Expertise
and insight
for the future

Romet Kalpus

Implementation of IoT Sensor Platform

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and communication technology (ICT)

Bachelor's Thesis

28 October 2019

Author Title Number of Pages Date	Romet Kalpus Implementation of IoT Sensor Platform 39 pages 28 October 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Smart Systems
Instructors	Keijo Länsikunnas, Senior Lecturer

This thesis documents an implementation of an IoT sensor platform from a point of view of a single software developer. The documentation goes through designing the hardware of the platform, developing the embedded software and developing a web application for end users to get access to the data the platform sends to the cloud.

The hardware designing part of the documentation goes through the evaluation of electronic components embedded to the project, the designing of electronic schematics backed up with relevant circuit theory and the designing layout of the printed circuit board. The process of ordering PCBs is documented with the total price of each ordered PCB.

The embedded software developing part documents embedded software engineering using modern tools in developing IoT devices. The concept of an IoT sensor platform is prototyped using a temperature / humidity sensor, sending sensor data to the cloud using the modern application level networking protocol used widely to send data to network from energy constricted embedded devices.

The web developing part documents the software implementation responsible for fetching sensor data from the cloud and the simple user interface implementation to display the sensor data for the end user to see in a human readable way. The web developing part of the project also aimed at using modern web software developing tools.

The documentation aims to give the reader some prospect of the extent of developing an IoT device.

Keywords	IoT, PCB design, embedded software

Tekijä Otsikko Sivumäärä Aika	Romet Kalpus Implementation of an IoT sensor platform 39 sivua 28.10.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Älykkäät järjestelmät
Ohjaajat	lehtori Keijo Länsikunnas

Kyseinen insinöörityön dokumentaatio dokumentoi IoT-anturointialustan suunnittelua yksittäisen ohjelmistosuunnittelijan näkökulmasta. Dokumentaatiossa käydään läpi alustan elektroniikka-suunnittelu, sulautetun ohjelmiston kehittäminen sekä web-sovelluksen kehittäminen, mistä käyttäjä pääsee käsiksi alustan pilveen lähettämään dataan.

Työn elektroniikkasuunniteluvaiheessa määritellään työhön sulautettavat komponentit, jotka ovat tarpeellisia IoT-alustan toiminnallisuuteen. Dokumentoidaan työn piirikaavion suunnittelu, missä käydään läpi tähän projektiin oleellinen piiriteoria, piirilevyn suunnittelu sekä suunniteltuun piirilevyyn ja siihen sulautettujen komponenttien kokonaiskustannukset.

IoT-alustan sulautetusta ohjelmistosta dokumentoidaan moderneja työkaluja ja niiden käyttöä kyseisen IoT-alustan ohjelmiston kehittämiseen. IoT-alustan konseptia on esitelty käyttäen lämpötila- tai kosteus anturia. Tietoa lähetetään pilvipalveluun käyttäen sovellustason verkko-protokollaa, joka on nykypäivänä laajasti käytössä resurssien puolesta rajoitetuissa laitteissa lähettämään dataa verkkoon.

Web-sovelluksen kehitysosuus antaa käyttäjälle käyttöliittymän, jonka kautta hän pääsee käsiksi IoT-alustan lähettämään dataan. Web-sovellukseen liittyy ohjelmisto, joka hakee pilvipalvelusta IoT-anturin lähettämän datan, sekä ohjelmisto, joka esittelee datan käyttäjälle.

Dokumentaatio pyrkii antamaan lukijalle kuvan IoT-laitteen suunnittelun eri osa-alueista ja sen laajuudesta.

Avainsanat	IoT, PCB suunnittelu, sulautettu ohjelmisto

Contents

List of Abbreviations

EMC	Electromagnetic compatibility. Ways of making a device electromagnetically compatible in an environment (I.e with other devices functioning in the environment) it is supposed to function.
RF	Radio frequency. A large portion of electromagnetic spectrum ranging from 30 Hz to 300 GHz. In this thesis it is referred to radio frequency in the context of WiFi and Bluetooth Low Energy. Both use radio frequencies ranging from 2.4 GHz to 2.5 GHz for communication.
BLE	Bluetooth Low Energy. Bluetooth communication specification that has smaller current consumption than original Bluetooth. Used widely, especially in low-power devices.
SPI	Serial peripheral interface. Widely used serial communication protocol.
I2C	Inter-integrated circuit. Widely used serial communication protocol.
UART	Universal asynchronous receiver-transmitter. Widely used serial communication protocol.
MCU	Microcontroller. A small computer on a single integrated circuit consisting of a processor, memories and other peripherals.
Balun	Balanced/Unbalanced. Used in antenna matching circuits when converting balanced (differential) signal to unbalanced (signal of ground reference) and other way around.
QFN	Quad Flat No-Leads. Type of square packaging of chips which does not have leads going outward. They can be made tiny and are temperature resistant.
QFP	Quad Flat Package. Type of square packaging of chips which has leads going outwards.

MOSFET	metal-oxide-semiconductor field-effect transistor. Type of field-effect transistor.
API	Application programming interface. A program accessible for a programmer by a function which has a purpose of abstracting away underlying details of some functionality.
IDE	Integrated development environment. Software developing environment consisting of various tools that help developing software.
REST	Representational State Transfer. An application level protocol used to exchange information between browser and server.
JSON	JavaScript object notation. In this project, it is type of format to encapsulate data in when sending data over network using RESTful web services. Other than using this format as REST style API, it is also widely used to represent configuration files. It is a convenient format to parse.
MQTT	Message Queuing Telemetry Transport. A lightweight application level protocol used to exchange information between client device and server.
TLS	Transport Layer Security. An application level protocol used to send encrypted data between devices.
HTML	Hypertext Markup Language. A document used by browser to render a web page.
BSP	Board Support Package. Software that is responsible for interfacing hardware and software of a specific development board / module / chip.
SPA	Single Page Application. Technology used to represent web applications in a way that client device downloads the whole software of web application from server. HTML is also generated on client device rather than on server.

1 Introduction

Devices connected to a network is nothing new but is a constantly growing trend as technology evolves. Printed circuit boards are relatively cheap nowadays allowing even individuals to easily order one, start prototyping and with the right tools, connect the device to the Internet. Miniature devices that measure environment and send data to the Internet which can be accessed by the end user to see via browser on a mobile device is a growing business trend. The terms “IoT” or “Internet of Things” and “smart” can be seen constantly as part of a marketing strategy for businesses. These terms aim to message people that the company uses modern technologies and co-operating with the company is profitable for everyone. These terms are used constantly in marketing but the actual implementation of these kinds of devices is completely unfamiliar to many, even to many of those who constantly use the terminology to market their businesses.

This thesis explores the implementation of an IoT platform and many of its design phases as implemented by a single software developer. The documentation aims to introduce the reader to the extensiveness of implementing an IoT device while practically going through the implementation of an actual device. The scope of the project is wide, ranging from designing hardware of the device to implementing user-interface to get access to the data via browser. Each sub-section of implementing the device introduces the reader to some of the modern technologies a developer can use to implement IoT device chosen from large array of optional technologies.

The documentation starts with specifying hardware for an IoT platform. After specifying the components and other hardware of the device, designing the electronic schematics and layout for a printed circuit board of the IoT platform is documented. Once the hardware of the platform is designed and assembled, the thesis takes the reader to developing embedded software.

The embedded software part introduces the reader to some of the modern technologies and tools that can be used to connect an embedded device to the cloud. This section also introduces the reader to some developing environments of Internet connected embedded devices.

The remaining sections go through some technology stacks used to bring the data that an IoT sensor platform sends to the cloud for the end user to see. This includes an implementation of back-end software responsible of fetching data from the cloud and client-side front-end software to display the data for end user.

2 Technologies Used

This section explains the key technologies used in the project. The purpose of this section is to give a basic overview of the technologies to the reader unfamiliar with the subjects to make the reading of this documentation more fluent.

2.2 Serial Communication Protocols

Serial communication protocols enable digital circuits to communicate with each other inside larger digital circuits. For example, an MCU (microcontroller unit), which can act as a larger digital circuit, contains number of different peripherals which need to communicate with each other. Peripherals can signify electronic circuits or devices such as memory modules or sensors.

Single peripherals themselves can also include numerous circuits which communicate with each other using serial communication protocols. Bigger electronic circuits, for example those that are designed to interpret environment electronically called sensors, often use serial communication protocols to communicate with a host microcontroller unit. Sensors have their own microcontroller units which include at least one central processing unit (CPU) and memory modules along with circuits used to interpret (or sense) environment. These numerous electronic circuits inside the sensor can communicate with each other using serial communication protocols and parallel communication protocols. Usually though, serial communication protocols are used between devices that include processing unit and memory to independently process the commands sent via serial communication protocols. It would be ineffective for host device to fully control the peripheral sending detailed commands and receiving all the data through one wire. Parallel communication is used for this purpose. Host microcontroller unit can have multiple sensors connected which often use serial communication protocols to receive the data from sensors which are then used for further processing.

The term “serial” comes from the way electronic signals are sent through the wire: the data is sent through a single wire, one signal at a time, and interpreted as bits by receiving circuits by changes in electronic potential in respect to ground potential or in respect to inverted signal in case of protocols utilizing differential signaling.

Serial communication protocols used in this project are UART, I2C and SPI. These are widely used in the field of electronics for inter-circuit communication. Saying that UART is a protocol is a bit ambiguous as it is actually a device that can be configured to send data in numerous ways but here it is abstracted away as it is used in a single way.

While UART (universal asynchronous receiver-transmitter) hardware has historically been included in transceivers of input devices of personal computers such as keyboard and mouse, nowadays it is mainly used to connect sensors and other peripherals in embedded systems. UART is one of the easiest serial communication technologies to use. It can only have two end-points as the protocol does not use addressing to differentiate between slave devices, the devices using this protocol have dedicated roles as receiving devices or transmitting devices as the hardware has dedicated circuits for receiving and transmitting.

Unlike UART, SPI (serial peripheral interface) and I2C (Inter-Integrated Circuit) make it possible to connect multiple devices to one communication hardware as I2C uses addressing to select chips (i.e. devices) and SPI uses dedicated pins called “chip select pins” for each connected device to select target device. I2C is usually configurable to data transmission speeds of 100 kb/s or 400 kb/s [1 pp. 8] while SPI can be configured to rates up to Megabits per second [2 pp. 917]. Using I2C most certainly saves “real estate” on PCB as it only requires two wires; one for data transmission and one for clock. Addressing chips is also done through data wire so no additional pins for slave devices is required. SPI needs at least four wires; for data transmission from master to slave, for data transmission from slave to master, for clock and for chip select. Additional wire is needed for every slave device that communicates with master device over SPI. This additional wire is needed to send a signal, that is usually pulling a chip select wire to the ground, notifying the slave device that the master device is about to send data.

Figure 1 demonstrates the setups of these popular serial protocols.

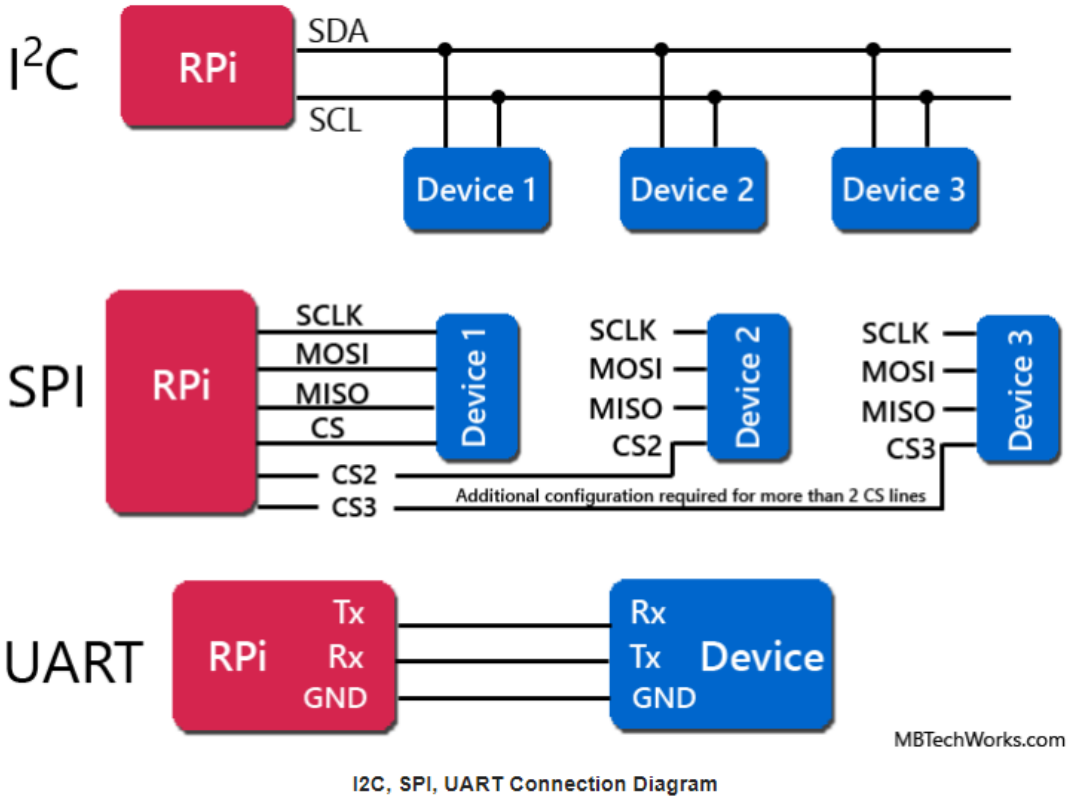

Figure 1 Setup of I2C, SPI and UART devices. [3].

Figure 1 shows the setups of I2C, SPI and UART devices. The illustration uses Rasperry Pi as a host device which is not used in this project, but it illustrates the typical setups regardless of the device used as a host device.

2.3 Radio Frequency Communication Technologies

Wi-Fi and BLE (Bluetooth Low Energy) technologies are used for radio frequency communication in this project. Both technologies use frequency channels in the area around 2.4 GHz. Figure 2 illustrates some of the modern radio frequency communication technologies operating at the range of 2.4 GHz.

There are modules available with a variable amount of software embedded in them. Most of them include software stacks included for basic communication that is specific to the protocol. Some Wi-Fi modules also include software stack for TCP/IP communication. This project attempts to use a Wi-Fi module without TCP/IP stack included.

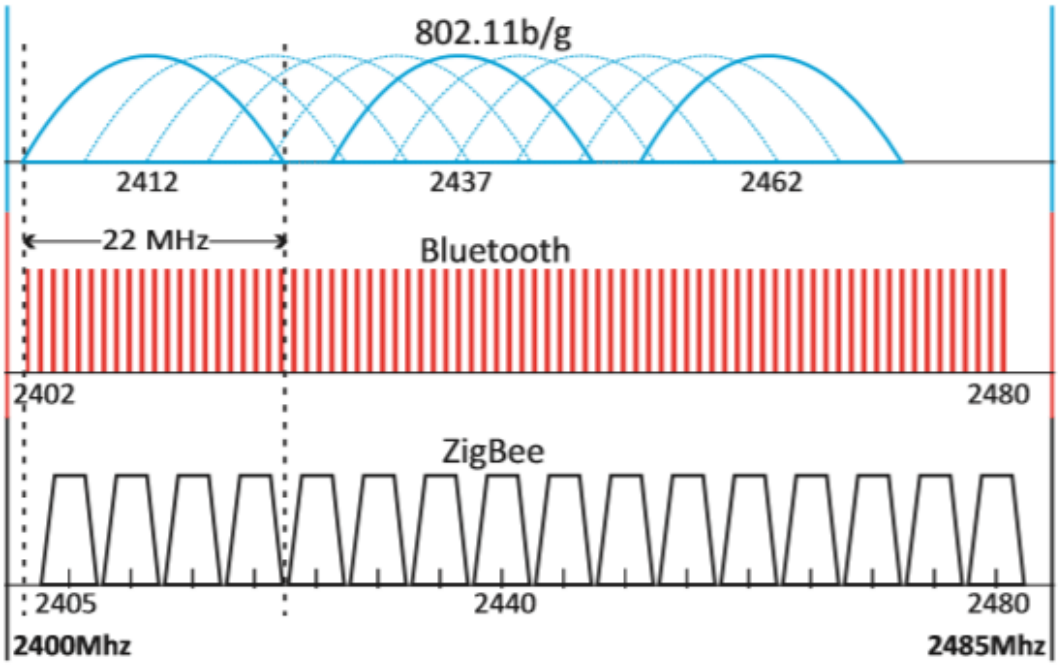

Figure 2 Radio frequency communication technologies operating at the range of 2.4 GHz. [4]

Figure 2 is taken from a white paper of a study which investigates interference of such technologies and by which a significant performance degradation can occur in small office and home environments [4].

A design of an antenna circuit is attempted for Wi-Fi and BLE modules. An important concept when designing an antenna circuit is a “balun” which stands for balanced/unbalanced. The baluns function is to convert balanced signal to unbalanced and other way around. By unbalanced signal, it is meant a signal that uses Ground as a reference. So, when an electric potential between Ground and a signal is 5 Volts, the signal is a 5 Volt signal. A signal can also be pulled to the Ground, meaning signal is a 0 Volt signal because there is no significant electrical potential between different points in Ground plane. By balanced signal it is meant a differential signal. A differential signal has two signal outputs; one for an original “positive” signal and one for an inverted “negative signal”. Rather than referencing signal to the Ground, the positive and negative signals are referenced to each other [5]. One way to interpret differential signals as bits in software would be interpreting a signal as 1 (or logical high) when “positive” signal has a positive amplitude (while “negative” signal has a negative amplitude), and as 0 (or logical low) when “negative” signal has a positive amplitude (while “positive” signal has a negative

amplitude) [5]. One benefit of differential signaling is more rapid sampling of logic levels. As the logical levels (logical 0 and 1) are not bound to a specific voltage level as in single-ended signaling (ground referenced signal) but to positions of negative and positive amplitude relative to each other. Another benefit is a higher signal-to-noise ratio. As noise will couple to both positive and negative signals (meaning the electrical peak of noise is introduced to both signals) and the logical level comes from referencing positive and negative signals to each other, differential signals are more resistant to noise than single-ended signals [6].

Figure 3 illustrates a digital single-ended signal (ground referenced signal) and digital differential signal.

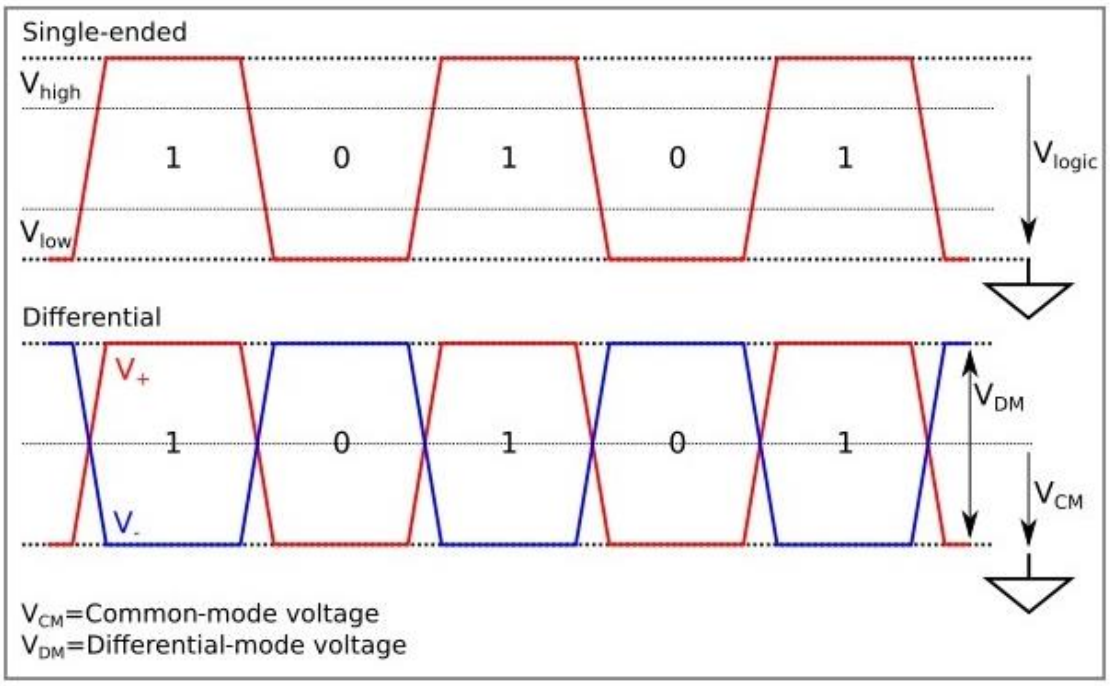

Figure 3 Illustration of digital single-ended (Ground referenced) and digital differential signal.

One way to interpret logical levels of differential signals in software would be at the crossing points of V+ and V-. This is potentially faster than interpreting logical levels of single-ended signals as single-ended signals need to “travel” above or beneath some threshold voltage, for example 1.7V, which is the case with STM32F4 devices [7 pp. 91].

2.4 Networking Protocols

Out of the application level protocols (think of OSI model) used in the project, MQTT is of most interest as it is used frequently for inter-network communication in the world of low-powered devices. The server receiving data using MQTT protocol has software acting as a *message broker*. Message broker is a piece of software that is responsible for routing data to the *topic* that it was intended. A topic in MQTT communication is essentially a document that represents a device [8]. In this project, a cloud service called AWS IoT Core was used for MQTT communication between the sensor platform and server. The cloud service uses JSON (JavaScript Object Notation) as document format representing a single device. In this environment, the broker software is used to route a message (i.e. data) to the correct topic which represents a device.

3 Designing Hardware of IoT Sensor Platform

This section goes through designing the hardware of the device. It starts from specifying the hardware tailored for the purpose of this project. After the components are specified, the documentation goes through designing the schematics of the circuit board with relevant theory. Documentation on designing electronic schematics is followed by documentation on designing electronic layout of the board, which specifies for the manufacturer how the printed circuit board should look like physically. Lastly, the documentation goes through the assembly and soldering of the printed circuit board.

3.1 Specifying Hardware

Among the first things for an embedded engineer to do is to specify the hardware which to program on. Even as the engineer is working as an embedded software developer, part of the job description can be specifying the best suiting chips or modules with a help of a hardware designer for a specific project. The decision of hardware compatibility in a project can be influenced by environmental requirements (i.e. in which kind of environment the device has to function), current consumption (e.g. how critical it is for a device to function for a long time without charging battery or changing the batteries) among many other factors. In a professional environment the design of a printed circuit board (PCB) is usually left for a hardware designer. The reason for that is because in addition to designing logical schematics and layout for the device, hardware designing also includes making the device electromagnetically compatible in the environment it is meant to function which includes passing certification markings in tests etc. Keeping up with the latest trends that involve both designing electromagnetically compatible hardware and tools for the effective software can be overwhelming for just one person to handle. In the present study also the PCB is designed by the embedded systems programmer without taking the electromagnetic compatibility (EMC) to nearby devices analytically into account.

3.1.1 Choosing Chips

The factors for choosing radio frequency (RF) communication devices (i.e. Wi-Fi and Bluetooth Low Energy (BLE) chips) were the amount of documentation available and

built-in support for known communication protocols. Both chips have support for SPI (serial peripheral interface: a widely used serial communication protocol) which is used to interface them to the host processor. Wi-Fi chip has additionally support for I2C (Inter-Integrated Circuit: a widely used serial communication protocol) and UART (universal asynchronous receiver-transmitter: a widely used serial communication protocol). The function of BLE chip would be to send data to nearby device that is connected to the network. Originally the plan was to design three different PCBs with two of them having BLE chips built in and one of them having Wi-Fi chip built in, that is the device that is connected to the network. Ordering PCBs with three different designs proved to be costly. Though the project was settled on one design, BLE chip remained. The function of a Wi-Fi chip is to connect the device to a nearby Wi-Fi access point connected to the wide area network. Optional ways to access wide area network would have been using LoRa network or mobile network using NB-IoT. Both are widely used technologies for low-powered IoT devices. The device is designed to function at home with Wi-Fi available and is designed with a possibility in mind that additional sensors will be added in the future. The sensors would not necessarily send data with long intervals. These are some of the reasons Wi-Fi was chosen over LoRa or NB-IoT for this project. The Wi-Fi chip used is Atmel's ATWILC1000B-MU-Y and BLE chip used in this project is ST's BLUENRG-MSQTR.

The factors affecting choosing the host microcontroller (MCU) were certainly satisfied efficiency for the first revision of the project, versatility, cost-to-quality ratio and previous familiarity with the microcontroller. For these reasons, ST's STM32F429 was chosen as a host MCU. In a package that costs around 13 euros (DigiKey), there are quite many peripherals available to connect sensors and other slave devices into. The processor can be clocked up to 180 MHz which is probably more than enough. The MCU has 2 MB of Flash memory available for programmers which is enough for relatively large and sophisticated embedded software.

Honeywell's HIH8120 acts as a sensor measuring temperature and humidity. The reason for choosing this is the certainty of accurate data from previous experience, although the price is relatively high. Also, the interface is programmer friendly. The main reason for choosing a sensor measuring temperature and humidity was to ease the proof of concept phase. Temperature and humidity data are easy to debug when testing if the device is sending valid data to the cloud.

Texas Instrument's BQ24075 acts as a charging circuit. The chip does not have data pins to communicate with the device feeding current, only 100 mA or 500 mA input current can be selected.

Low-dropout voltage regulator was needed to regulate the voltage. The reason for this is that even though the battery feeds the device with 3.6V, the regulator must still be able to regulate it down to 3.3V, hence the low-dropout voltage regulator. Non low-dropout voltage regulator would not be able to step down that slight voltage difference. ST's LD39150 was chosen for a regulator.

3.1.2 Choosing Printed Circuit Board

The printed circuit board has four layers. The top and bottom layers can be left for routing, the second layer can be set as ground and the third layer can be left for power floods. The first reason for choosing a four-layer PCB is a separate ground and power layer. In routing point of view, it is easier to get power straight from plane using "vias" rather than routing them separately from regulator to different circuits. Vias are "holes" that electrically connect different planes. There is the same advantage with a separate ground plane, the ground signal of the circuits can be grounded with via straight to ground plane rather than using more complex routing. The second reason is the reduced electromagnetic radiation compared to two-layer boards because of short power and ground signal routings. Short power and ground routings reduce unwanted inductance and voltage peaks in circuits. Voltage peaks arise due to rapid changes in current of AC circuits of high inductance [9]. Additionally, the layout design guides of both RF chips recommended to have a separate ground layer for reduced impedance and noise [10, 11].

3.1.3 Choosing Other Components

The passive components (resistors, inductors and capacitors) were chosen in the size of 0603 (1.6 mm × 0.8 mm). The reason for this was that the mounting of components on the PCB was by hand soldering and 0603 is the personal minimum when soldering by hand for the author. The device has a solar cell which has the function of lengthening the time to need to charge the battery. The solar cell constantly charges the battery with the current it can generate in a particular environment. The battery itself is a Lithium Ion battery in the package of 18650 which is widely used. It has a capacity of 3400 mAh and

has a voltage of 3.6 Volts. Antennas for Wi-Fi and BLE are in the package of 1206 (3.2 mm × 1.6 mm).

3.2 Designing Electronic Schematics

Designing electronic schematics involves making logical connections with components and between chips. This chapter goes through the most challenging parts of designing electronic schematics, namely designing baluns (balanced/unbalanced circuits) and impedance matching networks for Wi-Fi and BLE chips. The schematics were designed using the KiCad's Eeschema software.

Even though BLE chip has a reference schematic available as does Wi-Fi chip, the BLE reference schematics did not have values specified for the components in the balun and impedance matching network. Finding values for the capacitors and inductors of the BLE's balun circuits was an area of research for the author as there has been no previous experience of antenna design or designing radio frequency circuits. Even when using discrete balun calculators there needs to be some intuition about the balun design. The idea of a balun is to convert balanced (I.e. differential) signal to unbalanced (I.e. signal using ground as a reference) and other way around. A balun consists of a low-pass LC filter and a high-pass LC filter as seen in Figure 4.

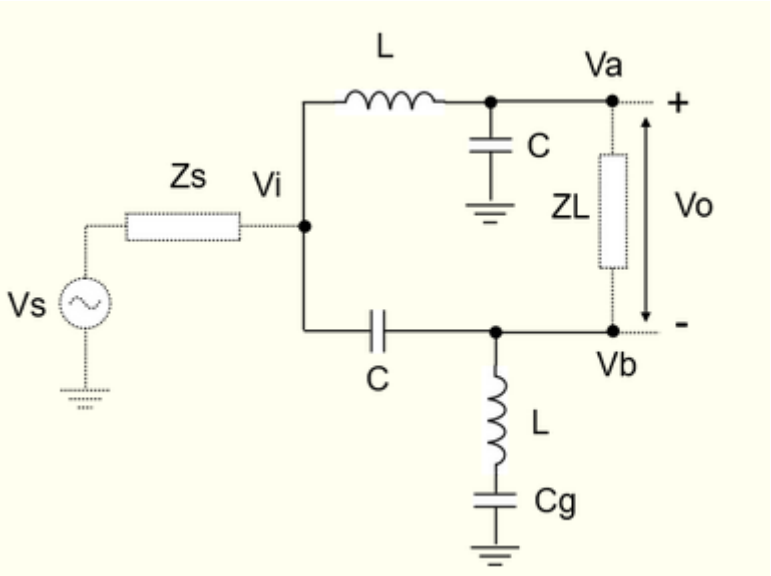

Figure 4: Low-pass LC filter (up) and high-pass LC filter (down). [12]

Wi-Fi and BLE chips send out differential signal from the RF pins, meaning that one of the two pins sends out inverted signal in respect to the other pin. The antenna circuit is a ground referenced circuit. When the antenna picks up a signal it needs to be converted to a differential signal before reaching the chip.

A low-pass LC filter makes voltage lag current by 90 degrees as it is a capacitive circuit and a high-pass LC filter makes voltage lead current by 90 degrees as it is an inductive circuit. The result is that the signal with a +90-degree phase shift goes to the positive RF pin of the chip and the signal with a -90-degree phase shift goes to the negative RF pin. The signal with a positive phase shift now leads the signal with a negative phase shift by 180 degrees. This means that the signal with a negative phase shift is inverted in respect to the signal with a positive phase shift meaning the signal is now differential when reaching the chip. [12,13] This concept of phase shift in inductive circuit is illustrated in Figure 5.

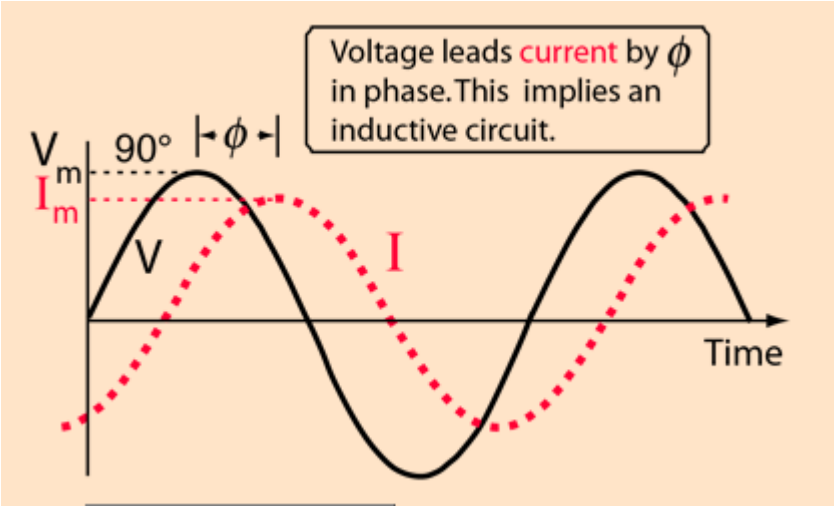

Figure 5 Phase shift in inductive AC circuit. [13]

Referencing to Figure 5 which illustrates the phase shift in the inductive AC circuit, in the capacitive circuit, it is other way around. In the capacitive circuit, the current leads voltage by 90 degrees.

Formulas 1 and 2 are referenced when explaining the concept of phase shifting mathematically.

$$Z_L = j\omega L$$

Formula 1: Impedance for inductor

$$Z_C = -j\frac{1}{\omega C}$$

Formula 2: Impedance for capacitor

Referencing to Formula 1, an impedance for an inductor, and Formula 2, an impedance for a capacitor, then mathematically imaginary j in the formula of inductor's impedance means +90 degrees in complex plane and -j in the formula of capacitor's impedance means -90 degrees in complex plane.

Another function of a balun is to match impedance of antenna circuit to the chip's circuit [14]. The most common impedance of an antenna is 50 Ohms for historical reasons and that is also the case with the antennas used in this project [15]. To find out which kind of impedance matching network has to be built for BLE, the reference schematic in the datasheet of an evaluation board with the same BLE chip as used in this project was consulted [16]. The evaluation board uses ST's integrated circuit balun as opposed to a lumped LC balun designed in this project. The balun evaluation board uses is a 1:1 balun, meaning that the impedance is matched from 50 Ohm unbalanced circuit to 50 Ohm balanced circuit. From this, the conclusion can be drawn that the impedance of RF circuit inside the BLE chip is also about 50 Ohms unless the evaluation board is not accurately designed.

The reason for impedance matching network is that if the difference of impedance between balanced and unbalanced circuits is large, reflections can occur back to the chip. When the signal reflects back, it means there will be some loss of data that is sent out. Another problem with signal reflection is that the signal going out of the chip forms a standing wave with the signal that is reflected back. At certain points the signal is destructive and at other points the signal is constructive. At constructive points, the amplitude of voltage can raise high enough to damage the chip. [17] Figure 6 illustrates an impedance mismatched circuit in a scenario when a signal with amplitude of 1V (incident signal) in a circuit with impedance of 75 Ohms encounters a circuit with impedance of 50 Ohms [18].

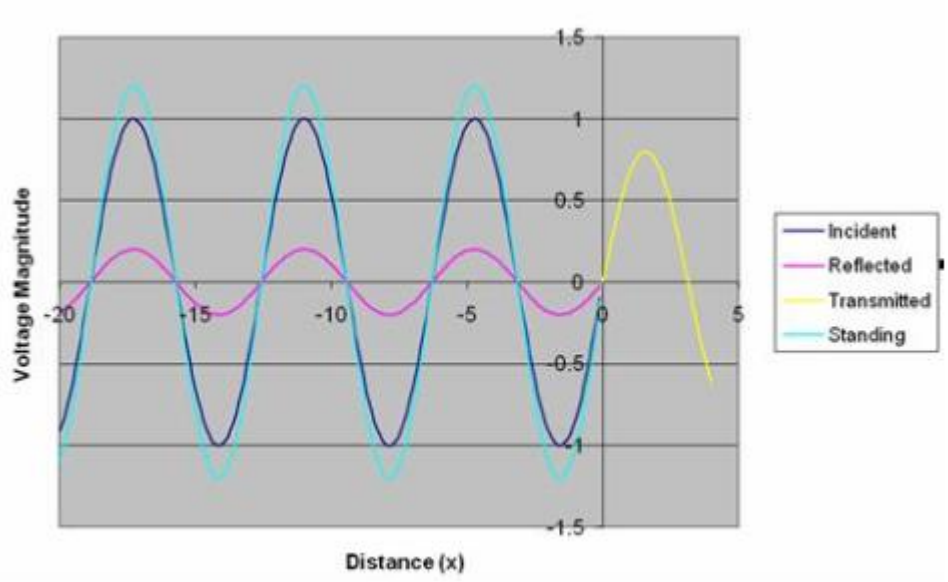

Figure 6 Illustration of reflected signals in an impedance mismatched circuit. [17]

In Figure 6 it can be seen that due to an impedance mismatch in two circuits, part of the signal is reflected back causing a standing wave when incident wave is in phase with a reflected wave. An incident wave constructs with a reflected wave, causing a standing wave with an elevated amplitude. In addition to that, part of the transmitted data can be lost if the mismatch of the two circuits is large.

3.3 Designing Printed Circuit Board Layout

Designing PCB layout involves placing the footpads on the layout and routing the connections of components as they will actually be printed on the board. Footpads or pads are spots of exposed copper where components will be mounted. One of the main things to consider is how to place the footpads so that the pads can be routed together as linearly as possible making the routing less complicated. Once again, the challenging part involved BLE and Wi-Fi chips. As these chips use radio frequencies in the area of 2.4 GHz to 2.5 GHz and the passive components are chosen to be large enough for hand soldering, the traces could not be made optimally short and that introduced the new area of research called transmission lines.

The rule of thumb before transmission line analysis comes into effect is when the length of a trace is at least one fourth of the signal wavelength [19]. In case of Wi-Fi, the trace

between the chip and antenna is about 18 mm. The wavelength of 2.45 GHz in vacuum is about 122 mm. Formula 3 illustrates general formula for wavelength used to calculate length of the trace.

$$\lambda = \frac{v}{f}$$

Formula 3: General formula for wavelength.

Taking into account that the electrical signal does not move through copper wire in the speed of light, the velocity which the signal travels is actually about 0.7 x speed of light. The scaler is not scientifically accurate but is used to scale the velocity of signal inside the copper wire to close enough for the analysis. Now calculating the wavelength of 2.45 GHz inside the copper wire with the scaled velocity, it is about 85 mm. Taking into account that transmission line effects become significant when the wavelength is about one fourth of the wavelength inside the copper wire, the threshold length is now 85 mm divided by 4 which is 21 mm. See Formula 4 for full expression. The result is close to the actual trace length from the chip to the antenna, meaning that it is a good idea to take impedance matching into account. Impedance matching network was designed during electronic schematic designing phase (cf. Chapter 2.2 Designing Electronic Schematic).

$$\lambda = \frac{\frac{0.7v}{f}}{4}$$

Formula 4: Full expression for calculating wavelength when transmission line effects start becoming significant inside copper trace.

The widths of traces have to be also taken into account when designing a controlled impedance radio frequency circuit. The width of trace is inversely proportional to the im-

pedance meaning that wider trace has lower impedance [20]. The factors needed to calculate trace width for controlled characteristic impedance of the trace include electrical dielectric constant of the substrate of PCB, height of the substrate of PCB and thickness of the trace which is 0.035 mm by standard but can be adjusted by consulting the manufacturer [21]. KiCad has some helpful calculators built in and one of them is a calculating tool for calculating transmission line parameters. This tool was used as aid to this complicated subject of calculating trace width to have a trace which has a characteristic impedance of 50 Ohms. Figure 7 illustrates some of the factors which have to be taken into account when calculating the width of the microstrip for controlled characteristic impedance. The length of a trace (or wire) does not affect its characteristic impedance as the same impedance exists in every point of the trace all along the trace [22].

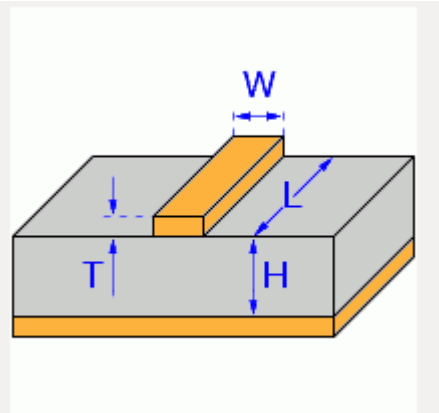

Figure 7 Some of the factors to calculate trace width for controlled impedance circuits. [21]

Figure 7 includes the variables needed to calculate the trace width for controlled impedance circuits. W is for width of the trace, T is for thickness of the trace, H is for height of the substrate and L is for length of the trace. Thickness of the trace, height of the substrate and dielectric constant of the substrate, which is specified by manufacturer, are needed to calculate width of the controlled impedance trace.

3.4 Assembly and Soldering

The boards were ordered from JLCPCB.com. They manufacture boards for prototyping purposes with up to two thousand PCBs per order but minimum of five. The manufactured PCB can have up to six layers. For four and six-layer boards, JLCPCB can also do impedance checking free of charge meaning that JLCPCB will follow the thickness of layers strictly for controlled impedance. This can be useful for RF designs such as the one in question. Among several other options, the customer can choose the color of the board other than green for no additional charge.

The components to be soldered on board were ordered from DigiKey and the batteries were ordered from Akkula.

3.4.1 First Revision

The cost of the PCB was 28.42 euros which was a special offer plus tax from Finnish customs 26.88 euros. The cost of components was 176,77 euros. The cost of two batteries along with a charger was 32,50 euros. The total price of the prototyping board was about 265 euros.

Figure 8 shows the prototyping PCB with components mounted. SC1 (a button switch) is not mounted because DigiKey shipped wrong connectors. The sensor, U4, is not mounted here either. There were a few mistakes on the board design noticed after mounting the components. As it can be seen in Figure 8, the USB connector's (J1) footpad is facing in the wrong direction. Also, the crystal for the oscillator circuit of Wi-Fi (Y3) has wrong pads grounded, meaning the crystal will not work which in turn means that the Wi-Fi will not work. Another major beginner's mistake in the design is that there is no help for debugging when developing the device. There are no LEDs and no additional UART which could be used to send data to the console. This makes it difficult to develop software on the board. The board is not re-cleaned with alcohol before taking the picture which is why it might appear dirty.

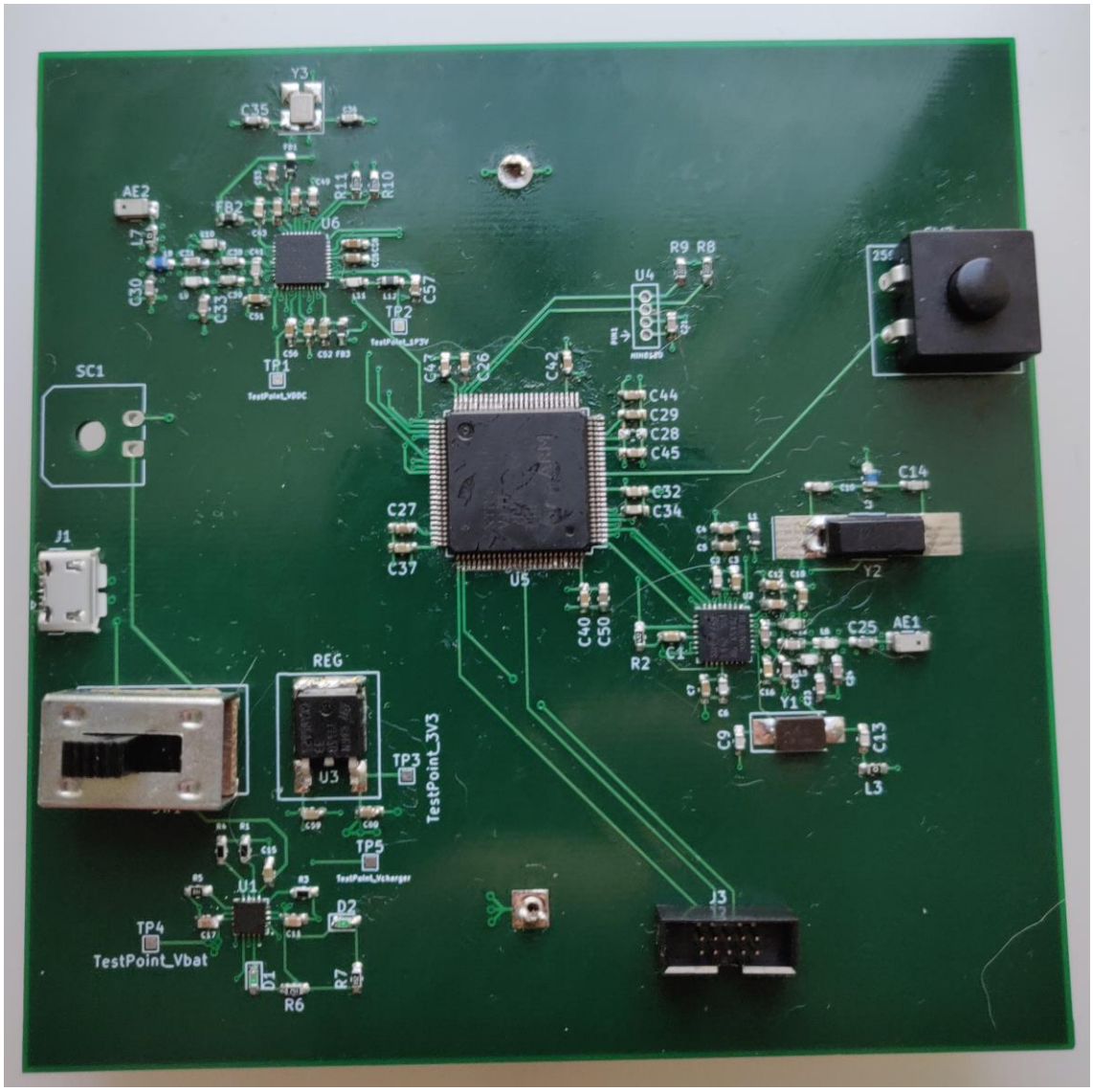

Figure 8 Front side of the first revision of PCB.

The voltages of the board are as they should be. At TP4, which is a voltage directly from battery, it is about 3.7 Volts with fully charged battery referenced to Ground. It is the same at TP5, which is a voltage from the charger chip. The charger chip outputs the

same voltage as battery when battery is fully charged. Voltage from output of the regulator to Ground is 3.3 Volts. Also, the Wi-Fi and BLE chips have correct voltages according to their datasheets [23, 24].

The QFN (Quad-Flat No Lead) packages on the board (i.e. charger, BLE and Wi-Fi chips) were soldered using hot air soldering station. The reason for that were short lead footpads designed on board and soldering the center pad which lays directly under the chip by warming up the chip. The challenges of using hot air soldering are finding the right air pressure which does not blow off the chip of its position and finding the right temperature. The risks of using hot air soldering are damaging the chip due to too high temperature of air, air soldering too close to the chip and air soldering for too long consecutively without giving the chip change to cool. Solder paste was used as a soldering material for QFN packaged chips.

For the QFP (Quad Flat Package) packaged chip (i.e. host MCU), solder paste was used as a soldering material and C shaped flat tip used as a soldering iron tip which is illustrated in Figure 9. C shaped tips are used to drag the tin across the footpads while using soldering flux as a separating material between the footpads.

Figure 9 Shape C soldering tip by Hakko.

Soldering this way, one does not have to solder one pin at a time. Solder paste itself is flux embedded with tiny tin granules making it easy to draw solder paste across footpads with C shaped tip. Negative aspects of solder paste are that it can be messy, hard to clean and one has to get the amount of solder paste just right to prevent hard to fix bridges (electrical shorts generated between pins or pads when soldering). One of the worst kind of bridges are ones that have center pad of QFN package shorted with other pins of the same QFN package. This one is especially hard to fix because it is difficult to

de-solder a QFN package without damaging the footpads of a cheap kind of PCB. Also, there is a risk of damaging the chip when de-soldering it with hot air.

Figure 10 shows the back side of the first revision of PCB.

Figure 10 Back side of the first revision of PCB.

The battery holder seen in Figure 10 holds batteries in the package of 18650.

3.4.2 Second Revision

The cost of a second PCB was 45,28 euros plus tax from Finnish customs 30,92 euros. The cost of components was 63 euros plus tax from Finnish customs about 15 euros. The total cost of the second prototyping PCB was about 155 euros. Figure 11 shows the front side of the second revision of the PCB.

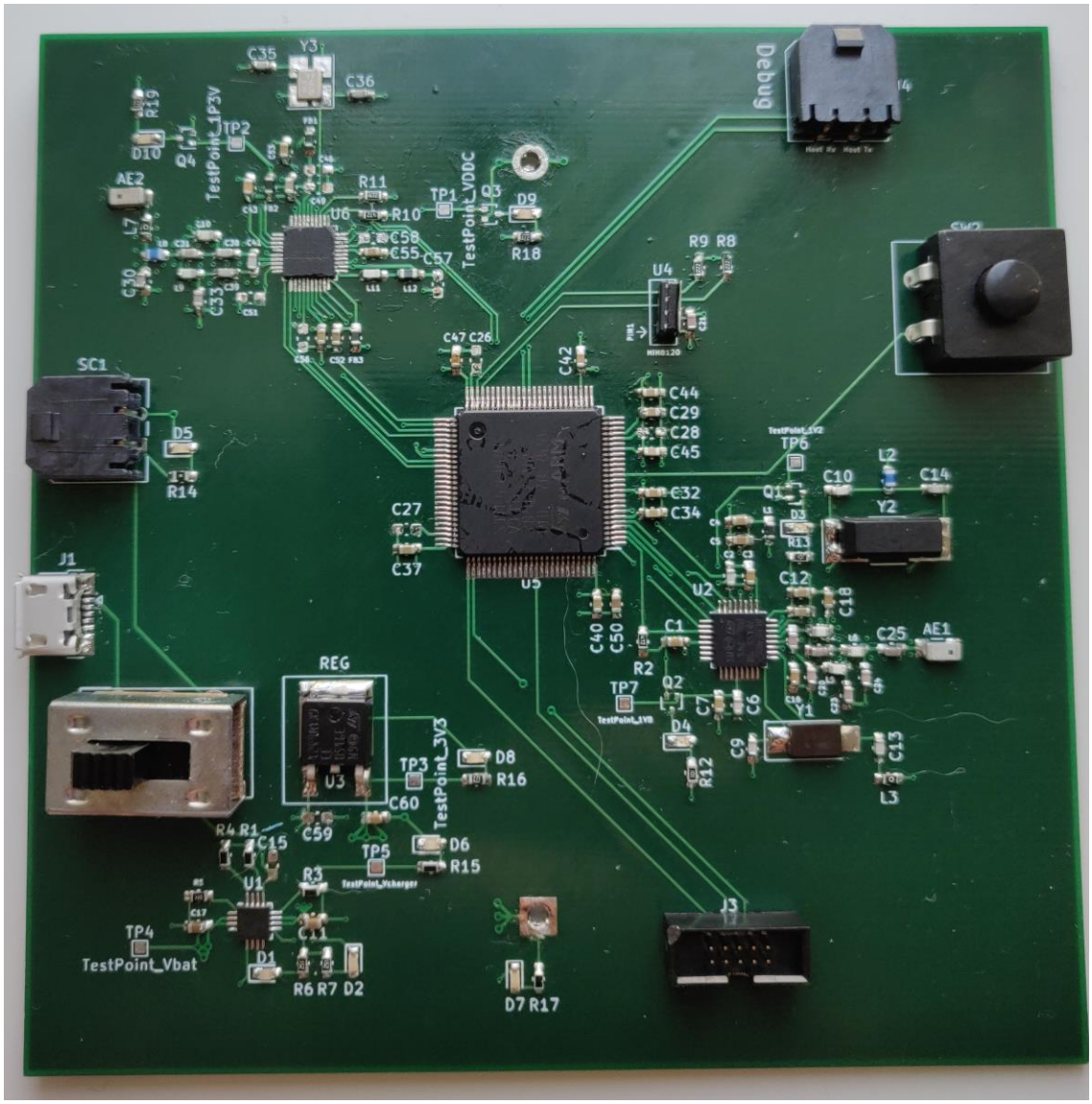

Figure 11 Front side of second revision of PCB.

Along with fixing the direction of USB connector's footpad and grounding of Y3 crystal, LEDs were added and additional UART used for printing data to console. Along with LEDs of Wi-Fi and BLE chips, some MOSFETs (metal-oxide-semiconductor field-effect transistor) had to be introduced to the board. The reason for that is the output voltages of Wi-Fi and BLE chips are too low for LED to function. Instead the output voltages are routed to the Gate of MOSFET to open a path of 3.3 Volts to LED. The footpads of leads of QFN packages are stretched to solder with C shaped soldering iron tip instead of hot air to minimize the possibility of damaging the chip. The center pads of QFN chips are

neglected. Also, the footpads of host processor's leads are stretched to ease the soldering. Notice that MOSFETs and some of the components are de-soldered to use for another board for debugging purposes.

When applying voltage to the board, Wi-Fi draws too much voltage, eventually heating up. When debugging the problem, MOSFETs were de-soldered. After the problem persisted, new board was assembled with components mounted that are only critical for powering the Wi-Fi chip. The problem persisted. The reason could be the neglected center pad which was to be connected to Ground. Other than center pad, the Wi-Fi chip has only one Ground pin. That may not be enough for the chip to function properly.

3.5 Conclusion

The author wanted to design the RF circuit, balun and power circuits himself because of educational purposes. Practically it is wiser to use modules that have the chip embedded made by third party companies or the chip provider itself. The design of the module is tiny, saving space. Modules are made EMC compatible and all the previous challenges related to RF design and power are abstracted away. This leaves the designer to only connect a module to the host device. The author did not want to order a third revision using ready-made modules because of lack of time, change in workplace (no soldering station) and the relatively expensive activity of constantly ordering new PCBs and components.

As higher-level software responsible for connecting the device logically to the network is interfaced to the hardware (e.g. Wi-Fi chip) via firmware, it makes the networking logic independent of hardware devices. For this reason, it is irrelevant which Wi-Fi chip (or other networking chip) is used when implementing an IoT sensor platform residing in home or office environments as the electromagnetic compatibility (EMC) between different Wi-Fi chips do not differ radically. As further manufacturing the custom board will prove to be costly and time consuming, introducing a commercial prototyping board seemed to be a sensible choice at the time being to continue developing and prototyping the software. The prototyping board chosen was LPC54018. It includes a Wi-Fi chip with embedded firmware readily interfaced to the microcontroller unit making it a suitable choice considering the requirements of the project. The project was continued using an

LPC54018 IoT prototyping board to explore the possibilities and APIs (application programming interfaces) of AWS FreeRTOS as reported in the next chapters.

4 Designing Embedded Software on IoT Sensor Platform

The conclusion about a self-designed board was to leave it as it is and continue with a commercial development board to explore the possibilities of the AWS IoT Core cloud service that works in co-operation with the Amazon FreeRTOS real-time operating system. The reasons for discarding a custom prototyping board was the cost and time it would have taken to fix design bugs, order, assemble and develop firmware for the third revision of the custom board. It was seen as a sensible solution to continue developing and prototyping software on a commercial prototyping board as the networking logic defined in higher level software is not bound to the hardware (e.g. Wi-Fi chips). The commercial prototyping board chosen was LPC54018 as it includes a Wi-Fi chip with embedded firmware readily interfaced to the microcontroller unit. This enabled the project to continue prototyping the software and networking architecture of an IoT sensor platform.

4.1 Background

Some embedded software was written for the self-designed board while waiting for the second revision of the PCB. The embedded software written was mainly firmware for ATWILC1000 Wi-Fi chip. A software design guide for the chip was used as a reference while writing firmware for the chip [25]. The documentation has guidelines for writing low level firmware (or drivers) for the Wi-Fi chip using SPI protocol and higher-level firmware on top of SPI drivers.

The abstraction layers of the firmware written consist of bus wrapper layer, communication layer and host interface layer.

The bus wrapper layer is responsible for calling low-level SPI driver software. HAL (hardware abstraction layer) library provided by ST (developer of STM microchips) was used for SPI driver functions.

The communication layer has functions for writing variable sized payloads to the chip and reading pending data from the chip. This layer also includes functions for controlling the chip such as resetting the chip and it includes all the status codes of the chip. The communication layer uses functions implemented in bus wrapper layer.

Host interface layer (HIF) is the layer responsible for the whole functionality of the chip such as addressing the device. This means writing to and reading from control registers, writing to and reading from data registers and handling the status codes provided by communication layer. This layer uses functions provided in the communication layer. Not much was implemented to the HIF layer apart from initializing the chip. The main reason was the non-functional PCB and lack of time reserved for completing the project.

Figure 12 shows a reference driver software architecture of the Wi-Fi chip. Bus interface is basically the same as “bus wrapper layer” software written by the author.

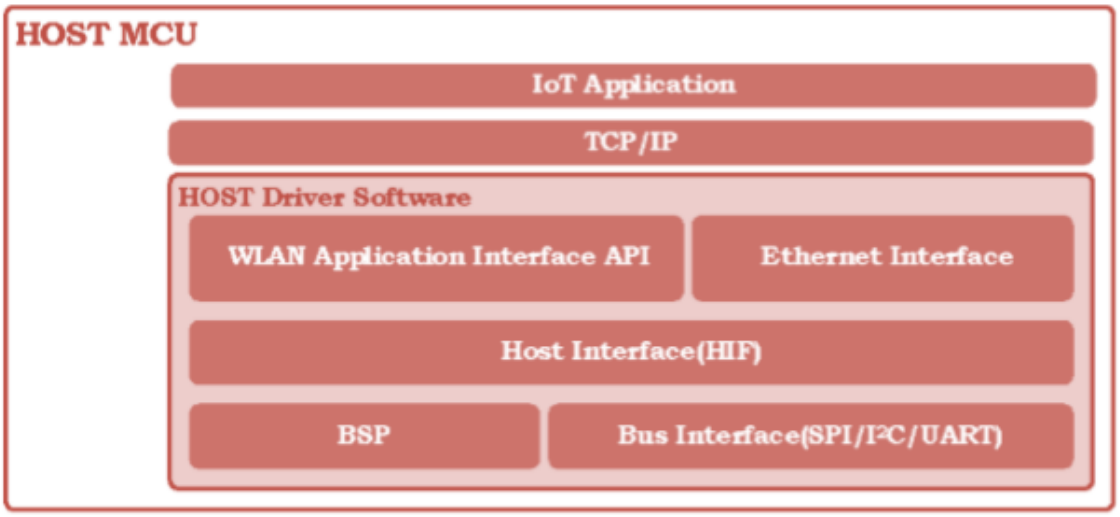

Figure 12 Example of software stack residing on host device controlling the ATWILC1000 Wi-Fi chip. [25]

The BSP (board support package; library of APIs that interface hardware of a board or chip) in Figure 12 is a firmware library for the host device controlling the Wi-Fi chip. HAL library provided by ST was used as BSP in this case. The communication layer does not exist in the reference software architecture illustration but was implemented to keep the software modular. The layers described were partly implemented while waiting for the second revision of PCB.

The WLAN Application Interface API layer would have common Wi-Fi functionality that are hardware independent. Most likely, the HIF layer would be responsible for actually implementing the functionality called by the operating system via the WLAN API layer. Ethernet Interface would be a network porting file that comes with FreeRTOS operating

system. It varies with different flavors of the operating system. For example, pure FreeRTOS + TCP/IP has a network interface porting file for this purpose. For Amazon FreeRTOS, Amazon provides guides for porting TCP/IP stack for Ethernet hardware [26].

As the project was continued with the LPC54018 prototyping board due to cost and time related difficulties of manufacturing a third revision of the custom board, developing software defining the functionality of Wi-Fi chip is not needed as it is already embedded in the prototyping board. This enabled to continue implementing and prototyping higher-level software logic for transporting sensor data to network. The higher-level software is not bound to hardware, making it possible to interface the custom board to higher-level software logic responsible of transporting sensor data to network when/if financial resources of designing and manufacturing the third revision of the custom board are allocated. Until then, the commercial prototyping board with Wi-Fi included is used to continue developing software logic for transporting sensor data to cloud.

4.2 Tools Used

This section is an overview of the tools used to explore Amazon Web Services in the context of IoT devices using a commercial development board.

4.2.1 Hardware Tools

The embedded software was downloaded on NXP's OM40008 development board that includes the LPC54018 microcontroller. The microcontroller has ARM Cortex-M4 CPU that can clock up to 180 MHz and has 360 kB of SRAM. The microcontroller includes 16 Mbytes of programmable external flash and plenty of serial peripherals for sensor communications. The project was prototyped with an HIH8120 sensor which is used for temperature and humidity data. The data is sent to AWS IoT Core cloud service.

4.2.2 Software Tools

The prototyping software was built upon a demo project provided by AWS. The project uses Amazon FreeRTOS as an operating system where among other tasks, networking tasks run upon. Amazon FreeRTOS is Amazon's extension of a popular open-source

real-time operating system called FreeRTOS. It is extended with APIs that help with connecting microcontrollers to Amazon Web Services. A simple polling I2C driver is programmed for fetching data from HIH8120 sensor. Polling software is a type of software that asks (or polls) another software or device for information constantly or in fixed intervals. The data is sent to the cloud using the MQTT protocol. MCUExpresso IDE (integrated development environment) is used as the software development environment. Otherwise slow and buggy environment, the reason it is used is good compatibility with NXP's devices and well-working debugging tools.

4.3 Setting up Developing Environment

The hardware setup of demonstrational environment can be seen in Figure 13.

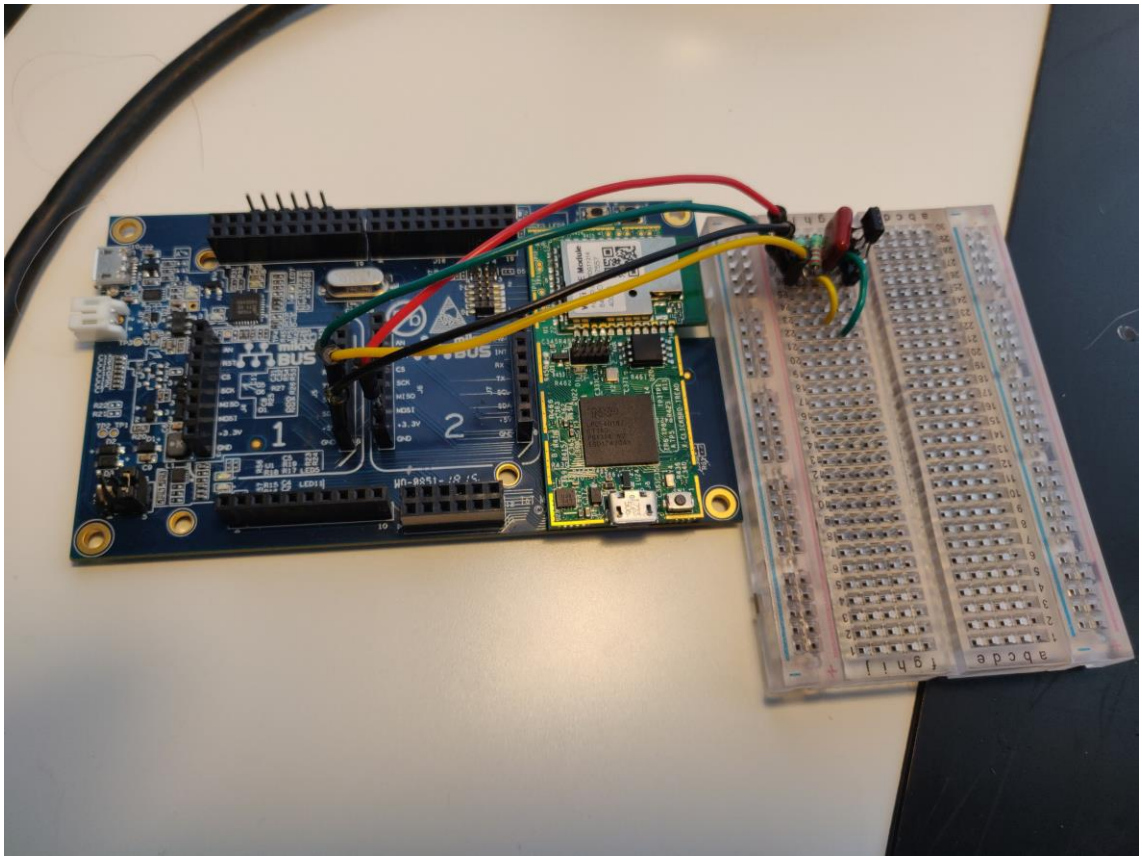

Figure 13 Development board setup

The sensor along with decoupling capacitor and two pull-up resistors for I2C functionality was implemented on breadboard. The HIH8120 sensor was soldered to additional pins

so it could be attached to the breadboard. The software part of the setup was done with the help of tutorial provided by AWS for the same developing board used in the project [27]. This worked as an excellent jump start for developing software. The tutorial included making certificate and public and private keys. Certificate and private key were added to the software to send encrypted data to the cloud. The frame of software of this project was done using a demo project for LPC54018 development board that comes with MCUExpresso IDE. The instructions on how to set up the demo project are included in the Amazon Web Services documentation [27].

4.4 Constructing Software

The original demo project demonstrates sending and fetching encrypted JSON (JavaScript object notation) formatted data to and from the cloud. This is done by using one task to take care of subscribing as MQTT (Message Queuing Telemetry Transport) publisher and another as MQTT subscriber. The publisher task is responsible for sending data to the cloud. The subscriber task gets a notification whenever the state of a JSON document in the cloud changes and fetches the new data. The JSON document that represents the current state of publishing device (i.e. the device that sends data to the cloud) is called the device's "shadow" in AWS terminology. The demo project includes Wi-Fi drivers for the Wi-Fi module that is part of the development board and all the other drivers necessary to fully operate the peripherals that are originally part of the development board.

As the project acts as an IoT sensor platform, its main objective is to publish the data to the cloud, not subscribe to receiving the data. Although, when working with more complex sensor networks, other sensor devices might need to know about the state of other devices, this is not necessary in this project. The program was modified to only publish JSON formatted data to the cloud. The data sent as a payload to the cloud is temperature and humidity data fetched from HIH8120 sensor. There is a task for fetching data from the sensor that is set to poll the sensor after every few seconds. The data retrieved is embedded to the JSON formatted string and sent to the cloud using MQTT protocol.

There are two header files that need to be configured before the device can connect to the cloud. The certificate and private key are to be added to aws_clientcredential_keys.h

and the MQTT broker endpoint address along with credentials of the access point the Wi-Fi module is to be connected need to be defined into aws_clientcredential.h.

After downloading and running the software on the development board, the device should connect via the developer-defined WLAN access point to the AWS IoT Core web service. From the web page of the cloud service, the developer can now see the state of the device which is a JSON document called “shadow” changing state.

At this point, the development board acting as an IoT sensor platform publishes sensor data to the cloud using MQTT protocol. To display the data to the user via some kind of a user interface, a back-end implementation needs to be implemented first which is responsible for subscribing to the state of the device as an MQTT subscriber and retrieving data for the user to see. This implementation is described in later sections.

4.5 Anatomy of Software on IoT Platform

The source code of this project is large. The size of compiled software is over 340 kB. This includes the drivers for the peripherals of LPC54018, the drivers and software stack for the Wi-Fi module and Amazon FreeRTOS with built-in MQTT support. It probably needs some optimizations when using the software in the field because low-power and low-cost embedded devices that are meant to measure environment for long periods of time are seldom equipped with memory modules this large. Then again, they probably do not need Wi-Fi support either and rely on some low-power networking technologies such as NB-IoT or LoRa.

The whole software is quite complex to dig into but then again it is almost out of the box and is very easy to get up and running and start prototyping. This is true at least in ready-made development board environments.

In Figure 14, the high-level flow of key tasks responsible of sending data to cloud can be seen.

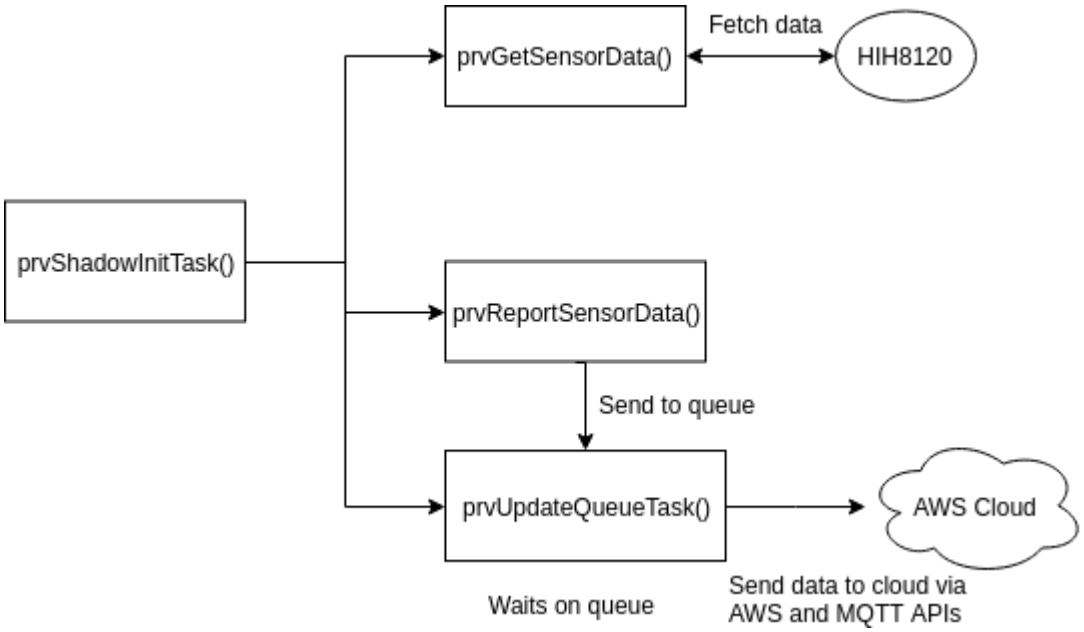

Figure 14 High level illustration of key tasks responsible for sending data to cloud.

Besides initializing data communication tasks, prvShadowInitTask() also initializes the task responsible for connecting the device to the cloud. Following the initialization, prvGetSensorData() starts fetching data from sensor, prvUpdateQueueTask() starts waiting on a queue and prvReportSensorData() embeds temperature and humidity data to JSON string and sends the buffer pointing to JSON formatted string to queue. The task gets access to sensor data via “Getter” functions provided in the file responsible for sensor communications (HIH8120 in the diagram). The latest fetched data is stored in the sensor communication file to global variables so tasks can fetch the data from these variables whenever they are ready. Once prvUpdateQueueTask() receives data from the queue, the task is responsible for calling functions that send the JSON formatted string to the cloud using the MQTT application level protocol.

4.6 Conclusion

Amazon FreeRTOS combined with AWS IoT Core provides a fast start for prototyping projects in embedded environments that are to be connected to the Internet. It is relatively easy to use and fast toolset to get project going. It takes some time to dig in as the software is large and quite complex. For actual battery-powered devices in the field it is probably better that the developer integrates API libraries to the software. Using a

demonstrational project that is compatible with a certain development board is a great way of getting to know the software, but it is probably too heavy to use in practice. The C-language API library meant for embedded environments that can be integrated to the Amazon FreeRTOS project can be found from GitHub [28]. It includes APIs for MQTT and TLS (transport layer security) networking protocols. As the library includes application layer (think OSI model) APIs, it is the developer's responsibility to choose the networking hardware (Wi-Fi, Bluetooth, LoRa, NB-IoT, etc.). If the networking module does not include software stack for communication, it is also the responsibility of the developer to implement one. In the case of Wi-Fi modules that do not include TCP/IP software stack, FreeRTOS Plus TCP can be ported to Amazon FreeRTOS project. AWS provides guides for the porting subject, which was referenced earlier in Section 4.1. FreeRTOS Plus TCP is a TCP/IP software stack for FreeRTOS kernel. It uses software abstractions called "sockets" derived from POSIX standard used in Unix operating systems which are used to establish and maintain connections and sending data between network endpoints.

The IoT sensor platform will connect directly to AWS IoT Core cloud services to publish sensor data. For the end user to access the data from web browser or mobile application, back-end software needs to be implemented which can retrieve sensor data from cloud for the end user to see. The back-end software will be used to subscribe to the MQTT topic residing in the AWS IoT Core cloud. Back-end software will get a notification every time new data is published to the topic and will retrieve the data for the front-end to render once a notification from the cloud is received. The implementation of back-end software and front-end software is discussed in the next chapter.

5 Designing Web Application for IoT Sensor Platform

In previous chapters, the making of an IoT sensor platform implementation sending data to the AWS IoT Core cloud services using the MQTT protocol was described. For the end user to get access to the data the IoT sensor platform sends, some front-end implementation needs to be made that renders the data on, for example, a web browser. The front-end application will fetch the data from the back-end software that is connected to the cloud and is responsible for fetching the data the IoT sensor platform has sent to the cloud. The web application will give an interface to monitor the validity of the data for testing purposes and later for the user to get access to the data.

This section documents an implementation of bringing the data the IoT sensor platform device sends to the cloud for an end user to see.

5.1 Background

IoT devices often have a centralized user-interface accessed from the web. Its function is to interpret and display the data the devices send to the back-end. This project also has a web application implementation that interprets and displays the data to the user. AWS IoT Core along with a program that is responsible of fetching data from the cloud implemented with .NET Core together act as a back-end. The front-end side of the web application was implemented with React. The front-end is the user-interface which can be accessed using a web browser (from a desktop computer) in this project.

The implementation of the back-end in this project refers to the program that is responsible for fetching data from the cloud. In the big picture the cloud is also part of the back-end but as the cloud along with its servers is not self-implemented and is completely abstracted away by Amazon Web Services, the design and implementation of the back-end in this chapter refers to the software that fetches data from the cloud.

5.2 Implementation of Back-end Application

The software responsible for fetching sensor data from the cloud was implemented with .NET Core. It is an open-source successor to .NET framework developed by Microsoft.

Although this software framework is widely used in back-end implementations in enterprises, .NET framework is starting to make room for NodeJS built on JavaScript engine [29]. This is illustrated in Figure 15.

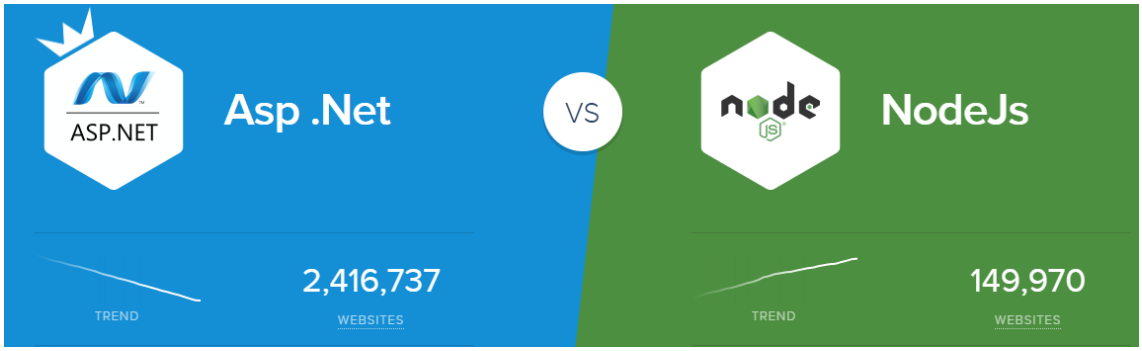

Figure 15 Asp.Net compared to NodeJS in number of websites and trend. [30]

Figure 15 compares ASP.NET and NodeJS frameworks. ASP.NET is a server-side software framework for building web applications which itself is part of .NET framework. Although the number of web applications built upon ASP.NET is dominant compared to web applications build upon NodeJS, the trend of using NodeJS is growing rapidly.

Practically there are not many reasons for one to use .NET Core over frameworks built on Node JS, especially with small web applications as this. The only reason it is used in this project is testing out the framework as the author lacked previous experience of .NET software framework. .NET Core can easily be integrated with React but as React is a JavaScript library and .NET is natively not used with JavaScript, it again makes more sense to use back-end frameworks that are built on NodeJS which uses the JavaScript engine as it makes it possible for a developer to work with tools that are similar. One of the reasons .NET is integrated with React in practice is when an enterprise already has a large back-end ecosystem implemented using .NET software framework but the software of client side user-interface needs to be updated to using modern front-end tools like React. Starting a completely new web project using React as a front-end tool and implementing back-end using .NET probably does not make sense. As nowadays it is possible to use JavaScript all the way through back-end to front-end, it makes sense to use this possibility as the integration between back-end and front-end is easier to implement and maintain using similar tools as opposed to using different tools on every subsection of web development. The possibility to program different parts of the project using

tools that can interpret the same programming language (read: JavaScript) makes it easier for a developer to maintain and develop software in full-stack. This means the developer is responsible for developing both server-side back-end and client-side front-end implementations.

The back-end application subscribes to the MQTT topic in AWS IoT Core cloud and fetches the data that the sensor publisher to the topic. A topic is essentially a string that enables the MQTT broker (I.e. the server) to differentiate between JSON documents representing devices. Figure 16 illustrates communication on high level between the back-end application and sensor platform.

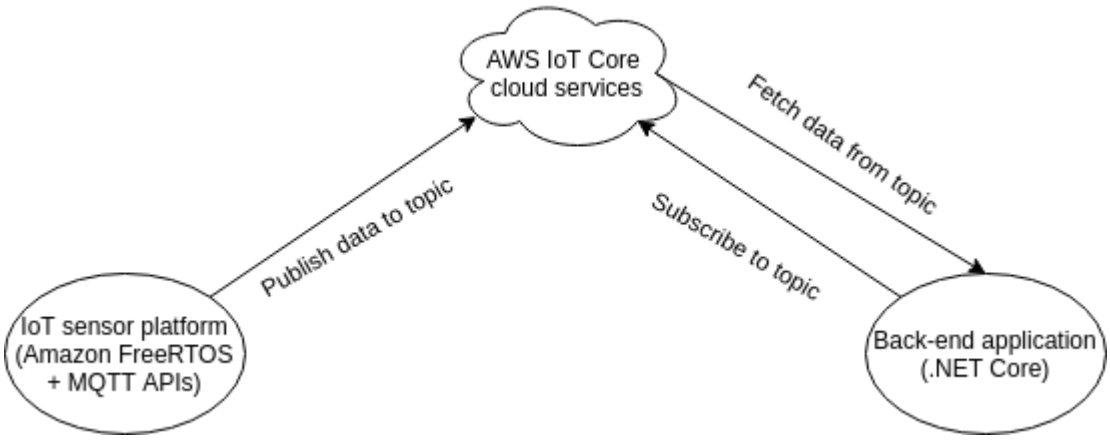

Figure 16 High level representation of MQTT data flow.

The back-end application uses the M2Mqtt library which includes APIs for MQTT communication when using .NET software framework

5.3 Implementation of Front-end Application

The front-end application that acts as a user-interface to access the sensor data is implemented with React. React is a popular JavaScript library used to develop front-end user-interfaces and is widely used to create modern single-page applications (SPA). React was created at Facebook.

Modern responsive web page applications are often implemented as single-page applications. By single-page application it is meant that when a user browses the web page, the program of the whole web page is downloaded on the client's device. Generating the HTML (Hypertext Markup Language) for the browser to render the web page is also done on the client's device. The cycle of browsing web applications using the single page application technology is illustrated in Figure 17.

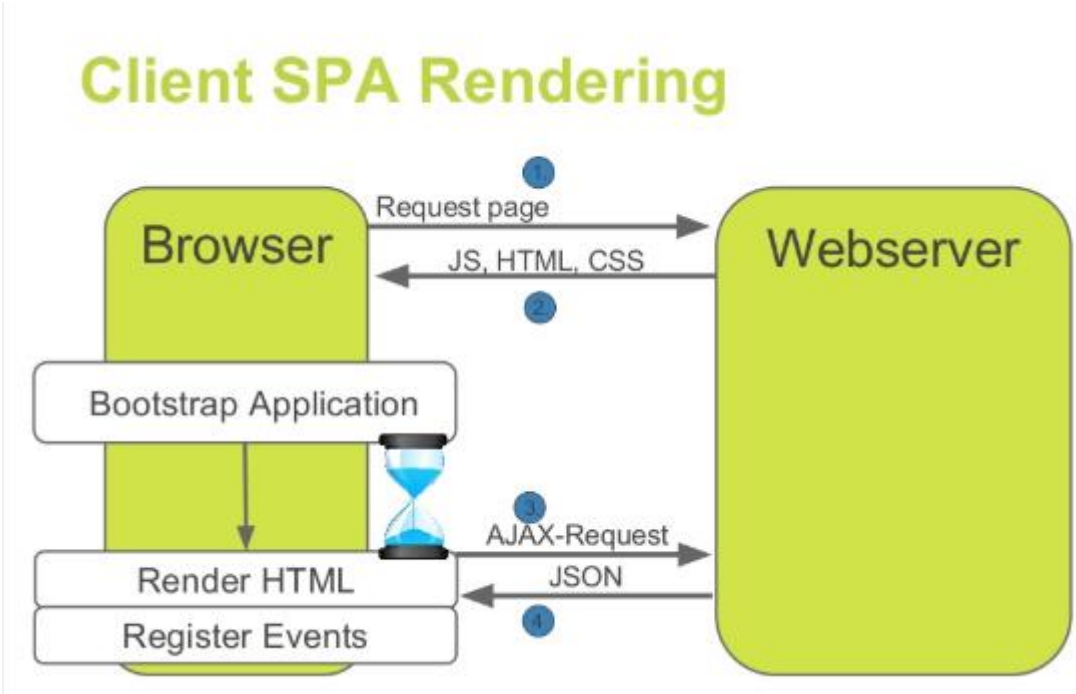

Figure 17 Cycle of SPA web applications. [30]

Traditional web page applications do not work this way. In traditional web applications, HTML files are generated on the server (back-end) and browser on client computer re-fetches the HTML files from the server after every modification done on the web page. By modification it is meant some user interaction that results in needing to generate a new element to render. So, when the user uses the web page, the browser on the client computer constantly communicates with the server to fetch a newly generated HTML file from the server that the browser can render on client side. Because modern single-page applications download the whole software of a web application on the client's computer and generating new HTML code for the browser to render is also done on the client side without the need to constantly communicate with the server when using the web page, it results in more responsive and smooth user experience. This is true at least with smaller web applications.

5.4 Conclusion

Implementing a web application for the user to view data sent by a sensor in a human readable format consists of back-end and front-end implementations. The function of back-end software in this project was to subscribe to an MQTT topic that resides in the AWS IoT Core cloud services and fetch the data whenever the sensor publishes new data to the topic. The back-end was implemented using .NET Core software framework. The software framework was chosen for testing purposes and practically it would have been wiser and easier to implement it using a back-end software framework built on Node JS. A great example of this kind of framework would be Express JS.

The function of front-end software was to represent sensor data visually and interpreted in a human readable way. The front-end implementation was done using a popular JavaScript library created for this purpose called React JS.

6 Summary

The project represented a life cycle of developing an IoT sensor platform from a practical point of view in which some of the technologies were introduced and demonstrated in each section of the development cycle. The market of optional products to use is wide in every section of the developing cycle. It is up to the developer in each area of developing to make decisions on which technologies and tools to use to satisfy the needs and specifications of the complete product.

The project started with designing hardware for an IoT sensor platform and paper introduces the tools for electronics hardware design, places to order electronic components and PCBs and some circuit design from a practical point of view backed up with some circuit theory.

Developing embedded software was started while waiting for the second revision of the PCB but due to the difficulties with the PCB, the deadlines of completing the project, lack of financial resources allocated for the project and workplace change and lack of tools for continuing hardware designing and debugging because of a change of workplace, developing embedded software was continued on a commercial development board. Amazon FreeRTOS, an extended version of popular real-time operating system used in embedded systems named FreeRTOS and its compatibility with Amazon Web Services was investigated here.

The paper goes through the back-end and front-end implementations and some optional technologies which could have been used in the project. The back-end part had software implemented which used MQTT APIs for subscribing to Amazon Web Services and retrieving data the sensor platform had sent there. The front-end part had a user interface implemented using modern technologies for an end user to access the data.

References

1 I2C-bus specification and user manual (UM10204). NXP Semiconductors. 2014. p. 8.

2 STM32F429 reference manual (RM0090). ST Microelectronics. p. 917.

3 Figure illustrating setup of I2C, SPI and UART devices [Internet], Figure available from: https://www.mbtechworks.com/hardware/raspberry-pi-UART-SPI-I2C.html

4 Baid A, Mathur S, Seskar I, Raychaudhuri D. Spectrum MRI: Towards Diagnosis of Multi-Radio Interference in the Unlicensed Band. 2011.

5 Pinkle C. The Why and How of Differential Signaling [Internet]. AllAboutCircuits.com. 2016. Article available from: https://www.allaboutcircuits.com/technical-articles/the-why-and-how-of-differential-signaling/

6 Kinnaird C. Differential signaling best practices [Internet]. ecnmag.com. 2012. Article available from: https://www.ecnmag.com/article/2012/12/differential-signaling-best-practices

7 STM32F429 datasheet (DM00071990). ST Microelectronics. p. 91.

8 MQTT Topics & Best Practices – MQTT Essentials: Part 5 [Internet]. The HiveMQ Team. 2019. Article available from: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

9 Arar S. How a Ground Plane Reduces PCB Noise [Internet]. AllAboutCircuits.com. 2019. Article available from: https://www.allaboutcircuits.com/technical-articles/how-a-ground-plane-reduces-pcb-noise/

10 ATWILC1000 Hardware Design Guidelines. Microchip Technology. p. 8.

11 PCB design guidelines for the BlueNRG and BlueNRG-MS devices (AN4630). ST Microelectronics. p. 9.

12 LC Baluns In Action [Internet]. RobustCircuitDesign.com. 2016. Article available from: https://www.robustcircuitdesign.com/signal-chain-explorer/lc-baluns-in-action/

13 Phase Relationships in AC Circuits [Internet]. Georgia State University. Article available from: http://hyperphysics.phy-astr.gsu.edu/hbase/electric/phase.html

14 Using Baluns and RF Components for Impedance Matching [Internet]. Coilcraft. Article available from: http://www.oh3ac.fi/coilcraft_baluns.pdf

15 The 50 Ohm Question: Impedance Matching in RF Design [Internet]. AllAboutCircuits.com. Article available from: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/real-life-rf-signals/the-50-question-impedance-matching-in-rf-design/

16 Schematic diagram of STEVAL-IDB006V1 evaluation board. ST Microelectronics. Available from: https://www.st.com/en/evaluation-tools/steval-idb006v1.html#resource

17 Understanding Reflections and Standing Waves in RF Circuit Design [Internet]. AllAboutCircuits.com. Article available from: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/real-life-rf-signals/understanding-reflections-and-standing-waves-rf-circuit-design/

18 Effects of Impedance Matching and Switch Quality on RF Test System Performance. National Instruments. 2007. Article available from: http://www.ni.com/product-documentation/5779/en/

19 “What Is a Transmission Line?” [Internet]. AllAboutCircuits.com. Article available from: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/real-life-rf-signals/what-is-a-transmission-line/

20 Characterizing your production process for controlled impedance builds. Polar Instruments. 2001. Paper available from: https://www.polarinstruments.com/support/cits/AP129.pdf

21 PCB Calculator tool which is part of KiCad software.

22 Characteristic Impedance [Internet]. AllAboutCircuits.com. Article available from: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-14/characteristic-impedance/

23 ATWILC1000B-MUT Datasheet. Atmel. pp. 11-12.

24 BlueNRG-MS Datasheet (DM00141263). ST Microelectronics. pp. 22.

25 ATWILC1000 SPI Wi-Fi Link Controller User Guide. Atmel.

26 Porting a TCP/IP Stack [Internet]. Amazon Web Services. Documentation available from: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-tcp.html

27 Getting Started with the NXP LPC54018 IoT Module. Amazon Web Services. Documentation available from: https://docs.aws.amazon.com/freertos/latest/userguide/getting_started_nxp.html

28 AWS IoT Device SDK Embedded C. Amazon Web Services. C-library available from: https://github.com/aws/aws-iot-device-sdk-embedded-C

29 ASP.NET vs NodeJS [Internet]. SimiliarTech. Article available from: https://www.similartech.com/compare/asp-net-vs-nodejs

30 Amend D. Client vs Server Rendering. 2014. Slideshow available from: https://www.slideshare.net/DavidAm/client-vs-server-rendering