    /* Enable the specified WDT counter */
    CySysWdtEnable(WDT_COUNTER_MASK);
    /* Lock out configuration changes to the Watchdog timer registers */
    CySysWdtLock();
}
/***********************
* Function Name: main
***********************
**********
* Summary:
* Main function.
* Parameters:
* None
* Return:
* None
* Theory:
* This function initializes the BLE component and then processes the BLE events routinely
***********************
**********

```c
int main()
{
    uint8_t letter_py;
    uint8_t indicator;

    int indicator_check[7];
    int counter = 1;
    int send = 0;
    int i = 0;

    CYBLE_GATTS_HANDLE_VALUE_NTF_T serverTemp;

    Advertising_LED_Write(LED_OFF);
    Connection_LED_Write(LED_OFF);

    CyGlobalIntEnable; // Uncomment this line to enable global interrupts.
    UART_Start();
    WDT_Start();
    UART_Pyboard_Start(); //connection with Pyboard UART
    CyBle_Start(StackEventhandler);

    for(;;)
    {
        CyBle_ProcessEvents();

        /* initialize */

        for(i = 0; i < 20; i++)
        {  
            temp1[i] = (uint8)NULL;
        }

        for(i = 0; i < 20; i++)
        {  
            temp2[i] = (uint8)NULL;
        }
```

for(i = 0; i < 7; i++)
    indicator_check[i] = 0;

send = 0;

/* start getting data from Pyboard */

indicator = UART_Pyboard_UartGetChar();    //temp[0]

if(indicator == '1'){
    UART_UartPutChar('A');

    for(i = 1; i < 20; i++) {
        letter_py = UART_Pyboard_UartGetChar();
        UART_UartPutChar(letter_py);

        //end of data
        if(letter_py == '+'){ // && indicator_check[0] == 1 && indicator_check[1] == 1)
            send = 1;
            break;
        }

        temp1[i-1] = (uint8)letter_py;
    }
}

else if(indicator == '2'){
    UART_UartPutChar('B');

    for(i = 1; i < 20; i++) {
        letter_py = UART_Pyboard_UartGetChar();
        UART_UartPutChar(letter_py);

        //end of data
            send = 1;
            break;
        }

        temp2[i-1] = (uint8)letter_py;
    }
}

UART_Pyboard_SpiUartClearRxBuffer();
CyDelay(1000);

    /*WDT interrupt triggered*/
    if(WDTInterrupt==TRUE && CyBle_GetState()==CYBLE_STATE_CONNECTED && notificationsEnabled==TRUE && send == 1)
    {
        if(counter%2 == 1)
            //Send notification
            serverTemp.attrHandle=CYBLE_CUSTOM_SERVICE_CUSTOM_CHARACTERISTIC_CHAR_HANDLE;
            serverTemp.value.val=temp1;
            serverTemp.value.len=sizeof(temp1);
if (CYBLE_ERROR_OK==CyBle_GattsNotification(cyBle_connHandle,&serverTemp))
{
    WDTInterrupt=FALSE;
    counter = counter +1;
}

}else{

    //Send notification
    serverTemp.attrHandle=CYBLE_CUSTOM_SERVICE_CUSTOM_CHARACTERISTIC_CHAR_HANDLE;
    serverTemp.value.val=temp2;
    serverTemp.value.len=sizeof(temp2);

    if (CYBLE_ERROR_OK==CyBle_GattsNotification(cyBle_connHandle,&serverTemp))
    {
        WDTInterrupt=FALSE;
        counter = counter +1;
    }

    send = 0;
}

/* [] END OF FILE */