

BotSpine - A Generic Simple Development Platform of Smartphones and Sensors or Robotics

A Thesis Submitted to the College of
Graduate and Postdoctoral Studies
In Partial Fulfillment of the Requirements
For the Degree of Master of Science
In the Department of Electrical and Computer Engineering
University of Saskatchewan
Saskatoon

By

Yating Liu

Copyright Yating Liu, January, 2017 All Rights Reserved

Permission to Use

In presenting this thesis/dissertation in partial fulfillment of the requirements for a Master of Science degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis/dissertation in any manner, in whole or in part, for scholarly purposes may be granted by Dr. Li Chen who supervised my thesis/dissertation work or, in his absence, by the Head of Department of Electrical and Computer Engineering or the Dean of College of Graduate and Postdoctoral studies in which my thesis work was done. It is understood that any copying or publication or use of this thesis/dissertation or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis/dissertation.

Requests for permission to copy or to make other uses of materials in this thesis/dissertation in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering
3B48 Engineering Building
57 Campus Drive, University of Saskatchewan
Saskatoon, SK S7N 5A9 Canada

OR

Dean of College of Graduate and Postdoctoral Studies
Room C180 Administration Building
105 Administration Place, University of Saskatchewan
Saskatoon SK S7N 5A2 Canada

Abstract

The Internet of Things (IoT) emergence leads to an “intelligence” technology revolution in industrial, social, environmental and almost every aspect of life and objectives. Sensor and actuators are heavily employed in industrial production and, under the trend of IoT, smart sensors are in great demand. Smartphones stand out from other computing terminals as a result of their incomparable popularity, mobility and computer comparable computing capability. However, current IoT designs are developed among diverse platforms and systems and are usually specific to applications and patterns. There is no a standardized developing interface between smartphones and sensors/electronics that is facile and rapid for either developers or consumers to connect and control through smartphones.

The goal of this thesis is to develop a simple and generic platform interconnecting smartphones and sensors and/or robotics, allowing users to develop, monitor and control all types of sensors, robotics or customer electronics simply over their smartphones through the developed platform. The research is in cooperation with a local company, Environmental Instruments Canada Inc. From the perspective of research and industrial interests, the proposed platform is designed for generally applicable, low cost, low energy, easily programmed, and smartphone based sensor and/or robotic development purposes.

I will build a platform interfacing smartphones and sensors including hardware, firmware structures and software application. The platform is named BotSpine and it provides an energy-efficient real-time wireless communication. This thesis also implements BotSpine by redesigning a radon sniffer robot with the developed interface, demonstrated that BotSpine is able to achieve expectations. BotSpine performs a fast and secure connection with smartphones and its command/BASIC program features render controlling and developing robotics and electronics easy and simple.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Dr. Chen for the continuous support of my MSc. study and related research, for his patience, motivation, and immense knowledge. His guidance was indispensable throughout the research process, as well as throughout the process of writing this thesis. I could not have imagined having a better advisor and mentor for my M.Sc. study.

My sincere thanks also go to Mr. Kai Kaletsch, who provided me an opportunity to join Environmental Instruments Canada as co-op and who gave access to the industrial facilities and the product designs related to the research. I would also like to thank the rest of the team of EIC, Sepehr Khatir, Scott Bulbeck and Cheryl Holst. Without their precious support, it would not be possible to conduct this research.

Last but not the least, I would like to thank my family: thank you to my parents and brother for supporting me spiritually throughout the writing of this thesis and throughout my life in general.

Table of Contents

Permission to Use	i
Abstract.....	ii
Acknowledgements.....	iii
Table of Contents.....	iv
List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Background and Motivation	1
1.2 Overview of the Company-specific Motivation.....	5
1.3 Thesis Aims and Objectives.....	6
2 Background	7
2.1 The Internet of Things	7
2.2 Challenges and Trends.....	10
2.3 Wireless Connection	13
2.3.1 Connection Choices.....	13
2.3.2 Bluetooth Low Energy	15
2.4 Wireless Microcontrollers.....	30
2.4.1 Texas Instrument Wireless Connectivity.....	31
3 BotSpine Hardware and Software Development.....	34
3.1 Design Concept.....	34
3.2 Hardware Design	37
3.2.1 CC2541.....	37
3.2.2 External I/O	38
3.2.3 PCB Antenna.....	38
3.2.4 Hardware Board Layout	41
3.3 Firmware Design.....	42

3.3.1	BotSpine Firmware Version 1.0.....	46
3.3.2	BotSpine Firmware Version 2.0.....	51
3.4	Software Design.....	54
3.5	BotSpine Performance	57
4	Implement BotSpine in Radon Sniffer	59
4.1	Radon Sniffer Principles	61
4.2	Radon Sniffer Hardware	63
4.3	Radon Sniffer Firmware	65
4.4	Radon Sniffer Performance.....	67
5	Conclusion.....	72
	References.....	74
	Appendix.....	79

List of Figures

Figure 2.1 <i>Bluetooth Stack</i>	17
Figure 2.2 <i>Advertising and Data Channels</i>	17
Figure 2.3 <i>State diagram of the Link Layer state machine</i>	20
Figure 2.4 <i>Connection Event Diagram</i>	20
Figure 2.5 <i>ATT architecture</i>	24
Figure 2.6 <i>LE Pairing Phases</i>	25
Figure 2.7 <i>GATT Profile hierarchy</i>	28
Figure 2.8 <i>Integration variations for cloud connectivity</i>	30
Figure 2.9 <i>CC2541 Functional Diagram [38]</i>	33
Figure 3.1 <i>BotSpine Logo</i>	35
Figure 3.2 <i>Development Flow Comparison of BotSpine (right) and General (left)</i>	35
Figure 3.3 <i>XDATA Memory Space</i>	39
Figure 3.4 <i>Antenna Dimensions</i>	39
Figure 3.5 <i>BotSpine PCB</i>	41
Figure 3.6 <i>OSAL Tasks Array</i>	43
Figure 3.7 <i>OSAL Task Loop</i>	43
Figure 3.8 <i>BotSpine Custom Profile</i>	47
Figure 3.9 <i>BotBasic Custom Profile</i>	53
Figure 3.10 <i>BotSpine App Scan Page</i>	55
Figure 3.11 <i>BotBasic App Command Page</i>	55
Figure 3.12 <i>BotBasic App lined program example</i>	56
Figure 4.1 <i>VS472 Radon Sniffer System Schematic</i>	61
Figure 4.2 <i>VS472 Radon Sniffer Assembling</i>	64
Figure 4.3 <i>VS472 Connected to Generic BLE Apps</i>	68
Figure 4.4 <i>VS472 Characteristic Writing and Reading</i>	68
Figure 4.5 <i>One Minute Counts Reading from BotSpine and previous Radon Sniffer</i>	69

Figure 4.6 *Radon Concentrations Comparison in Sniffer Mode for LONG (5-min) Average* 70

List of Tables

Table 2.1 <i>Link Layer packet format</i>	20
Table 2.2 <i>Possible permissions</i>	24
Table 2.3 <i>LE Pairing Method</i>	27
Table 2.4 <i>TI Wireless MCUs Comparison</i>	32
Table 3.1 <i>BotSpine I/O Mapping</i>	40
Table 3.2 <i>Antenna Dimension Values</i>	40
Table 3.3 <i>BotSpine Version 1.0 Functions</i>	47
Table 3.4 <i>BotSpine Version 1.0 Function Parameters</i>	50
Table 3.5 <i>BotSpine Version 1.0 Error Code</i>	50
Table 3.6 <i>Power Consumption Comparison</i>	58
Table 4.1 <i>Characteristic ID and ID Values</i>	66

1 Introduction

1.1 Background and Motivation

The Internet of Things (IoT) has become increasingly appealing to industry and even hobbyists since it emerged to the public. Its rising popularity is predominantly derived from the concept of a widely available connection and easily accessible interaction network. In the IoT world, people can make better decisions and have more convenience in life by virtue of the comprehensive information collected detailing every aspect of their lives.

In the future IoT era, people will be able to obtain information about anything or from anywhere through the “Things” network, regardless of time or location. Moreover, we envisage that the network will be capable of automatically generating, transmitting and analyzing information, thus providing the most optimal solutions to any encountered circumstance and executing the optimized proposal. With many appealing benefits, companies and professionals are devoting time and effort toward making progress in IoT. The heating trend of IoT also inspires universities and researchers to introduce more IoT related projects for students, allowing them to gain valuable professional experience [1]. IoT is projected to have a bright future, with a prediction of approximately 50 billion devices connected into IoT by 2025 [2].

One of the most significant technologies of IoT is the machine to machine (M2M) interaction among but not limited to computers, embedded processors, smart sensors, actuators and mobile devices [3]. To make the “Things” interactive, connectable to the network and stand-alone capable, machines must not only have the intelligence to conduct their own tasks, but must also be able to communicate with each other without technical barriers.

Smart sensors play an indispensable role in the IoT by converting analog or natural signals into digitals. Multi-sensors and microcontrollers are more and more commonly implemented in industrial products and home appliances. Sensors and actuators are the most basic and primary electric mechanical elements in most machines, including robotics.

In addition to sensors, robotics and electronics will also be designed to be intelligent and connectable in terms of raw data collection and actions execution. Robotics are becoming pervasive in the home setting with applications such as cleaner machine robotics, in social environments as telepresence robots and nursing robots for the elderly, and for commercial or entertaining purposes as aerial imaging platforms. Experts believe that robotics will be commonplace within the next few decades. “AI and robotics will be integrated into nearly every aspect of most people’s daily lives” [4] says Aaron Smith, senior researcher at the Pew Research Center’s Internet Project. Given the fact that there was a significant price-performance ratio drop of sensors and electronics around the early 21th century, the concept of building customized automated applications with inexpensive electronics and robotics appears to pique the interest of people [5].

With the happening mega-trend development of IoT, it will be possible for people in the future to access and control various types of robotics and machines by solely using one smart control monitor. Amongst a range of popular smart processing terminals, smartphones are one of the most popular and preferable choices for remote controlling, in comparison to PCs and laptops, now and in near future; the portability and powerful multi-functionality of the smartphone sets it apart from its competitors.

As mentioned above, M2M technology requires smart sensors or smart actuators, which are capable of automatically executing their tasks as well as conducting barrier free communication platforms for processing terminals, in this case referring to smartphones. In order to make sensors and robotics smart, the addition of embedded microprocessors and microcontrollers is considered one of the most common and popular approaches.

The major concerns of running continuous and real-time sensing applications with smart sensors or actuators are energy efficiency and data computing load problems. Many research papers [6] [7] [8] choose smartphones as their data sensing processing terminals in order to address these concerns. Smartphones have competitive computing capability, as compared to computers; meanwhile, the wireless techniques, mobility and widely spread usage render smartphones incomparable real-time sensing applications.

The next pertinent question to address will be how people easily connect their smartphones to sensors or robotics and also connect to the Internet. In terms of wire connection, considering that

smartphones have different data interfaces like USB and earphone connectors depending on their operation systems and manufacturers, there is no universal wire interface between smartphones and the sensor network [9]. However, the wireless sensor network (WSN) uses IEEE 802.15.4 standard wireless technology, which smartphones are not equipped with, as well as other wireless technologies, such as Bluetooth Low Energy (BLE) and Wi-Fi. While the sensors are connected to smartphones wirelessly, it is also possible to connect the Internet network with smartphone mid-station.

Currently, there may be development tools or interfaces in the market that are able to connect smart phones and sensors or robotics [6]. Developers and researchers tend to choose Raspberry Pi [10] or Arduino [11] as their development tools to target specific applications because of their multi-functional potential and flexibility. People tend to make great innovations with Raspberry Pi and Arduino boards; however, both of these require time and effort to develop, even with a professional programming skills background, and may require additional hardware or software setups. For instance, Raspberry Pi employs the Linux system, which may require additional time for new-to-Linux users to get started. More importantly, in terms of mass production in industrial implementation, the cost is of key consideration. Compared to commonly used microcontrollers, the powerful functionality Raspberry Pi and Arduino appear to be over-qualified for low cost, rapidly developed industrial end products.

Electric Imp Inc. provides another innovative service solution for IoT [12]. Electric Imp focuses on business solutions for well-managed product-to-network connections and powerful cloud computing. The connection of Imp is Wi-Fi, with a prominent focus on data processing for cloud computing services. Electric Imp provides industrial and enterprise grade level services. It appears to be an acceptable option for companies to build efficient and high quality IoT products; however, it is still not a universal interface solution for generic smartphone-sensor adaptations.

The lack of a universal wireless interface for smartphones and smart devices is a thorny issue of sensor-device performance rapid development and monitoring in industry or consumer-grade project developments. For example, some radiation detection companies have developed some accurate and efficient radiation detecting sensors, but they usually have to devote sufficient time and effort in order to work on connecting sensors to smartphones. Building interface platforms for

each individual project is simple, but requires that engineers and computer software professionals devote time and effort.

A universal interface capable of simply connecting various types of smartphones and sensors/actuators wirelessly will efficiently improve product development cycles and efforts in industry. Additionally, it will serve as a generic design protocol by effectively reusing available resources or adapting new designs. The universal interface is better with cross platform and wireless capabilities. Software development platforms or operating systems corresponding to each specific intelligent robotic product are different. Furthermore, if people want to monitor and operate several kinds of robotics in the future, they will likely require several controllers with different operating systems, some of which may not even be compatible with smart phones. Wireless technology renders the cross platform feature easier to implement, due to interface differences between smartphones and various other systems and differences between sensors from diverse manufacturers.

We would call it BotSpine because the interface is expected to act similar to a spine, which is a core message communication channel between the body (sensors/robotics) and the brain (smartphones). The body could be as small as sensors or any electronics or as complicated as a robotic machine, while the brain refers to a smart phone, which takes charge of controlling, monitoring and data processing tasks. BotSpine performs an identical job with the actual spine and serves as a mediator between sensors and smartphones. BotSpine receives high-level language messages and commands from smartphones and transfers them to machine level hardware signal controls on the actuators. Meanwhile, BotSpine obtains stimulation hardware signals from sensors or robotics and sends useful information back to smartphones. In this case, sensors and robotics reduce and share data load and energy consumption with smartphones, and smartphones can act as a widely adapted sensor/robotic controller and developer. With BotSpine, an inexpensive, generic, ready-to-use interface with smart phone console application, users can build their own project within a day and without assistance.

1.2 Overview of the Company-specific Motivation

There are also some industrial motivations behind this thesis project. The thesis research is in cooperation with a local company, Environmental Instruments Canada Inc.

Environmental Instruments Canada (EIC) Inc. is a leading designer and manufacturer of radiation detection equipment, primarily for the uranium mining industry. To expand their business, EIC has developed personal radiation detection devices for non-expert users. In order to meet the interface, usage and price expectations of non-expert users, EIC uses popular mobile computing platforms like smartphones as the main user interface (UI) for the radiation detectors, which connect to the phone via Bluetooth.

EIC has found considerable interest in being able to connect other sensors to smartphones in a similar manner. EIC has decided to develop an enabling technology, which can significantly reduce the development time and effort required to connect sensors or any other electronics to a smartphone. The overriding design criterion for BotSpine is that it will be the absolute easiest approach to connect electronics to a smartphone in order to monitor or develop the electronic devices.

The company has developed a powerful BLE high voltage (HV) board for their previous radiation detection project. The board was designed for a device to connect to smartphones with BLE and to protect users from dangerous radiation. Now, a project is underway to further develop the BLE board to a more general control board module that is able to control the behaviors of the robotics over the applications on the smartphones for development purposes. This research is expected to make a contribution to this promising technology in robotics development.

1.3 Thesis Aims and Objectives

The objective of this thesis is to design and develop an easy, ready-to-use, universal interface between sensor electronics and smartphones. The design of BotSpine includes interface structure design, hardware design, functions and firmware construction, application implementation and performance testing.

The detailed objectives are:

- 1) To design the BotSpine structure, including the methods of connecting sensors and smart phones, and outlining the functions BotSpine aims to achieve;
- 2) To design a proper hardware circuit by selecting an appropriate microcontroller, wireless connection and circuit board design;
- 3) To build stable and functional firmware systems so that BotSpine fulfills all expectations when operating;
- 4) To build a proper application implementing BotSpine to validate BotSpine as an easy and ready-to-use interface connecting smart phones and sensor electronics;
- 5) To discuss achievements and limitations of BotSpine and develop a conclusion that outlines BotSpine's performance and areas of future improvement.

The following outline is based on the detailed objectives listed above. Chapter 2 will introduce background knowledge of BotSpine, including the relationship between IoT and BotSpine, the current development tool limitations, connection technique and embedded systems chosen for BotSpine. Chapter 3 will detail the BotSpine design and the process of how it is built. Chapter 4 will demonstrate an implementation of BotSpine, Radon Sniffer design. The last Chapter will draw conclusions on BotSpine performance and future improvements.

2 Background

2.1 The Internet of Things

The Internet has allowed the whole world to communicate efficiently and connect closely. The “Internet” we have known since the 1990s has been a computer-to-computer data sharing based network. It can also be recognized as human-to-human communication because computers are technically computing machines that are directly or indirectly operated by people. People share information, experiences, knowledge and technologies with each other all over the world by means of the Internet. As a matter of fact, technologies in various industrial areas have made an incredible development leap in recent decades.

However, this is not enough. The world is comprised of more than merely human existence; there are animals, plants or, to a larger extent, other lifeless things. People always think big and they believe the idea of the “Internet” could be extended to every object on earth. If animals, plants and even unliving things can “communicate” with humans, we are able to make better decisions, and move toward a world of optimal convenience and efficiency. This brings to mind the concept of the Internet of Things (IoT).

The term “the Internet of Things” was initially brought by Kevin Ashton in 1999 when he tried to promote RFID technology. There seems to be more than one definition of IoT and one of the more proper and accurate definitions is from McKinsey:

“Sensors and actuators embedded in physical objects are linked through wired and wireless networks, often using the same Internet Protocol (IP) that connects the Internet.” [13]

The Internet of Things is becoming one of the leading development trends in modern and future technology. The core concept of the Internet of Things is to have every object in the world connected as a network. M2M technology, embedded internet and Internet of Everything concept terms are different but similarly related to IoT and a brief history of IoT can be linked to Reference [14]. IoT can be conceptualized as human-to-thing or thing-to-thing communication, with the characteristic objective of making things “smart”. With the ever-developing concept of the IoT, it

is conceivable that the concepts of smart home and smart city will become a reality in the not-so-distant future.

IoT implementation follows three pivotal steps: intelligence, connectivity and interaction [15]. IoT refers to the capabilities of awareness of self-status and surrounding environmental changes [16]. Things can then be connected using wired or, more popularly, wireless technologies. By this time, interaction will be the last step to assure things are able to achieve context awareness and able to communicate with humans or other things.

For the first step, we need to make things intelligent. Intelligence means things have the ability to self-configure and notice their status; for instance, they must have the ability to know the past, current and following processes. They must be able to automatically complete their tasks alone and adjust status according to ambience. Sensors are usually used to transform physical environment information into data that can be analyzed and processed. In the IoT era, more sensors will be implemented to allow things to be able to “speak”. However, sensors can only generate information; furthermore, besides sensors, intelligent things need embedded micro-controllers or a hard-wired programmed finite state machine in order to achieve self-awareness.

For the next step, we must establish the bridge of connection in security. Security is priority addressed problem whenever a connection is about to established. Although both wired and wireless technologies are available, wireless technologies play a dominant role because there is no limitation of location and distance and no requirement for additional connection cables. Wi-Fi, ZigBee, RFID, Near Field Communication, Bluetooth etc. are popular wireless technologies for data transmission in the IoT. Just like security protection in Internet, IoT security is also primary consideration. Bluetooth 4.0 or above is the connection method for BotSpine, the reason of which will explained in the subsequent section.

When a thing has the capability to generate data information and to be connected, the interaction between the thing and human or thing and thing is the last but also the core process of IoT. Without this interaction, the thing is merely repeating set up programs. It keeps all information itself and never has an effect on the world. Interaction involves two parties; in this case, it involves human-thing or thing-thing. Context awareness is the foundation of intelligent interaction. Connected objects always communicate, and this communication predominantly involves context awareness.

Thing can comprehend messages and commands from human while thing sends out data that can be understood by human after processing.

As mentioned in Chapter 1.1, the generic interface connecting sensors and smartphones is not easy to implement. The interface built is usually specific to merely one specialised application and it normally requires substantial time and effort to develop. IoT requires standardization from industrial to personal applications, with support from standardized interfaces, segments and communication technologies [17] [18]. Therefore, BotSpine is expected to be easy to use and portable to all embedded hardware circuits and software operating systems. Sensors and machines become intelligent and connectable through BotSpine, and BotSpine also securely sets up the interaction channels between people and sensors.

BotSpine can be considered a generic mini IoT network module for a fast and easy-to-go IoT development tool. By wiring BotSpine hardware Input/Output to sensors or electronics, the thing becomes intelligent; with the microcontroller embedded, it can self-configure status and self-process data sending and receiving. The embedded BLE module provides a wireless connection, which is widely implemented on most smart phones with efficiency and extremely low power consumption. The built-in BASIC language interpreter makes the interaction possible. Not only is BotSpine able to repeat defined routines, but it also accepts newly programmed executions and responds properly with desired data.

2.2 Challenges and Trends

Things of IoT indicate everything that has no computer attributes, including but not limited to daily items, sensors or objectives [19]. In respect to the “things”, sensors and robotic electronics are priority in our case studies. Enabling technologies are the technologies that make IoT possible by combining the “things” into a computing network. From this viewpoint, we chose smartphones as our computing terminal because smartphones have the same capability to process and compute metadata that is not especially vast as computers, but have the additional advantage of optimal portability. However, current existing wireless technologies cannot support direct communications between smartphones and sensors [20]. Moreover, there also seems to be lack of efficient and flexible workout solutions to this gap interface.

Another challenge would be the lack of a complete universal interface solution for cross-platform or sensor components compatibility issues [21]. Smartphones run on diverse operation systems, such as IOs, Android or Windows. Each design only deploys to the special design sensor/actuator device with unique console application on smartphones [22]. When a new design starts or some sensors are to be replaced, it means that design patterns are likely not capable of reusing the existing design features, either hardware or software stacks [23], but will start again. This is a waste of time, labor and intelligence resources.

There are some options for development tools in the current market which make “things” interactive enough to interoperate with smartphones. The most common and easiest option is to connect with a computing interface, like microcomputers or microcontrollers. The first and most popular choice that comes to mind is Raspberry Pi. Furthermore, Arduino and Electric Imp are other possible options for project design.

Raspberry Pi is considered a credit-card sized computer that is mainly utilized for educational purposes [24]. Raspberry Pi allows students to understand how the computer works and to learn some programming skills, such as Scratch and Python. To set up and start with Raspberry Pi, people must obtain a Raspberry Pi, SD card, connection cables, power supply and all input and output devices, such as display, keyboard and mouse. Users must also select their operating systems. In addition, only Pi 3 has embedded Wi-Fi and Bluetooth models.

It is glaringly obvious that Raspberry Pi is not the most facile solution for designing projects. First of all, people must gather and set up the hardware system connection. They must also know or learn to write on Python and use Scratch for programming. Both hardware and software setup requires a substantial amount of time and effort. As compared to most other sensors, the size of the hardware remains too large for the purpose of miniaturisation, one of the key features in enabling technologies. In addition, a kit of Raspberry Pi costs over \$100 CAD, with the Raspberry Pi board alone costing more than \$40 CAD. In industry, cost is priority and using Raspberry Pi for mass manufacture would constitute overspending, rendering it less attractive to implement Raspberry Pi for this application.

Arduino is more than merely a functional microcontrollers-embedded board; it is a prototyping platform providing both hardware and software build tools for open source [25]. Similar to BotSpine, the purpose of Arduino is easy to use, even for beginners. The hardware board is multifunctional and capable of most compulsory computing and operating functions. The software IDE supports OSX, Windows and Linux and provides a clear and text editing style code (called “sketch”) writing environment. The language is merely based on C or C++ functions.

Arduino, on the other hand, may have more resemblances with BotSpine, but still does not fulfill the expectations. It requires computers to install its software tools and then to program, while BotSpine can have all programming conducted on smart phones through the console app. Arduino also requires that users have some C or C++ program skills. Most importantly, after writing codes, it must go through compiling and downloading or burning to hardware board. Contrastingly, BotSpine is expected to download all programs wirelessly and can execute this without compiling.

For the low power consumption usage purpose of the thesis, Arduino and Raspberry Pi are not chosen because of their higher power. For example, it is not possible to run Pi or Arduino boards with a coin cell battery for months.

Electric Imp is a business solution dedicated to helping companies and individuals build their products in the manner of IoT. If customers have products and services that they would like to connect to internet for features and business models, Electric Imp provides full support by building wireless hardware, functional features, secure Wi-Fi connection, software and powerful cloud computing systems. Imp has an integrated platform in every aspect of the systems, wireless models,

software operation systems, cloud services, etc. BotSpine, on the other hand, is looking for a simple connection tool in a way of DIY. It will not have to be perfect, but it can be ready for users to play.

There has been substantial research conducted and a proposed solution ascertained for a smart sensor- smartphone interface [6] [7] [8] [11]. A universal sensor interface chip [26] and two-wire sensor interface [27] were also proposed to expand the flexibility of diverse sensor usage, which mainly focuses on sensor compatibility in IoT environments. However, no popular and well-designed universal interface emerges for rapid smartphone and sensor/robotics connection and control development.

The IoT developing technology trend is expected to spend increasingly less time and effort, and become more accessible for generic users. In terms of industrial usage, inexpensive cost is one of the significant factors. BotSpine is expected to fulfill these expectations. Firstly, BotSpine only requires the less expensive BotSpine board and its free smart phone console app. BotSpine is more economical for industrial IoT applications. BotSpine does not require a whole set of programming design process flow, but merely to type and run. BotSpine is considered a better, ready to use developing interface connecting sensors and smart phones.

2.3 Wireless Connection

2.3.1 Connection Choices

There are many wireless communication technologies available in terms of IOT applications, ranging from Personal Area Network (PAN), Local Area Network (LAN), and Neighborhood Area Network (NAN) to Wide Area Network (WAN) [28]. PAN is a network range within a few tens of meters, commonly used to connect to personal wireless accessories such as wearable sport monitors and wireless headsets. Low power consumption is an important characteristic of PAN devices. LAN, on the other hand, covers up to 100 meters range, which is mainly implemented at home or in buildings. Wi-Fi is a standard example of LAN applications. Considering the project range covering sensor network and smart phones, NAN and WAN are not involved in further discussion.

Bluetooth Low Energy, Wi-Fi and ZigBee are well known in the predominant wireless technologies of PANs and LANs. These three technologies will be introduced in detail and compared to ascertain the preferable wireless technology.

Wi-Fi may be one of the priority options because it has been widely embedded and used in smart phones, laptops, PCs and tablets [29]. Wi-Fi is defined as “wireless local area network (WLAN) products that are based on the Institute of Electrical and Electronics Engineering’ (IEEE) 802.11 standards” by the Wi-Fi Alliance, a global industrial association that owns the Wi-Fi registered trademark [30]. Wi-Fi operation frequency is ISM 2.4 band and in 5 GHz for more channels and its band rate can be up to 54 Mbps. Therefore, Wi-Fi is suitable for high speed large data transfer, which consumes significant power when operating. Moreover, Wi-Fi has relatively large and complex software that requires powerful processing. Laptops and smartphones have no impact because they have powerful microprocessors and notable memory units; comparatively, most sensor devices and smart “things” are not capable of significant and mass processing, rendering Wi-Fi not cost effective in industry.

ZigBee is another option for PAN application. Interestingly, the name of “ZigBee” comes from Waggle Dance of bees and it also reflects its mesh topology nature. ZigBee has standard IEEE 802.15.4 link layer defined under IEEE, with an operation frequency mainly in 2.4 GHz ISM band

and a data rate that is normally low. Long sleep intervals and low operation duty cycles allow cell batteries in ZigBee devices to be powered year long. ZigBee is widely used in sensor networks, home and industrial control and automation.

Bluetooth has become a standard default component in all mobile phones for years. Most personal electronics are Bluetooth connectable, such as Bluetooth headsets, music speakers, and health and fitness accessories. Bluetooth Special Interest Group (SIG) released Bluetooth version 4.0 and above, which renders Bluetooth technology a competitive contender in this new power-saving technology era. Bluetooth now has three categories, Bluetooth Smart, Bluetooth Smart Ready and Bluetooth Classic, respectively. Bluetooth Smart is also known as Bluetooth Low Energy (BLE) and only supports devices with Bluetooth Low Energy, rather than traditional Bluetooth. Compared to traditional Bluetooth, BLE, as the name implies, has much lower data throughput and thus requires less power. In terms of security, BLE has Man-in-the-Middle (MITM) function to secure all connection attempts. Although BLE is more for PAN implementation, it can be involved in Internet connection through smartphones' Ethernet protocol, which is safer and quicker path to Internet connection.

This research is industrial based and, therefore, the goal of wireless connection technology must be low power [31], cost effective, have a sufficient data rate for transferring small amounts of data, and most importantly, be supported by most smart phones. According to the above description, Wi-Fi fails in terms of high power consumption. ZigBee is not adopted in any smart phones, at least not in the near future. In conclusion, Bluetooth Low Energy with extreme low energy consumption is the best matched wireless connection for the thesis among the three selections.

2.3.2 Bluetooth Low Energy

Bluetooth technology was named after Harold Bluetooth, the Danish tenth century King who helped the union of many interwar countries [32]. As the name implies, Bluetooth was created in 1994 and alternates the traditional cables with wireless radio data exchanges. Bluetooth has been widely employed as one of the wireless standards connecting diverse products together.

Bluetooth technology is well known in wireless audio applications, Bluetooth headsets and Bluetooth music speakers. The version of Bluetooth for streaming data in high quality and efficiency is BR/EDR (basic rate/enhanced data rate).

Another important version of Bluetooth is BLE (Bluetooth Low Energy), which makes Bluetooth one of the enablers of IoT. Because of ultra-low power driving, products or sensors using Bluetooth can be much smaller and last for months with coin-cell batteries.

BR/EDR and BLE are the two core specifications of Bluetooth, with completely different architectures and application use cases [33]. Adopted as Bluetooth version 2.0/2.1, BR/EDR is suitable for audio streaming by providing continuous wireless connection in short range. BLE is Bluetooth version 4.0 and above and builds long-sleep interval short-burst connection for long battery cycles. There is a dual-mode chipset supporting single devices, such as smart phones or tables, to connect both BR/EDR and BLE enabled devices.

Although diverse Bluetooth applications have their own specific Bluetooth specifications, the basic elements of the Bluetooth core system architecture are consistent. As well defined in Bluetooth specification, the core system includes radio protocol (RF), link control protocol (LC), link manager protocol (LM) and logical link control and adaptation protocol (L2CAP).

In respect of Bluetooth stack in detail, Figure 2.1 illustrates all protocols and layers within a BLE stack. In addition to an applications layer (APP) on the top of each specific project, all layers are generally divided into two subsystems, Host and Controller. Controller includes the lowest layers, Physical Layer, Link Layer and Direct Test Mode, while Host involves other higher layers, L2CAP, Security Manager, Attribute Protocol, Generic Attribute Profile and Generic Access Profile. The interface between Host and Controller is referred to as the Host Controller Interface (HCI).

Bluetooth stack specification promotes system interoperability. Considering the fact that all layers are well defined and packed into stack, for each specific application or self-define project, the whole stack can be implemented as a sample project with only a few layer protocol modifications. Layers communicate with messages and these internal messages are accurately exchanged within equivalent layers or between subsystems of disparate Bluetooth systems.

Physical Layer Specification

Bluetooth physical layer must meet standard requirements defined in Bluetooth specification. For example, Bluetooth LE operates at an unlicensed 2.4 GHz ISM and 1Mb/s data rate [34]. The LE system operates a total of 40 channels, and these RF channels are defined as two types:

- Advertising Channels
- Data Channels

As illustrated in Figure 2.2 [33], channel 37 to 39 are used to advertise device discovery, connection establishment and data broadcasting. The remaining channels are data channels, which obviously exchange data with connected devices.

Link Layer Specification

Link layer is designed as a finite state machine (see Figure 2.3) [35]. It includes the following five states:

- Standby State
- Advertising State

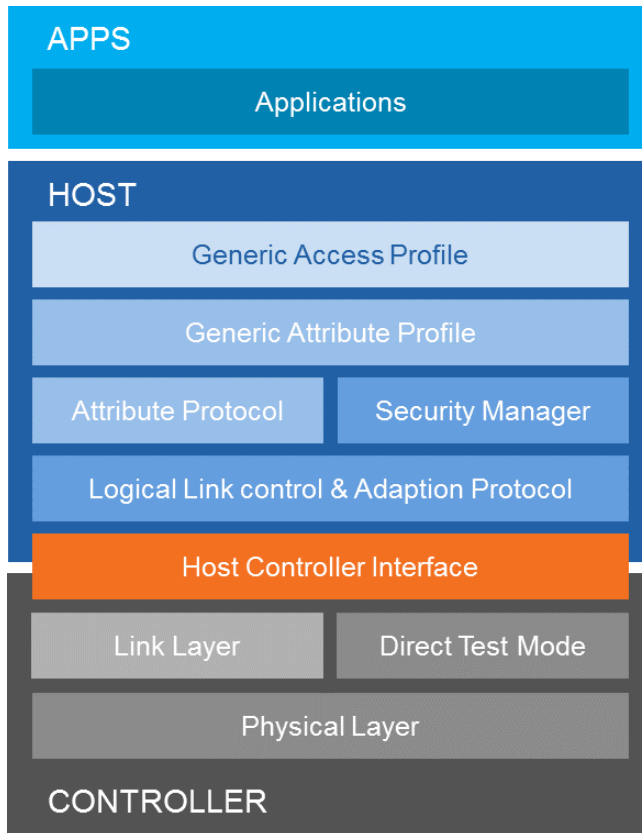
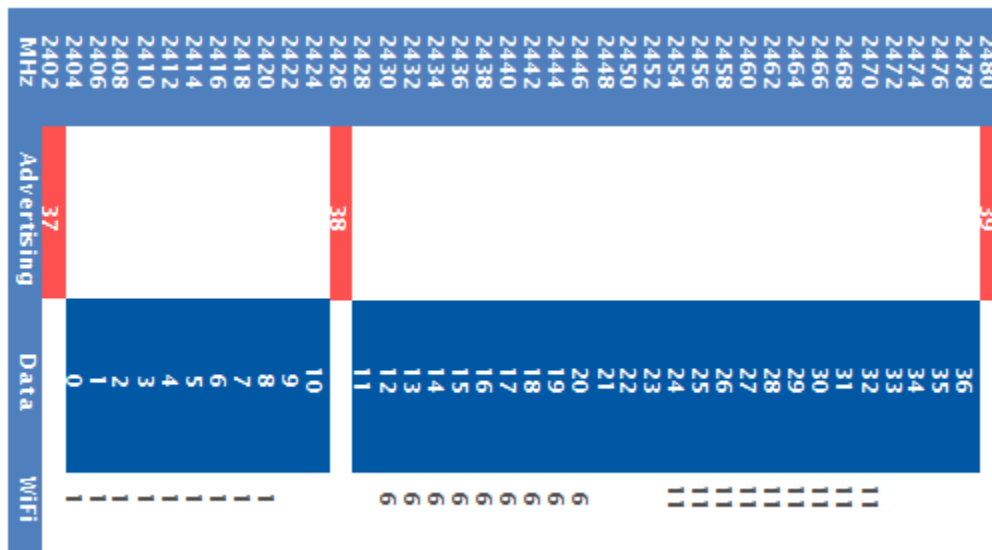


Figure 2.1 Bluetooth Stack



Lower guard band of 2MHz, upper guard band of 3.5MHz

Figure 2.2 Advertising and Data Channels

- Scanning State
- Initiating State
- Connection State

Standby State is a ready status and does no transmitting or receiving. Any state can enter into Standby State, but Standby State cannot directly enter into Connection State.

Advertising State is when the device advertises with advertising channel packets, and may also listen to and respond to other advertising packets in the range. That device is referred to as “advertiser” during advertisement.

Scanning State, on the other hand, is specifically for listening to advertising packets. A scanner is called when the device is scanning.

Initiating State refers to a state initiating a connection with the specific device after listening and responding to one and the other. An initiator is called during initiating.

Connection State denotes that the devices are in connection with one another. Connection State can only be entered from two states, Advertising State or Initiating State. Depending on the state entered from, the device plays two different roles:

- Master Role – If the connected device is from Initiating State, the device plays Master Role in Connection State. Master Role device communicates with a Slave Role device and determines transmission timings.
- Slave Role – If the connected device is from Advertising State, the device plays Slave Role in Connection State. Slave Role device communicates with a Master Role device.

BLE device uses the defined only packet format transmitting in both advertising and data channels. The bit order of the packet follows the Little-Endian format. The packet depicted in Table 2.1 includes four parts: Preamble, Access Address, Protocol Data Unit (PDU) and Cyclic Redundancy Error Check (CRC).

All packets in Link Layer have an 8-bit preamble in their LSB and the preamble allows the receiver to be able to synchronize frequency, estimate the symbol time and to train Automatic Gain Control

(AGC). Access Address distinguishes various connections between any two connected devices by different addresses. The following PDU is defined in either form, depending on either advertising or data channels. The final 24-bit CRC is generated by linear feedback shift register (LFSR) to calculate over PDU.

The Connection State is entered when either an initiator sends or an advertiser receives CONNECT_REQ PDU from another. At that point, a connection is considered to be created but not established. The difference between when the connection is created and when the connection is established is defined by the supervision timeout of the Link Layer connection.

Only two BLE addresses can be in one single connection and they play either a master or a slave. The master takes charge of the connection time and the synchronization point between master and slave is called a connection event. A connection request is supposed to be sent to or received from devices that are not already connected. Otherwise, the request should be ignored.

A connection event should include one or usually more than one packet that the master sends. The connection event stays when the packets are alternatively sent among both connected devices. When the connection is established, both the master and the slave shall define the identical data channel index used in all the packets during connection events. The timing of connection events has two important factors: connection intervals (*connInterval*) and slave latency (*connSlaveLatency*).

Figure 2.4 demonstrates the relations of parameters in connection events. Each block illustrates one packet sending either from the master to the slave or from the slave to the master. Each connection event begins at a point when the master starts sending PDU to the slave. These points of starts are called anchor points. Each connection event has a space known as a connection interval, starting from one anchor point to the next anchor point. The time inter frame space ($T_{IFS} = 150 \mu s$) is the minimum interval between consecutive

packets. Connection intervals should not overlap and the anchor point must have a space of at least T_IFS after the previous connection event. The *connInterval* shall not be less than 7.5ms and no greater than 4.0s, with a multiple of 1.25ms.

Slave Latency is the number of consecutive connection events the slave is allowed to ignore within the Supervision Timeout. The legal range of *connSlaveLatency* must be a positive integer and less than 500. The *connSlaveLatency* can also be counted by determining the number of anchor points that the slave does not listen when it receives packets from the master.

$$0 \leq \text{connSlaveLatency} \leq \left\{ \frac{\text{connSupervisionTimeout}}{\text{connInterval} * 2} - 1 \right\}$$

A connection loss can be the result of many factors, such as out of range distance, signal interference or power driving failures. To enable the detection of link loss for both the master and slave, Link Layer connection supervision timer ($T_{LLconnSupervision}$) is introduced. When a valid packet is received, $T_{LLconnSupervision}$ is reset. By the time $T_{LLconnSupervision}$ is equal to $6 * \text{connInterval}$ without any established connection, the connection is considered lost.

Upon the established connection, the supervision timeout (*connSpervisionTimeout*) is the time measured between two received Data Packet PDU. At any point of time that exceeds the *connSupervisionTimeout* value after an established connection, the connection is considered lost. The *connSupervisionTimeout* shall be no less than 100ms and no greater than 32.0s, with a multiple of 10ms in addition of

$$\text{connSupervisionTimeout} > (1 + \text{connSlaveLatency}) * \text{connInterval} * 2$$

If the connection is considered lost, the Link Layer enters the Standby State from the Connection State, while there is a notice of connection loss to the Host.

Direct Test Mode

Direct Test Mode is a layer that allows testers to directly send instruction commands to the PHY layer through the HCI or 2-wire UART.

Host Controller Interface

The interface between BLE Host and Controller provides various options of communication layers [35]:

- UART Transport Layer
- USB Transport Layer
- Secure Digital Transport Layer
- Three-wire UART Transport Layer

Logical Link Control and Adaption Protocol (L2CAP) Layer

L2CAP acts as the Data packets transmission center to the HCI, while it performs packet management and process protocol for higher level protocols. L2CAP manages Data packets segmentation and reassembly and multiplexing from higher layers to the HCI. Additionally, it imparts service information to the higher protocols [35].

Attribute Protocol (ATT)

ATT defines two roles, a server role and a client role, such that a server exposes a set of attributes to a client who gains access through the attribute protocol [35].

The general ATT architecture is depicted in Figure 2.5. A BLE device can perform as either a server or a client, or both roles concurrently. As exemplified in the example of Figure 2.5, Device A is a server to multiple clients, Device B, C and D. At the same time, A can also be a client of other devices. As a server, A shall have only a single set of attributes exposed to clients, which indicates that each attribute on the server is identical to all clients by its identical handle. Some attributes may be grouped as a service by higher layer specifications, but this still does not impact the uniqueness of attributes.

The Attribute Protocol has various methods efficiently commuting between server and client, such as requests, responses, notifications and indications.

An attribute normally consists of four elements as a discrete value:

- Attribute type – UUID
- Attribute handle
- Permissions
- Attribute value

The attribute type is used to make a client understand what the attribute represents when the attributes are exposed on a server. The attribute type is implemented by means of a universal unique identifier (UUID). Anyone can create UUIDs and UUIDs that are unique in any space or time.

Attribute handle is a 16-bit non-zero unique reference for a client to locate each attribute on a server. Attribute handles are assigned locally by the server.

Permissions specify how a client can access the attributes depending on the security levels. The permission is defined by a higher layer specification. A client can read and/or write the attribute values on a server, as well as notify and/or indicate, with specific permission. For the sake of attribute security, a specific secure link is required before accessing the attributes. There is an encryption link, authentication link and authorization link. Table 2.2 outlines the possible access permissions with different secure links.

Security Manager

Security is of great importance of Internet connection. Security Manager is installed in BLE stack to ensure BLE connection safety. Security Manager proves identity and executes encryption functions using key distribution. In this way, keys are generated separately by each device, which means the keys are secure under the control of their own device without interference with each other.

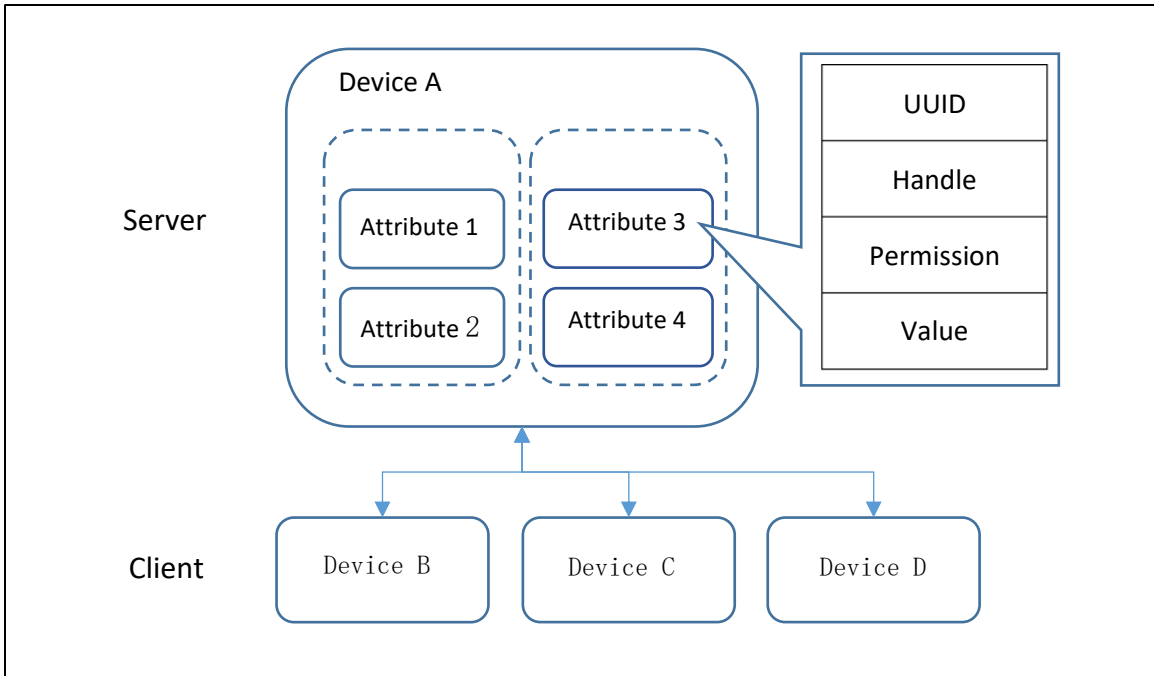


Figure 2.5 ATT architecture

Table 2.2 Possible permissions

Security		Access (may choose more than one)
No requirement	+	Readable
Encryption required		Writable
Authentication required		Notifiable
Authorization required		Indictable

In terms of security, the initiator has higher priority access memory and processing, as compared to the responder.

Pairing is the process of implementing security management, detailed in Figure 2.6 [35]. There are three phases with two optional choices for Phase 2:

- Phase 1: Pairing Feature Exchange
- Phase 2 (LE legacy pairing): Short Term Key (STK) Generation
- Phase 2 (LE Secure Connection): Long Terms Key (LTK) Generation
- Phase 3: Transport Specific Key Distribution

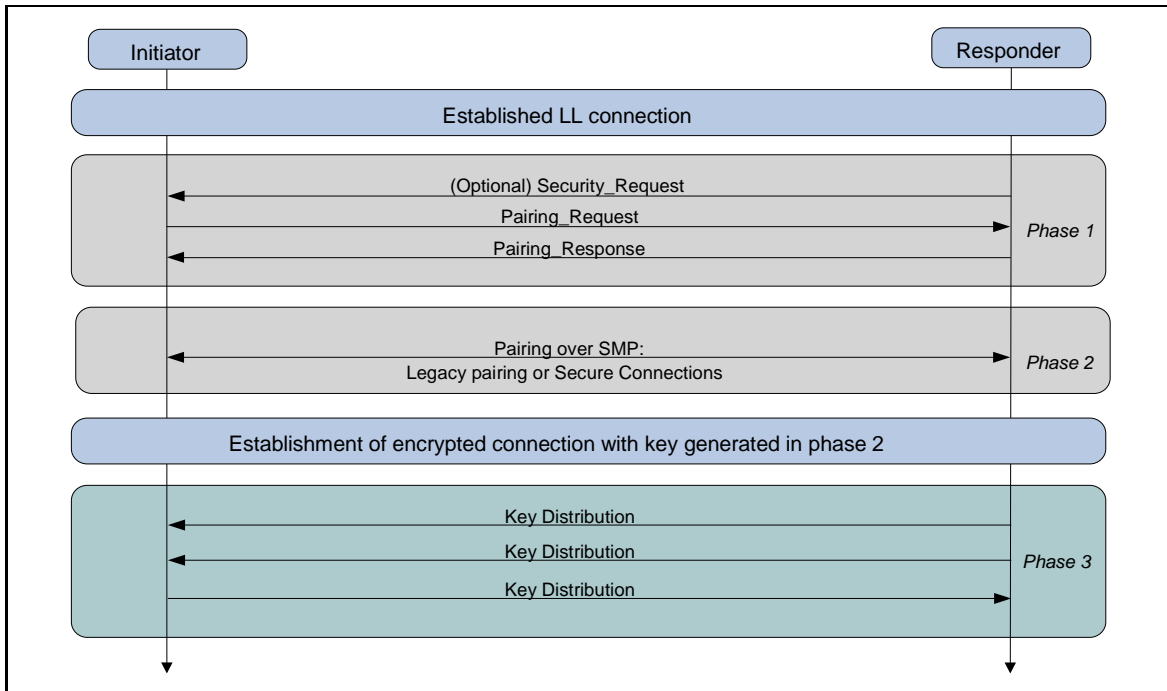


Figure 2.6 LE Pairing Phases

Firstly, after the Link Layer connection is established, the devices exchange their pairing features, including:

- IO capabilities for both devices
- Data availability for Out of Band (OOB) authentication
- Authentication requirement
- Key size requirement
- Selection for transport specific keys

The first three features are the decisive factors of Phase 2 optional approaches. In order for the initiator to send out the pairing, the responder must provide its identity. According to the IO functionality of both devices, one of the possible pairing methods illustrated in Table 2.3 is going to implement as well as determine whether LE legacy pairing or LE secure connection shall follow.

There are two keys generated in LE legacy pairing:

- Temporary Key (TK) – used to generate STK during pairing
- Short Term Key (STK) – used to encrypt a connection after pairing

The LE Secure Connection generates only LTK for connection encryption after pairing and consecutive connections.

Authentication requirements involve the enablers of bonding and/or man-in-the-middle protection (MITM) and they shall be set in GAP. Authenticated MITM can be implemented by the passkey entry pairing method or OOB pairing method in LE legacy pairing, while in the LE Secure Connection, the numeric comparison is also one of the possible methods.

Generic Attribute Profile (GATT)

The GATT profile is an implementation of the Attribute Protocol in respect to application view. In GATT layer, the client and server devices exchange information in a framework of services and characteristics. The hierarchy of GATT is depicted in Figure 2.7 [35].

A profile refers to the GATT profile, which contains multiple services to perform a complete use case. Each service includes several characteristics. The characteristic is a structure consisting of many essential elements describing the characteristic, such as characteristic properties, values, user descriptions and configurations. All services and characteristics are formed by attributes defined and stored in the Attribute Protocol.

Table 2.3 LE Pairing Method

Responder	Initiator					
	I/O Capabilities	DisplayOnly	Display YesNo	Keyboard	NoInput NoOutput	Keyboard Display
Display Only		Just Works	Just Works	Passkey Display (responder displays, initiator inputs)	Just Works	Passkey Display (responder displays, initiator inputs)
		Unauthenticated	Unauthenticated	Authenticated	Unauthenticated	Authenticated
Display YesNo		Just Works	Just Works [For LE Legacy Pairing]	Passkey Display (responder displays, initiator inputs)	Just Works	Passkey Display (responder displays, initiator inputs) [For LE Legacy Pairing]
			Unauthenticated			Authenticated
		Unauthenticated	Authenticated	Authenticated	Unauthenticated	Authenticated
Keyboard Only		Passkey Display (initiator displays, responder inputs)	Passkey Display (initiator displays, responder inputs)	Passkey Display (initiator displays, responder inputs)	Just Works	Passkey Display (initiator displays, responder inputs)
		Authenticated	Authenticated	Authenticated	Unauthenticated	Authenticated
NoInput No Output		Just Works	Just Works	Just Works	Just Works	Just Works
		Unauthenticated	Unauthenticated	Unauthenticated	Unauthenticated	Unauthenticated
Keyboard Display		Passkey Display (initiator displays, responder inputs)	Passkey Display [For LE Legacy Pairing] (initiator displays, responder inputs)	Passkey Display (responder displays, initiator inputs)	Just Works	Passkey Display [For LE Legacy Pairing] (initiator displays, responder inputs)
			Authenticated			Authenticated
		Authenticated	Authenticated	Authenticated	Unauthenticated	Authenticated

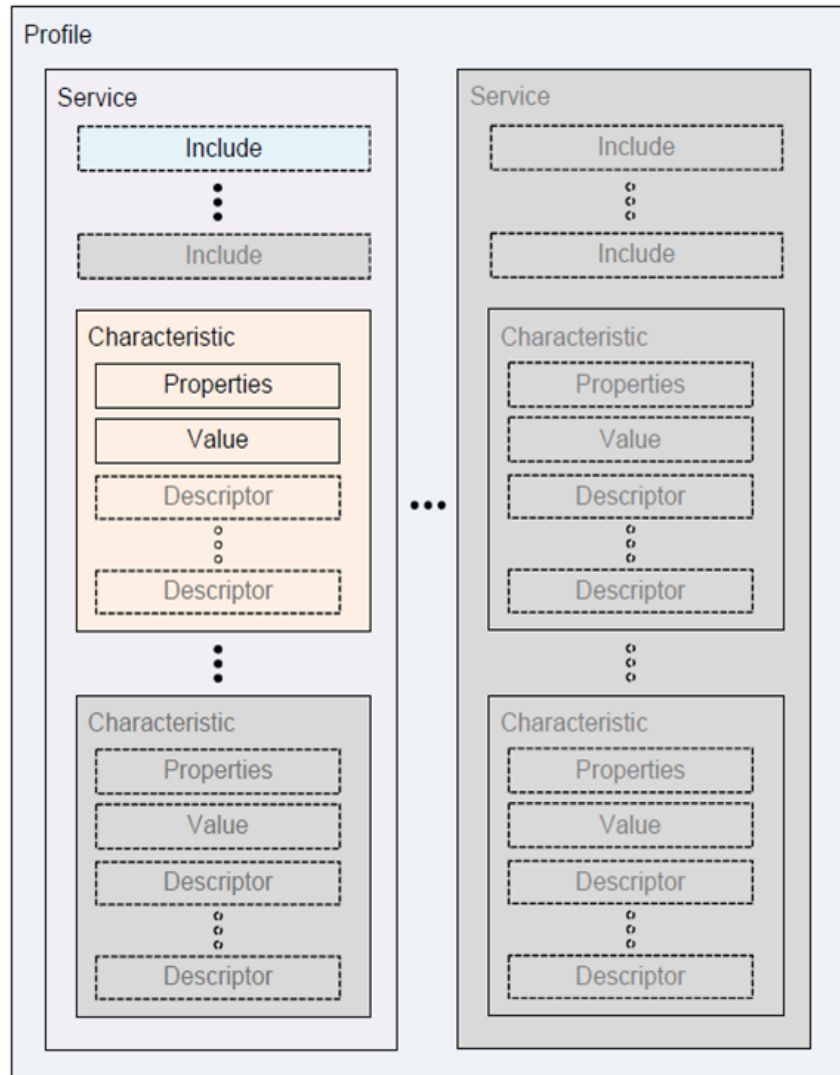


Figure 2.7 GATT Profile hierarchy

A server and a client exchange data based on characteristic level, including cases of:

- Configuration exchange
- Services and characteristics discovery
- Characteristic value reading
- Characteristic value writing
- Characteristic value notification
- Characteristic value Indication

GATT is a brand-new framework from Bluetooth version 4.0, which improves the flexibility of application development for users. More importantly, GATT framework allows Bluetooth devices

to connect in any scenario. Consequently, Bluetooth achieves low energy performance by connecting devices efficiently and connecting to smart phone applications directly.

Generic Access Profile (GAP)

GAP is used [35]

- To define procedures and operations related to modes and profiles.
- To describe device behaviours in states of connection loss, connection establishment and secure conditional connection.
- To simplify and establish a user interface that is user-friendly using coding schemes and naming procedures and parameters.

GAP is defined as four role performances related to the Physical Layer and Link Layer of BLE devices when operating physical transport:

- Broadcaster – send advertising events
- Observer – receive advertising events
- Peripheral – both transmitter and receiver in the Slave role in Connection State of Link Layer
- Central - both transmitter and receiver in the Master role in Connection State of Link Layer

2.4 Wireless Microcontrollers

Easy wireless connectivity with sensor network is indispensable to the industry. In recent years, the introduction of a new end-to-end building blocks model has vastly simplified industrial access to IoT [36]. As illustrated in Figure 2.8, in respect to integration development for cloud connectivity, the newest model includes low-power sensors, wireless microcontroller units (MCU), gateways and servers. By integrating a radio frequency (RF) transceiver, RF stack and MCU into wireless MCUs, industrial system designers do not have to spend significant time and effort on RF design techniques like an expert.

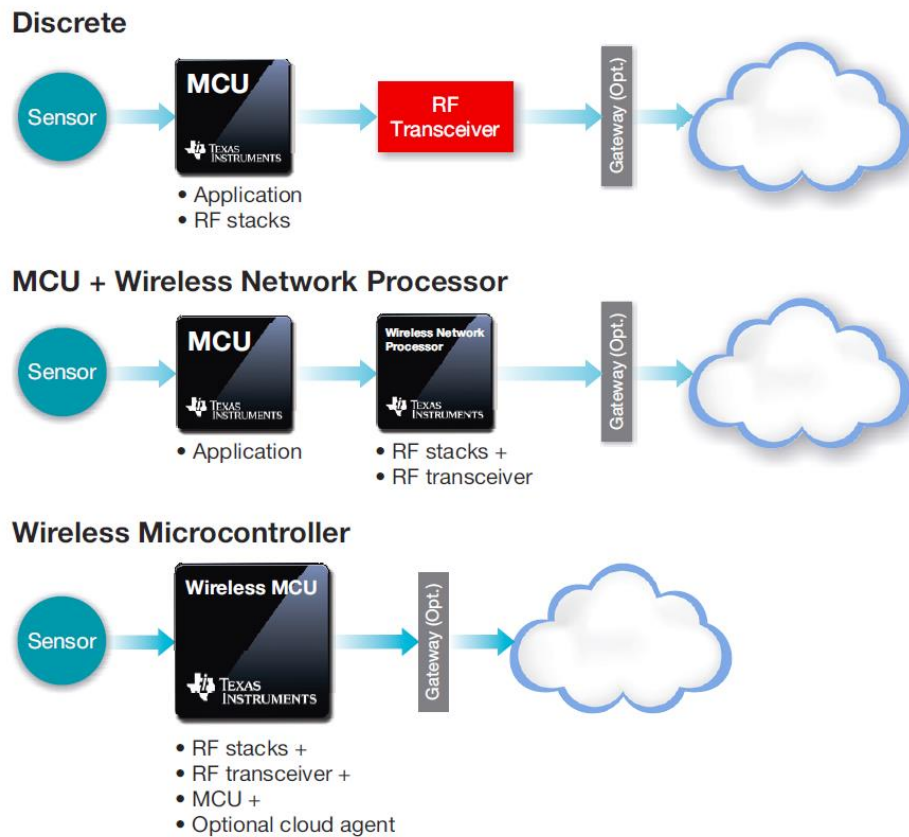


Figure 2.8 Integration variations for cloud connectivity

2.4.1 Texas Instrument Wireless Connectivity

In terms of wireless MCUs, Texas Instruments (TI), one of the world's leading semiconductor companies, was one the first marketing Bluetooth Low Energy with CC254x series and has a series of wireless connectivity on Bluetooth and/or Bluetooth Low Energy. TI has a variety of BLE MCU available. When the IoT starts to appear in the market, SensorTag and KeyFob from TI are one of the great options to have a trial of simple link with common sensors or objects.

Although up to today, TI has more updated MCUs such as CC2640R2F or CC2640 wireless MCUs, the MCU chose for the thesis was in 2014, when CC24xx series are one of the main MCU trends of TI.

The following table (Table 2.4) listed some Bluetooth Low Energy wireless MCU [37]. Although it seems not the most powerful MCU option, CC2541 could still be an fine choice for this thesis. In general performance, all series are more or less acceptable, but CC2541 has the maximum data rate, lower standby current and lower maximum TX power.

The CC2541 is a Bluetooth Low Energy System-on-Chip (SoC) solution released by Texas Instruments that operates in ultra-low power consumption. The CC2541 is composed of an enhanced single-cycle 8051-compatible MCU, a Bluetooth Low Energy-compliant RF transceiver with proprietary mode, 8-KB SRAM, 128/256 KB programmable flash and many powerful peripherals, including DMA controller, ADC, I2C, USART, timers and interrupt controller [38]. The functional block diagram of CC2541 is illustrated in Figure 2.9.

Table 2.4 TI Wireless MCUs Comparison

	CC2640	CC2541	CC2540	CC2650
Device Type	Wireless MCU	Wireless MCU	Wireless MCU	Wireless MCU
Bluetooth Standard	Bluetooth Smart (Bluetooth low energy)	Bluetooth Smart (Bluetooth low energy)	Bluetooth Smart (Bluetooth low energy)	Bluetooth Smart (Bluetooth low energy)
Flash size (KB)	128	128 256	128 256	128
RAM size (KB)	20	8	8	
Data Rate (Max) (kbps)	1000	2000	1000	1000
Operating Voltage (Min) (V)	1.7	2	2	
Operating Voltage (Max) (V)	3.8	3.6	3.6	
RX Current (Lowest) (mA)	5.9	17.9	19.6	5.9
Standby Current (uA)	1	0.5	0.4	1
Wakeup Time (PD→RX/TX) (us)		500	530	
Sensitivity (Best) (dBm)	-97	-93	-93	
TX Power (Max) (dBm)	5	0	4	
Package Group	VQFN	VQFN	VQFN	VQFN
Estimated Package Size (WxL) (mm²)	32VQFN: 16 mm2: 4 x 4 32VQFN: 25 mm2: 5 x 5 48VQFN: 49 mm2: 7 x 7	40VQFN: 36 mm2: 6 x 6	40VQFN: 36 mm2: 6 x 6	32VQFN: 16 mm2: 4 x 4 32VQFN: 25 mm2: 5 x 5 48VQFN: 49 mm2: 7 x 7

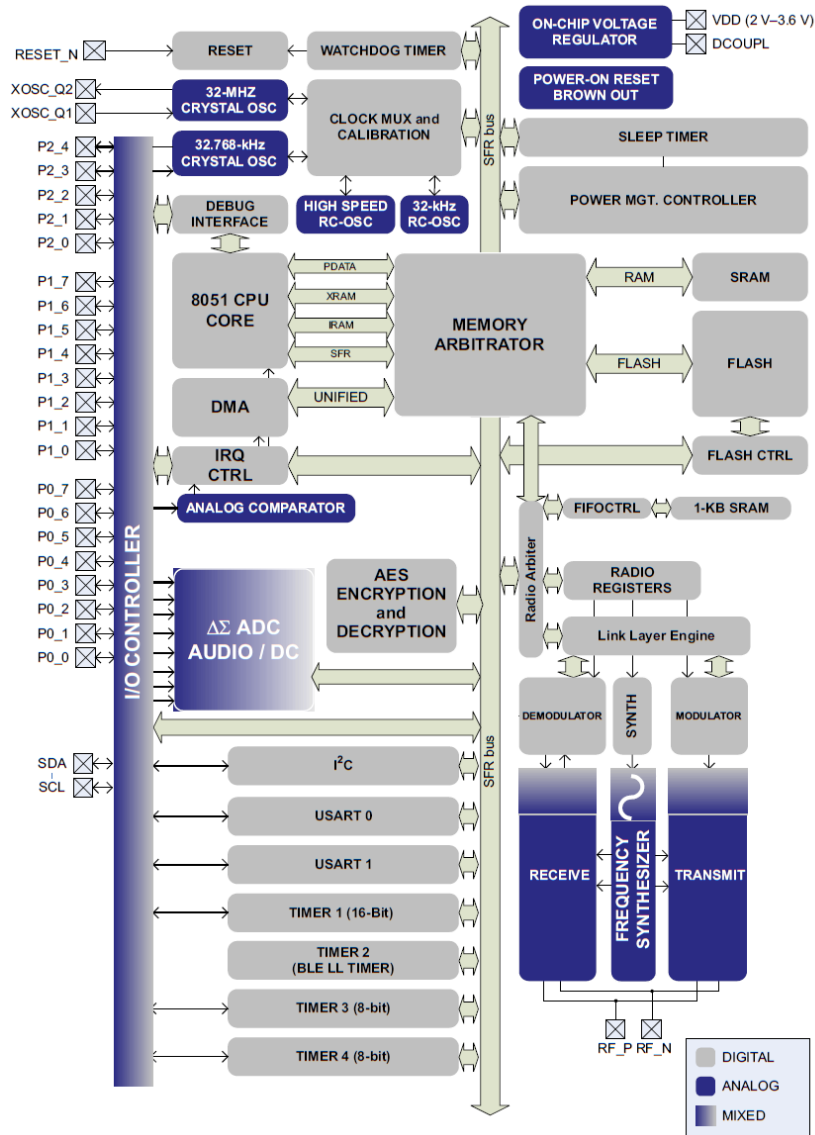


Figure 2.9 CC2541 Functional Diagram [38]

3 BotSpine Hardware and Software Development

3.1 Design Concept

Considering the current lack of an easy and rapid interface between smart phones and sensors, BotSpine is developed as a generic BLE development framework that rapidly and easily connects smart phones to the sensor network or electronics for ready-to-use development.

In hardware, BotSpine has an embedded microchip with a BLE module installed. By hard wiring sensors or electronics with BotSpine, BotSpine can acquire or generate digital data from physical environments or analog signals. BotSpine is able to connect to smart phones with BLE, and its phone console app allows people to obtain data transmitted from BotSpine or control BotSpine and its wire connected electronics by sending commands or pre-loaded programs via BLE. With access through the console application UI, people have the ability to analyze the data from BotSpine using more advanced microprocessors in smart phones, which will be faster and more efficient. Moreover, with a well-designed BASIC language interpreter built on firmware, BotSpine can “understand” and then execute commands or programs people send through on the console phone app. Figure 3.1 is the BotSpine logo and it vividly illustrates how BotSpine works.

There may be some development platforms available interfacing between tablets or computers and sensors or hardware circuits. However, they are either not interfacing with smart phones or too complicated to develop. As mentioned in Chapter 2, the sensor network is basically ZigBee enabled while smart phones do not embed with ZigBee, but rather with Wi-Fi and Bluetooth; this indicates the lack of a ready-to-use interface between smart phones and sensors in the current market.

From another perspective, BotSpine provides an easier and more time/money efficient development approach than the conventional flow. Due to the reality of the IoT worldwide

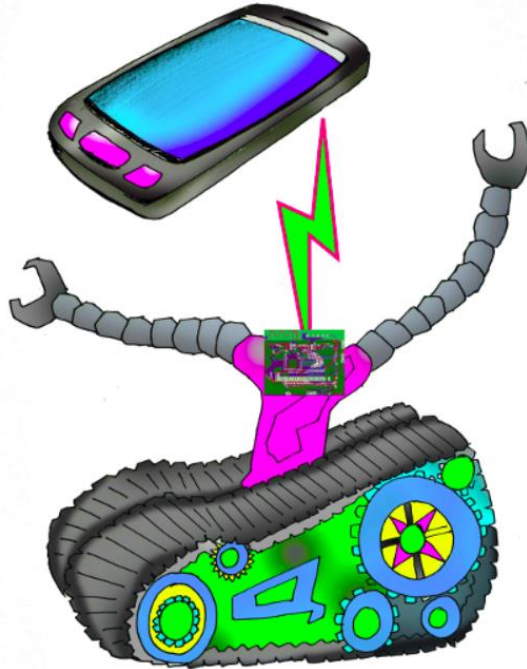


Figure 3.1 BotSpine Logo

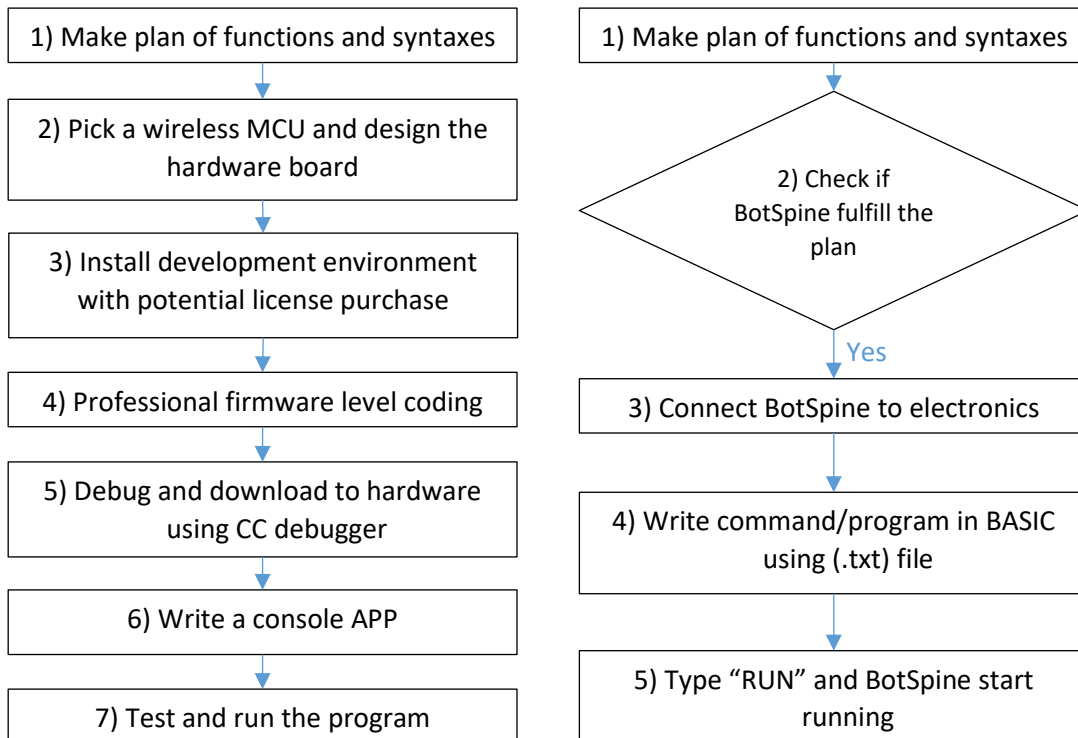


Figure 3.2 Development Flow Comparison of BotSpine (right) and General (left)

innovative application potential, the design patterns can be diverse and complicated [39]. Figure 3.2 compares a general project design flow with and without BotSpine. Evidently, the conventional routine requires more steps, as compared to using BotSpine.

It conventionally starts with making a project plan and ascertaining the expected end result. The next step is to select an appropriate processor and design the hardware, followed by firmware design. After installing the required integrated development environment (IDE), the firmware coding may be so professional that an electrical engineer may be required. Additionally, a software licence to run the IDE, such as an IAR license, may be quite expensive. Next, we must compile and download the firmware to hardware using a CC debugger. To run and control the program, an application on either tablet, computer or smart phone is necessary for data collection and analysis.

With BotSpine, the design flow and time can be vastly reduced. Check and see if BotSpine can satisfy the plan. If the answer is yes, then connect the BotSpine circuit board with the sensors or electric circuit and connect the smart phone to the BotSpine with the console application via BLE. Then type in execution commands or a stand-alone program for a project. This can be done directly on the console app; meanwhile, the recommendation is to write the code on Notes or text file and load to BotSpine over the air. The executions start as soon as the button is hit or “RUN” is typed in.

BotSpine is designed to be a generic development interface connecting smart phones and sensors or electronics, in accordance with the trend of IoT. BotSpine exposes its external IO ports for easier wiring and BLE is embedded to communicate with smart phones or BLE enabled devices. The development environment is the BotSpine console APP on Android system and the android app is text based so that commands and programs are sending to BotSpine over the air, similar to sending messages. In respect to power saving, BotSpine is powered by 3V, two AA batteries that are expected to run for one or two years before changing.

The following sections introduce additional design aspects of BotSpine in detail, respectively in hardware (HW) design, firmware (FW) design and software (SW) design.

3.2 Hardware Design

3.2.1 CC2541

As introduced in section 2.4.1, a wireless MCU simplifies the industrial development procedure, which is beneficial to IoT development. CC2541 from TI is the appropriate wireless MCU for this thesis. CC2541 is comprised of a 8051 CPU core, interrupt controller, up to 8KB SRAM, up to 256KB flash and many other functional peripherals, such as timers, ADC, USART, I2C, and I/O controller. [40]

The 8051 CPU in CC2541 is the enhanced core from the standard 8051 core. The difference is that the enhanced version uses object code compiled with the industry-standard 8051 assembler but has identical functionalities. The enhancements are:

- Speed improvement – one clock per instruction cycle is 12 times faster than the standard 8051
- A second data pointer
- Extended interrupt unit to support up to 18 sources

In addition, the CPU performance can be improved by 33% if flash prefetching is enabled. Although it may consume slightly more power, its faster performance improves energy consumption in general.

The memory of 8051 CPU uses Harvard architecture, which has separate data and program memory. There are four memory spaces:

CODE – program memory spaces 64KB for read-only.

DATA – data memory spaces 256 bytes for read-or-write. It can be accessed within one cycle CPU instruction. The address location can be direct or indirect for the lower 128 bytes, but only direct for the upper half.

XDATA – another data memory space that requires access time for 4-5 CPU instruction because it shares common bus with the CODE memory. There are 64KB spaces.

SFR – register memory space for read-or-write access within one CPU instruction cycle. There are 128 bytes of space and each address bit is addressable.

Although these four memory spaces are separated in zones in 8051 architecture, there are some overlaps mapping into each other's space. The DMA controller necessitates access to all physical memory. Therefore, a portion of SFR and DATA memory spaces are mapped into XDATA memory space, as depicted in Figure 3.3.

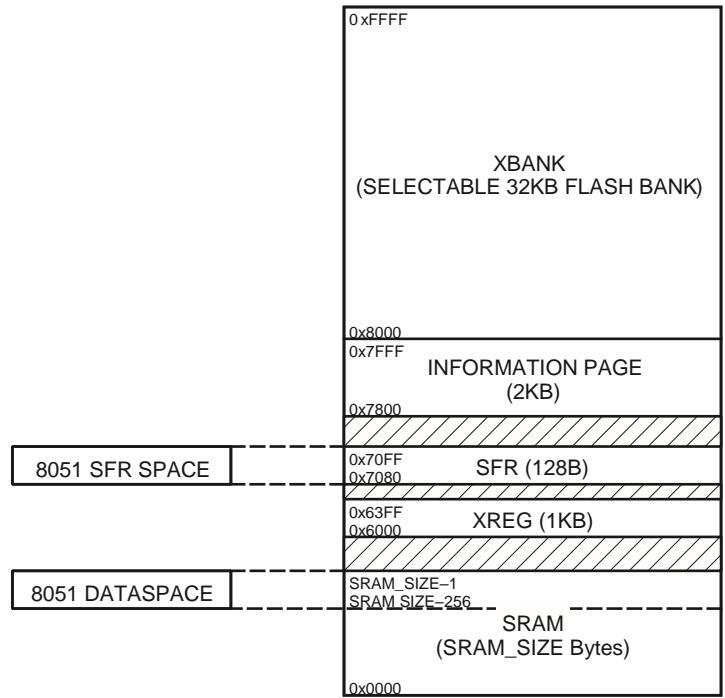
3.2.2 External I/O

In terms of I/O ports, BotSpine provides 20 external I/O pins that are available for wire connection. These I/O pins can be configured for either general purposes or peripheral purposes. Before using these pins, they need to be configured with their purposes, directions and many specific enablers. For general purposes, pins can be set as digital pulling high/low, and also as interrupt enables and the DMA triggers. In regards to peripheral functionality, Table 3.1 lists all peripheral functions and their respective pin names.

3.2.3 PCB Antenna

The PCB antenna introduced can be implemented to all 2.4GHz designs, such as CC25xx and CC24xx [41]. The PCB antenna is also known as Inverted F Antenna (IFA), which was specially designed as an alternate of 50 Ohm impedance at 2.45 GHz. The supportive distance range of the PCB antenna is about 30 meters.

The PCB antenna design has extremely accurate dimension requirements, such that the reference design must be exactly replicated in the aspects of shape and dimensions. It is noted that any small changes of the reference design could result in performance failure. Furthermore, the antenna design follows exactly as demonstrated in Figure 3.4 and Table 3.2.



M0097-02

Figure 3.3 XDATA Memory Space

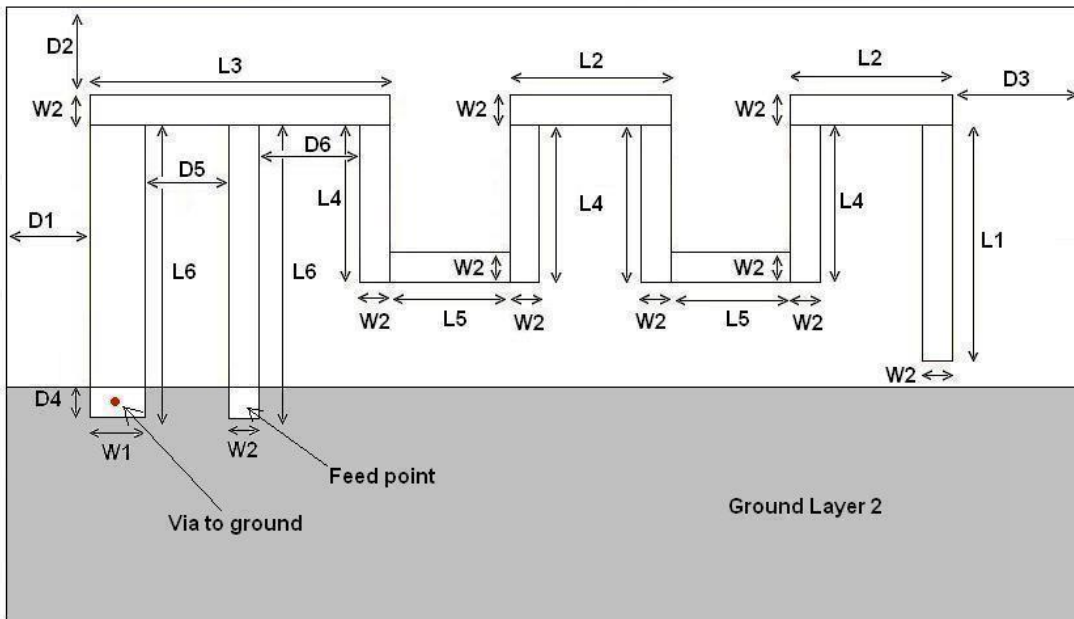


Figure 3.4 Antenna Dimensions

Table 3.1 *BotSpine I/O Mapping*

BASIC	SPI0	SPI1	SPI2	SPI3	SERIAL	CC254X	ADC	COMMENT	OLED
P0(0)						P0_0	ADC0		
P0(1)						P0_1	ADC1		
P0(2)	MISO					P0_2	ADC2		D/C
P0(3)	MOSI		CLK			P0_3	ADC3		CLK
P0(4)			MOSI			P0_4	ADC4		Data
P0(5)	CLK		MISO			P0_5	ADC5		Reset
P0(6)						P0_6	ADC6		
P0(7)						P0_7	ADC7		
P1(0)						P1_0		20 mA NO PULLUP/DOWN	
P1(1)						P1_1		20 mA NO PULLUP/DOWN	
P1(2)					CTS	P1_2			
P1(3)		CLK			RTS	P1_3		Can't be used as output	
P1(4)		MISO			RX	P1_4			
P1(5)		MOSI		CLK	TX	P1_5			
P1(6)				MOSI		P1_6			
P1(7)				MISO		P1_7		Top pad on side connector	
P2(0)						P2_0			
P2(1)						P2_1		On program connector	
P2(2)						P2_2		On program connector	

Table 3.2 *Antenna Dimension Values*

L1	3.94 mm
L2	2.70 mm
L3	5.00 mm
L4	2.64 mm
L5	2.00 mm
L6	4.90 mm
W1	0.90 mm
W2	0.50 mm
D1	0.50 mm
D2	0.30 mm
D3	0.30 mm
D4	0.50 mm
D5	1.40mm
D6	1.70 mm

3.2.4 Hardware Board Layout

The BotSpine hardware board includes a wireless MCU - CC2541 from Texas Instruments. There is a total of 20 external I/O pins available to wire with external electronics, such as sensors and robotics. The external connector uses a 20-pin female header. The PCB antenna follows the design of IFA for small size 2.4 GHz. The PCB print out is illustrated in Figure 3.5.

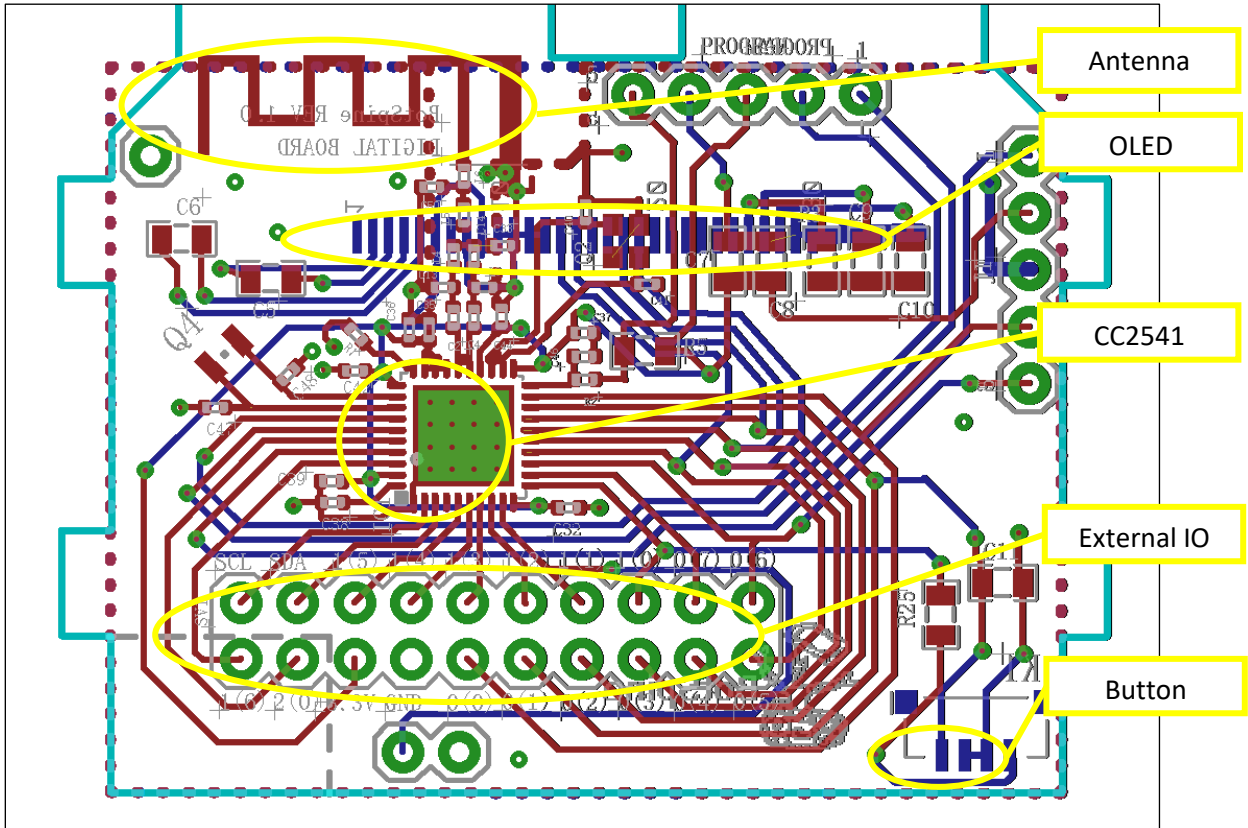


Figure 3.5 BotSpine PCB

Moreover, an OLED display and button are added to the circuit board so that users can add the hardware directly to the printed-out board without building additional circuits. On the other hand, the OLED display functionality is also reserved in the firmware library in order to directly implement the displays on the OLED screen.

3.3 Firmware Design

Based on the CC2541 complete System-on-Chip solution, TI provides a complete BLE software development kit for customized application development [42], which refers to the firmware of the thesis. The development kit includes the following five elements:

- OSAL
- HAL
- The BLE Protocol Stack
- Profiles
- Application

The Operating System Abstraction Layer (OSAL)

OSAL is the fundamental execution layer of the software that supplies controls to the BLE protocol stack, profiles and applications. The OSAL is a control loop that executes defined tasks according to execution conditions and pre-set priorities. Each layer of software is considered a functional task. Tasks are defined with task identifiers (ID), task initialization routines and event processing routines. These tasks must be set up in a priority scheme, in which the Link layer has the highest priority as a result of timing requirements. The following (Figure 3.6) is the defined tasks array in the OSAL.c file.

After OSAL initialization of all layers, the system runs into a perpetual loop to check and execute task events, as depicted as Figure 3.7.

Events are regarded as processing routines under each task and they are defined as each single bit in a 16-bit variable, ensuring that there are no concurrent events being processed. If an event is set, OSAL will call the task layer to process the callback function for that event.

```

// The order in this table must be identical to the task initialization calls below in osalInitTask.
const pTaskEventHandlerFn tasksArr[] =
{
  LL_ProcessEvent,           // task 0
  Hal_ProcessEvent,        // task 1
  HCI_ProcessEvent,        // task 2
#if defined ( OSAL_CBTIMER_NUM_TASKS )
  OSAL_CBTIMER_PROCESS_EVENT( osal_CbTimerProcessEvent ), // task 3
#endif
  L2CAP_ProcessEvent,      // task 4
  GAP_ProcessEvent,        // task 5
  GATT_ProcessEvent,       // task 6
  SM_ProcessEvent,         // task 7
  GAPRole_ProcessEvent,    // task 8
  GAPBondMgr_ProcessEvent, // task 9
  GATTServApp_ProcessEvent, // task 10
  BotSpine_ProcessEvent    // task 11
};

```

Figure 3.6 OSAL Tasks Array

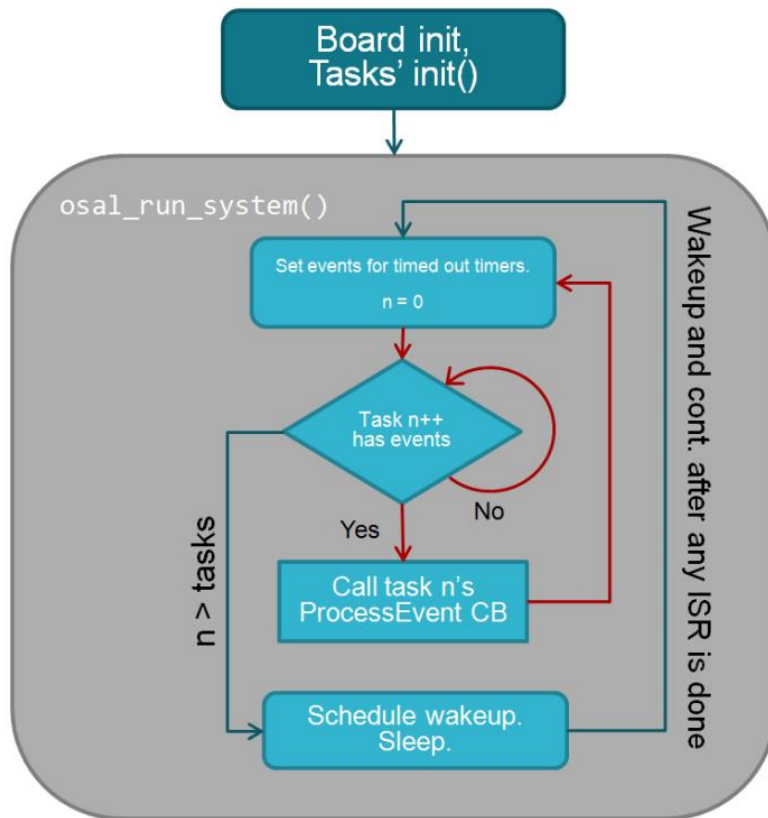


Figure 3.7 OSAL Task Loop

In addition, users can set up OSAL events in layers other than OSAL. One method is to use the `osal_set_event()` function, while the other is to use the `osal_start_timerEx()` function. The `osal_set_event()` function is able to immediately set up a new OSAL event, while the `osal_start_timerEx()` function has an additional timer function that sets the specific event when reaching time-out values.

Hardware Abstraction Layer (HAL)

The HAL is an abstracted interface that controls physical hardware behaviors by means of software coding. The most notable benefit of HAL is that only the HAL needs to modify to fit a new hardware design, while the applications and profile source codes remain exactly the same. Therefore, when there is a new hardware design, the HAL must make the changes for the consistency between the hardware and software.

The HAL supports a variety of hardware drivers:

- ADC
- AES – encryption
- LCD
- LED
- KEY
- DMA
- UART and SPI
- Simple NV (SNV)

The Applications and Profiles are source codes describing the desired behaviors for the specific application cases. They can be applied as sample applications provided with the development kit or to be modified to the customized applications.

BotSpine is not only a BASIC language translator translating human readable language into executable machine codes, but also an electrical engineer managing all registers and enablers in hardware by interpreting a single command.

As it currently stands, BotSpine has two versions. The following sections will introduce the two versions of BotSpine in detail and the difference between them.

3.3.1 BotSpine Firmware Version 1.0

The first version of BotSpine is essentially a single command mediator. BotSpine has a service called “botService” in the profile, which entails command handling and execution. The profile also includes a profile of battery reading and a profile of device information. A whole profile is outlined in Figure 3.8.

BotSpine receives a 20-byte command as a characteristic from the console application, and BotSpine service interprets the received command messages by comparing word by word with the keyword library created in the profile. Each function has its key word to trigger, register and enable the corresponding function.

The command instruction is straightforward. It begins with reserved letters or signs, followed by the port and pin number it is expected to work on. The functions and their syntax designed on BotSpine are listed on Table 3.3.

BotSpine is designed to make the control uncomplicated for everyone, even customers with no program experience. The users are only required to write commands in order to control the BotSpine. Rather than delving into technical programming aspects, which are unfamiliar to most people, BotSpine is designed to write down the command in the following general format:

[Reserved Sign/Letters] + [Port#] + [Pin#] + [State/Value]

For example, if they want to pull P1_5 high, they just need to type “d151”. Then, the FW will set P1_5 high. There is no space between keywords. The maximum size of syntax that can be transmitted is 20 bytes. Hence, if the total string exceeds 20 bytes, the device will run into invalid mode and then disconnect from the app.

Furthermore, users can only send one 20-word “text” at a time, which means one action command is sent at a time. Each text will be sent out and transmitted every time users tap the “Go” button, just like people send texts.

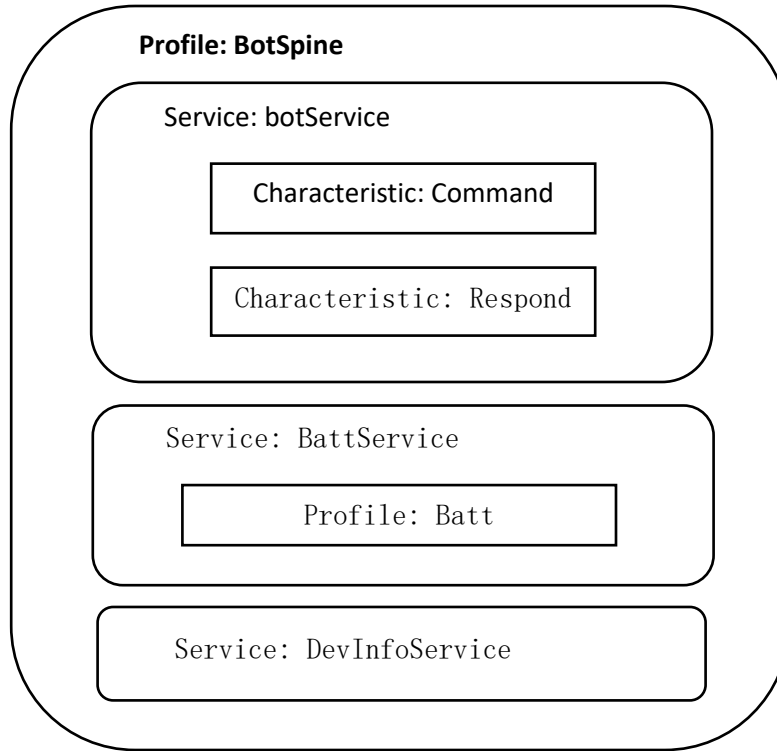


Figure 3.8 BotSpine Custom Profile

Table 3.3 BotSpine Version 1.0 Functions

Function	Description	APP UI Syntax	Syntax Example	FW Action
Digital Write	Set one pin high/low; ALL pins are available for this function including SCL & SDA	<i>D/d</i> +Port+Pin+State or <i>D/d</i> +SD(or SC)+State	"d151"	Set P1_5 high
Digital Read	Read digital on pin; ALL pins are available for this function including SCL & SDA	<i>R/r</i> +Port+Pin or <i>Rr</i> +SD(or SC)	"r00"	Read P0_0 state
OLED Display	Display text on the OLED; P0_2-P0_5 are reserved for this function	? +output-string**	"?Hello World"	"Hello World" on OLED screen

Enable Password Mode	Secure device; set password before enabling password	E/e+1/0*	"e1"	Enable password mode
Enter Password	Enter password to get access into functions executable	P/p+password	"P1234"	Check if password is correct
Write Password	Overwrite password	P/p+ W/w+password	"Pw5678"	Reset password to "5678"
ADC	Analog-to-digital converter; only for Port-0; Vmax=1.25v	V/v+Pin+1/0*	"v61"	Read the analog input voltage at P0_6
PWM Enable	Enable/Disable PWM; output pulse at some pins (refer to Table 1 or 2 for PWM)***	M/m+Port+Pin+1/0*	"m201"	Enable to output pulse width modulation at P2_0
Enable Counting	Enable/disable Counts	C/c+Port+Pin+1/0*	"c061"	Enable counting
Show counts	Show count numbers	#+C/c	"#c"	Show count numbers
Change Setting parameters	Change the parameter setting: font size, row number, ADC reference voltage, etc. (See Table 4)	\$+Function+Keyword+V alues	"\$?f2"	Change the OLED display font size to 2

Note: *1-Enable; 0-Disable.

Reserved command: "?logo**" - Display "BotSpine" splash screen.

***PWM could work at all its alternative location at the same time; but if one is disabled, some others may be affected and need to re-enable to make it work again.

How does the FW work? Syntax written on the UI will be stored in the flash and then transmitted to FW as strings; UI syntax is user friendly and expected to be short in certain formats, allowing sufficient space to store and ensure fast transmission. In terms of the FW, all words in the text will be read and recognized as keywords, and the defined corresponding functions in FW will be executed. The firmware functions all have default settings. However, users can still change those settings by sending commands that start with “\$”. For parameters changing, all changeable parameters and their values are demonstrated in Table 3.4.

If an invalid command has been sent, the user will receive an error-happen response beginning with “F,” followed by a specific error code such as “F01”. Table 3.5 lists all error codes and their causes.

There are two operation modes: non-password mode and password-required mode. By default, the device is in non-password mode, which allows it to execute all commands received from the APP. However, if the device were used as personal device controller, such as a garage door opener, the user may prefer to operate under password-required operation mode.

Before going into password-required mode, users can create their password by sending a password writing command. Users can subsequently enable the password mode. Under the password mode, the device will execute commands only when the correct password is entered. If the entered password is wrong, the connection will be terminated and the device will start advertising again.

The device remembers the operation mode even if the connection is lost, signifying that the password is required after disconnection under password-required mode.

Users can rewrite the password under non-password mode or after entering the correct password under password-required mode.

Table 3.4 BotSpine Version 1.0 Function Parameters

Parameters	Function Keyword	Keyword	Values
Font size on OLED	?	f	1 to 8 (default as 1)
Row number on OLED	?	r	0 to 7 (default as 3)
ADC external reference*	V	x	None
ADC resolution	V	s	8,10,12 or 14 (bit)
PWM frequency	M	f	245 to 65535

*If use ADC external reference, the output digital value would be raw ADC reading value.

Table 3.5 BotSpine Version 1.0 Error Code

Error Code	Results & Causes
F00	No password has been entered under password mode
F01	Digital Write fails because of invalid port/pin number
F02	ADC fails because of invalid pin number
F03	PWM start fails because of low frequency
F04	Parameter changing fails because of invalid function-word/font size/row number/resolution
F05	Counting couldn't be enabled because of invalid pin number
F06	Digital Read fails because of invalid port/pin number
F07	PWM initialization fails because of invalid port/pin
F08	PWM stop fails because of invalid port/pin

3.3.2 BotSpine Firmware Version 2.0

BotSpine version 2.0 is upgraded based on the same concept as BotSpine version 1.0 by integrating with BlueBasic. Furthermore, the BotSpine version is also referred to as “BotBasic”.

After the first version, BotSpine was expected to become more powerful so that it could support the stand-alone execution. Furthermore, rather than merely executing one single command or function at a time, BotSpine shall be able to run a preloaded program as a finite state machine and load all of the collected data to smart phones at once when it is connected with the phones. Therefore, BotSpine does not have to stay connected to smart phones all of the time, which reduces power consumption and grants smart phones more freedom for other more complex algorithms and processing.

To achieve this goal, systematic microcontroller architecture is required to store the program codes in a pre-determined order and to accurately relocate and execute the stored programs. Meanwhile, BlueBasic has come into our sight.

BlueBasic is an open source embedded BASIC interpreter running on TI CC2541, and was created by Tim Wilkinson [43]. The motivation of the BlueBasic project is to allow users to develop firmware of CC2540 or CC2541 with a free firmware development platform. By default, IAR Systems is the development environment to develop, compile and debug these TI BLE microchips. However, customers commonly buy the CC254x series chips at a very low price of less than \$5, while the development of these chips requires an expensive IAR licence of about \$3,500. It is unfair for customers to spend such an amount of money on the complicated firmware coding and compiling. Therefore, Tim decided to build a heavily modified interpreter on CC2540/41 in BASIC language.

Interpreter structure firmware brings memory saving and compiling free advantages. Instead of compiler, installing interpreter structure in firmware takes super smaller compact within limited memory. Interpreter requires no compiling and building process which gives freedom of downloading codes during running. Among all popular program languages, such as C, Python, most of them needs to be compiled and compiler and compiling process take some certain amount of memory compact. BASIC, compared to those languages, is much smaller compact of language and

its simple syntax helps to keep the interpreter small. So an interpreter with BASIC language is a great combination of small and efficient compiling free firmware structure.

With the system architecture trade-off between the existing version of BotSpine and BlueBasic, a combination system emerges as BotSpine version 2.0, named “BotBasic”. BotBasic retains the single command functionality and all implemented functions of BotSpine and also adapts the modified flash storing lined programs from BlueBasic. In addition, some of the syntaxes are adapted to follow the BASIC language format in order to keep the consistency of the whole BotBasic system.

The BotBasic profile is depicted in Figure 3.9. As discernable in the figure, the consoleProfileService service contains two characteristics, input and output, which are functionally identical to the command and respond characteristics of version 1.0.

Moreover, BotBasic consists of the OAD service, which refers to Over the Air Download. OAD is one of the provided profiles by BLE stack from TI for CC254x. OAD provides the loading approach over the air rather than using a physical programming header. The mechanism of OAD entails sending the image from the server to the client.

With the integration with BlueBasic, BotBasic achieves the expectation of stand-alone mode running with the addition of BlueBasic existing functions; for example, BotBasic is able to create new BLE services and characteristics, as well as manage advertisement and connections by typing in BASIC.

In addition to the aforementioned achievements, BotSpine is also able to perform a larger range of functionalities:

- OLED library
- PWM function
- Fast counter
- Buzzer function
- Password for program protection

Although there may be more functions to add in the future for improvement, the current up-to-date version of BotSpine is functional satisfactory for many basic project developments.

BASIC is a simple program language with small compact and compatible with build-in interpreter structure in firmware. It also remains the possibilities to use other program language as the interpreted language such as Python. The interpreter structure is based on keyword matching logic. When the program line comes in, the firmware recognizes each word in the line and categorise its target functions and settings and setup accordingly in firmware code. Switching to another program language, the keywords and expected syntax logic will need some changes in firmware.

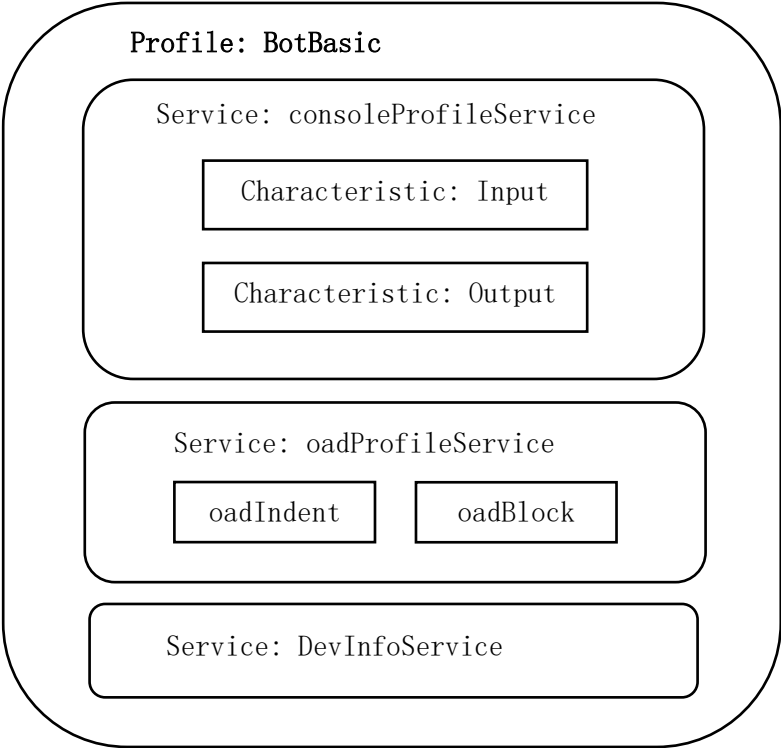


Figure 3.9 BotBasic Custom Profile

3.4 Software Design

The smart phone application is developed on the Android system because it is easier to develop than the IOS system.

The APP is drafted from the TI sensor tag application on Google Play. Both versions of BotSpine work well with the developed Android App.

The App begins with a scanning page and lists all discoverable BLE devices in the range, as conveyed in Figure 3.10. When the app has established a connection with one of the BotSpine devices, it will jump to the command page. If the connecting device is not BotSpine which does not contain BotService characteristics, the connection will be terminated.

The initial command page displays a list of information if a successful connection is made, including the application version and total available free flash space, followed by “OK”. Figure 3.11 illustrates the appearance of the command page. All responding information also appears as a message toast (see Figure 3.11 left).

There are a few buttons set up in the command page. The “Go” button is used to send out a characteristic *Input* when a command is ready to send over to the device. After each command is sent, a response will be displayed on the white space below, as well as a toast message form. If it is an unlined command, the respond will be “OK” or there will be error messages indicating the problems. Only after receiving the responds can the next command be sent out. On the other hand, if it is a lined program, no respond message will show up until the program starts to run by sending “RUN”.

The “Clear Console” button is used to delete all of the display responds. The “Log Output” button loads the collected data from the sensors to the smart phone as a .txt file. “Read BBB.txt” and “Read AAA.txt” are used to read and send the written lined BASIC program from smart phone to BotSpine board. The program code is sent line by line until it reads the end of the program. It is more efficient if the program is quite large, which takes time and effort to write and send. The program can be written in the text file form, which allows more flexibility for users to type programming codes with their computers or tablets.

After the program is loaded to BotSpine, “RUN” or “AUTORUN ON” will trigger the program execution.

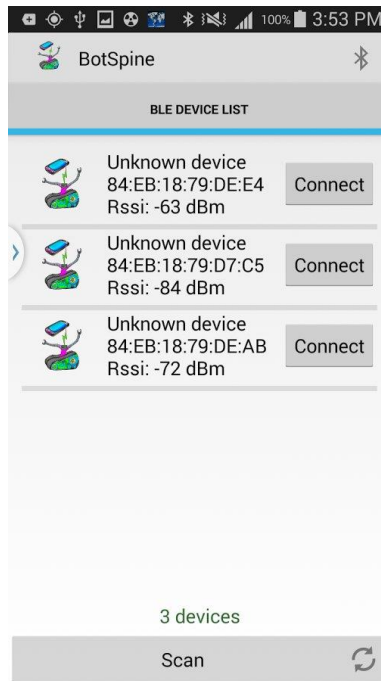


Figure 3.10 BotSpine App Scan Page

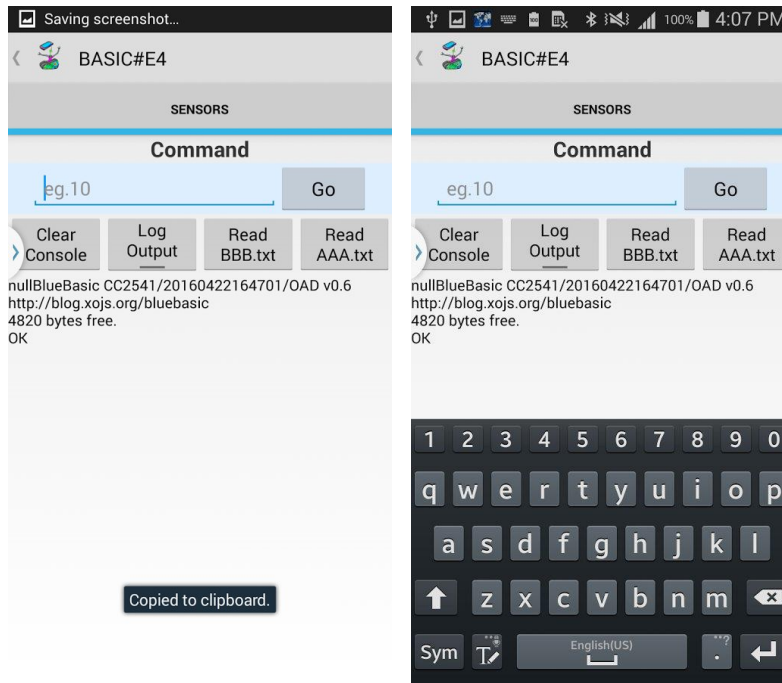
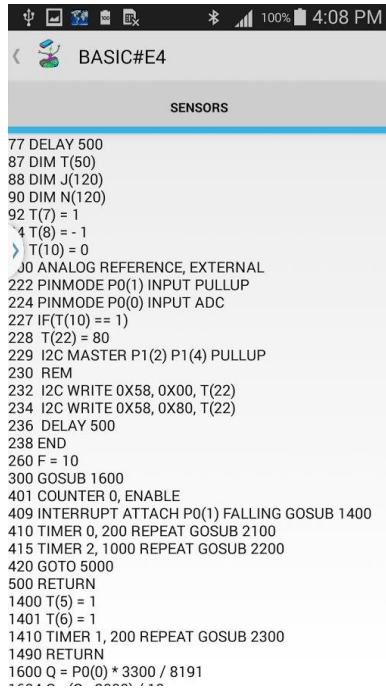


Figure 3.11 BotBasic App Command Page



The screenshot shows a mobile application interface for 'BASIC#E4'. At the top, there is a status bar with various icons and the time '4:08 PM'. Below the title bar, the word 'SENSORS' is centered. The main area contains a list of BASIC program lines, including delay, timer, I2C, and interrupt-related commands.

```
77 DELAY 500
87 DIM T(50)
88 DIM J(120)
90 DIM N(120)
92 T(7) = 1
  4 T(8) = - 1
  > T(10) = 0
  J0 ANALOG REFERENCE, EXTERNAL
222 PINMODE P0(1) INPUT PULLUP
224 PINMODE P0(0) INPUT ADC
227 IF(T(10) = 1)
228 T(22) = 80
229 I2C MASTER P1(2) P1(4) PULLUP
230 REM
232 I2C WRITE 0X58, 0X00, T(22)
234 I2C WRITE 0X58, 0X80, T(22)
236 DELAY 500
238 END
260 F = 10
300 GOSUB 1600
401 COUNTER 0, ENABLE
409 INTERRUPT ATTACH P0(1) FALLING GOSUB 1400
410 TIMER 0, 200 REPEAT GOSUB 2100
415 TIMER 2, 1000 REPEAT GOSUB 2200
420 GOTO 5000
500 RETURN
1400 T(5) = 1
1401 T(6) = 1
1410 TIMER 1, 200 REPEAT GOSUB 2300
1490 RETURN
1600 Q = P0(0) * 3300 / 8191
```

Figure 3.12 BotBasic App lined program example

3.5 BotSpine Performance

Upon BotSpine design completion, some of the performances and features were tested to provide a general idea of the performance of BotSine and its applied range.

BotSpine features are mostly based on the features of CC2541 [38]:

- 2.4 GHz Bluetooth low energy RF signals
- 250 kbps, 500 kbps, 1 Mbps, 2 Mbps Data rate
- Low power
 - ◆ Advertising: 0.6 mA
 - ◆ Connected: 1.3 mA
 - ◆ OLED displaying: 9.6 mA
 - ◆ Sleep: 1.7 μ A
 - ◆ Supply voltage: 2 V – 3.6 V
- 8 KB RAM
- 4 timers
- Battery monitor
- 12 bit ADC with 8 channels
- 2 USARTs to support 4 channels of SPI
- 1 Serial channel
- 20 external general-purpose I/O Pins
- 5 PWM channels
- 1 buzzer pre-set pin
- I2C interface
- Fast pulse interrupt counter up to 30 kHz

Based on our tests, BotSpine could become overwhelmed if the work load is overloaded. For example, updating OLED display during high frequency (more than 30,000 counts per second) interrupts and synchronizes multi-timer tasks, BotSpine has a higher risk to break down. It is resolved with more delays between OLED display tasks and more efficient timer arrangement. However, with the limit of an 8051 microcontroller core, there may not be room for improvement.

Compared to other similar solutions, Raspberry Pi or Arduino, BotSpine consumes ultra low power.

Table 3.6 *Power Consumption Comparison*

Current (mA)	BotSoine	Raspberry Pi (A) [44]	Arduino (YUN) [44]
Idle	0.6	120	240
Active	1.3	150	280

4 Implement BotSpine in Radon Sniffer

BotSpine has been implemented into interesting and useful projects. The sample projects built by EIC are the autonomous robot that scans floors for radioactive contamination and the world's smallest two-player game of Pong [45].

The objective of BotSpine implementation in this thesis is to re-design radon sniffer using BotSpine. Radon Sniffer implementation is to prove how BotSpine can be implemented in industrial products and of commercial usages.

VS472 Radon Sniffer (abbreviated to VS472) is a radioactive detector developed by EIC that was designed with a large circuit board ($143 \times 95 \text{ mm}$) and complicated firmware coding, as well as an inconvenient multi-button control panel operation. With the implementation of BotSpine, the new version of radon sniffer will be able to:

- Minimize its circuit size so that the overall packing of VS472 can be smaller and more convenient. Replace the giant hardware circuit board with miniature circuit boards header connected in parallel including BotSpine board and other relative circuit boards, such as high voltage and amplifying board and regulated pump circuits, in the size dimension of $45 \times 40 \times 14 \text{ mm}$.
- Reduce battery voltage and battery capacity, while retaining the capability of operation. The new sniffer replaces five AA rechargeable batteries for the voltage of 6.5 with only one pack of +3.7V rechargeable lithium battery.
- Simplify the 12-button control panel to one button control. The button is used to start or stop operation or to toggle screens while other function selections are done through the console application.
- Make itself wirelessly connectable. BotSpine provides SV472 with BLE connection capability of wireless control, parameters modification and data monitoring through smart phone apps.
- Simplify program codes. With BotSpine firmware support, the sniffer can be built with hundreds of lines of BASIC codes. Time and effort is saved to build functions in BASIC

rather than C language, and over the air download makes future firmware and BASIC code update more convenient.

4.1 Radon Sniffer Principles

VS472 radon sniffer is a radon activity concentration detector. The overall system schematic of the previous sniffer version is depicted in Figure 4.1. The system equipment includes VS472, Lucas cell, air hoses, filter, desiccant, and the air intake.

The principle of VS472 is to convert radon decay into digital pulses. The air is sucked into the air intake and purified without moisture and radon progeny through the filter and desiccant. Subsequently, the sampled air enters the Lucas cell. The Lucas cell is coated by a Zinc sulfide film that translates the radon activity concentration into alpha particles. Inside VS472, there is a photomultiplier tube (PMT) connected with the Lucas cell. The alpha particles are detected inside the PMT. The detected signal is transmitted to digital pulses, which are wired to the circuit board for calibration.

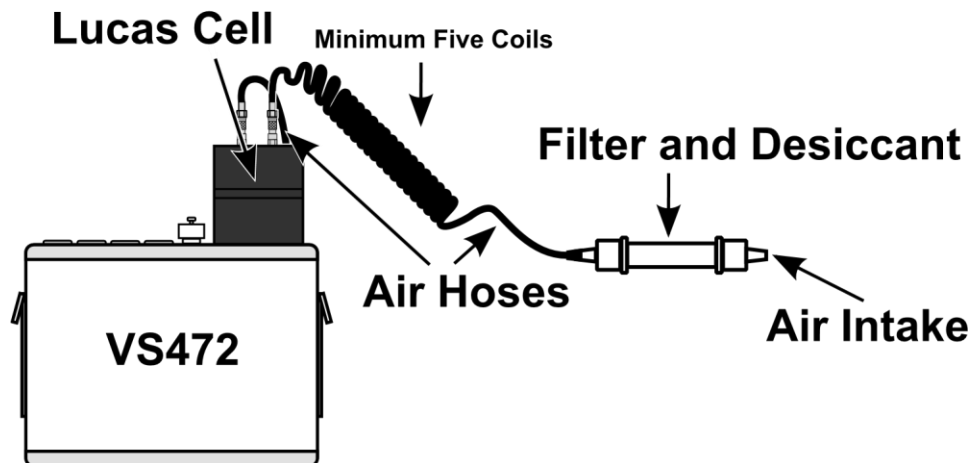


Figure 4.1 VS472 Radon Sniffer System Schematic

The new design radon sniffer follows the same principles as the previous sniffer. It intakes radon from filtered air and converts them into alpha particles that are detectable in PMT and finally transmitted into digital pulses for radon activity concentration calibration. Luca cell and PMT remain implemented in the new design, while the circuit, power supply, keypads, screen and pump are replaced or updated.

VS472 has a few operation modes, including pump mode, sniffer mode and timer mode. Pump mode involves pumping the air into the cell only. Sniffer mode is used to detect and calibrate the radon activity concentration levels. The timer mode is used to detect and display radon pulse counts

for the user-defined time frame. Sniffer mode has more than one calibrating reading; there are 15-second, 5-minute and total average radon activity concentration calibration readings available, referring to SHORT, LONG and TOTAL average radon concentration respectively.

The previous VS472 had a twelve-button control panel, which controlled not only turning the sniffer on and off, but also mode switching, screen toggling and calibration unit conversion. In a new view of VS472 design, the majority of the buttons are removed, with only one button used to handle necessary switches. The renewed basic operation of VS472 is:

- 1) Power up VS472 so that VS472 starts advertising and becomes connectable via BLE;
- 2) Connect VS472 with the console app installed in smart phones;
- 3) Select the desired operation mode and the running time period, if required, on the console app. NOTE: if the operation mode is changed during operation, the display may be updated, but the calibration may be inaccurate because it does not start with the proper settings;
- 4) Press the button for more than two seconds to start running the selected operation. NOTE: it does not matter if VS472 is connected to the phone or not at this point, as long as the mode has been selected prior to this;
- 5) Press the button for less than seconds to toggle OLED screens during operation. In Sniffer mode, the screen will be toggled with radon average concentrations in a different time frame, from 15-second average to 5-minute average to total average since started. There is also OLED off screen toggling in all modes;
- 6) Press the button for more than two seconds to terminate the operation;
- 7) Power down VS472 or leave it sleep for further operation.

The following sections delineate the radon sniffer design in terms of hardware, firmware and software, respectively.

4.2 Radon Sniffer Hardware

The assembled radon sniffer is depicted in Figure 4.2. The hardware of radon sniffer includes:

- PCB circuit boards:
 - ◆ BotSpine board with OLED screen
 - ◆ HV board to supply high voltage to drive PMT
 - ◆ AMP board to amplify PMT output signals
 - ◆ Pump circuit with 1.3MHz Step-Up DC/DC converter
- Air pump
- PMT
- Lucas Cell
- Rechargeable battery
- Button
- Air tubes
- Air filter and Desiccant

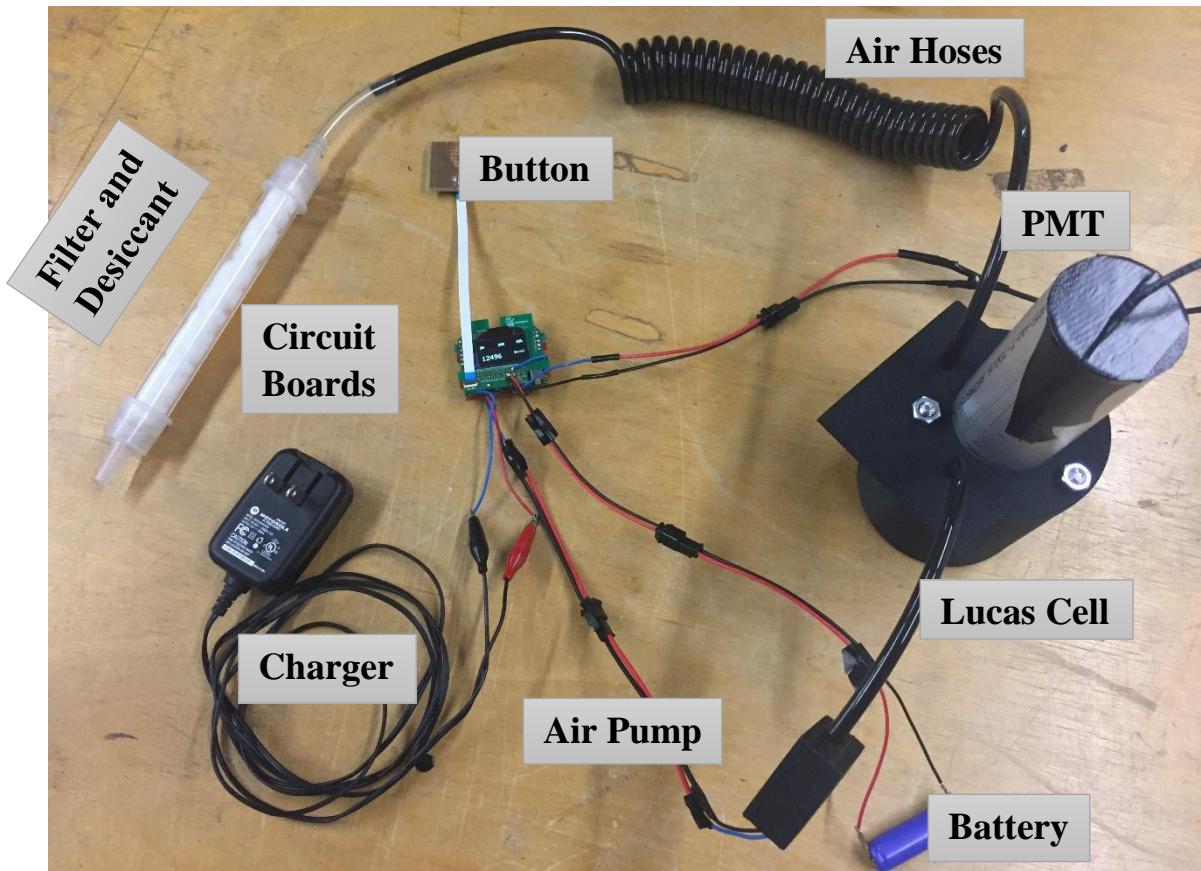


Figure 4.2 VS472 Radon Sniffer Assembling

4.3 Radon Sniffer Firmware

The firmware code of the sniffer design uses BotSpine firmware version 2.0. BotSpine firmware version 2.0, as introduced in Section 3.3.2, acts like a heavily modified interpreter in BASIC. This indicates that developers only need to design and program the radon sniffer functions on the BASIC language level, and the firmware embedded in BotSpine board will handle all lower levels of corresponding functional and mechanical initializations and related complicated computing. After downloading the up-to-date BotSpine firmware into BotSpine board, the sniffer firmware design is ready for the final BASIC functional programming.

The finished BASIC program code for VS472 radon sniffer is attached in Appendix.

The idea behind the BASIC code is: when the device boots up, it starts running the preloaded program and starts advertising with its unique address and given device name. There is one BLE service built in the sniffer program with three characteristics, characteristic ID, ID values and counts, respectively. The logic behind creating as few characteristics as possible is that each characteristic occupies 100 bytes out of 8000 bytes of total free memory, while no other structural blocks takes more than a few bytes.

Upon connection via BLE, users can read or write a characteristic ID and its values through the apps. In terms of characteristic IDs and values, Table 4.1 outlines the relationship defined between characteristic ID and ID values. After the ID number is selected, the user will need to write the desired number into the “Value” characteristic. For example, all characteristics are initialized to zero when rebooting. ID-0 and its ID value 0 indicate that, by default, the selected operation mode is sniffer mode. If the user wants to operate timer mode, he/she will need to write 1 in ID values, while 0 remains for ID. In addition, if timer mode is chosen, the running time also needs to be defined or timer mode will run for 1 minute by default. Counts characteristic is separated from IDs because it must be consistent for reading.

Table 4.1 *Characteristic ID and ID Values*

ID Values ID	0	1	2
0 – Mode	Sniffer Mode	Timer Mode	Pump Mode
1 – Run Time (Timer Mode)	1 – 60 min		
2 – Battery %	0% – 100%		

The starting point of operation entails pressing the button for more than two seconds when no operation is performing at that point. OLED screen will read “START” for two seconds and OLED off. The device then immediately sets up the corresponding mechanical and software registers, such as pin initializations, timer and initialization interruption. It takes 15 seconds to display or update the display on the operating calibration screen, which is based on a 15-second timer for all the pulse reading, calibration and display updates.

During operation, OLED screen can be toggled by instantly pressing the button for less than two seconds. In Sniffer mode, users can have the radon activity concentration for different time periods by toggling screens, from 15 seconds average to 5 minutes average to total average since started. There is OLED screen off to be toggled in all three operation modes, which helps to save power when no display is needed.

Pressing the button for more than two seconds during operation halts the current operation, although timer mode may have stopped itself when the time is up.

Upon the fundamental sniffer program introduced above, the additional LED flashing and buzzing functions may be added to indicate recharging states in the future.

4.4 Radon Sniffer Performance

The firmware performance, BLE characteristic service and the result calibrations are possible assessment items that are used to test and evaluate the performance of the radon sniffer.

The firmware performance includes button interrupts, timer tasks, IO signals and OLED display. The finished sniffer design firmware behaves as expected and described in section 4.3.

In terms of BLE services, the sniffer can be observed in the nearby BLE devices scan list and connectable. Considering the fact that the BotSpine android console app does not have characteristic reading or writing capability, the service test uses one of the generic BLE apps, called LightBlue from PunchThrough.

After connected to LightBlue, it shows all BLE services built in the device. Service UUID F100FFD0-0451-4100-B100-000000000000 is the service built upon the BASIC program, as depicted in Figure 4.3. As described in section 4.3, characteristic ID and ID values are to be modified by clicking in the displayed characteristics to “Write new value”. Counts characteristic can be read or heard for notifications.

The Figure 4.4 left snap displays writing hex 1 into ID-0 value, which changed operation mode from sniffer mode to timer mode. From the right snap of Figure 4.4, it was the characteristic read or listening for notification of counts sending from VS472. This function will be used when designing the sniffer phone app. The received pulse counts will be accurately computed and calibrated to radon activity concentrations on the app side, while the BotSpine has the rough computing for display. The results showed that BLE services of BotSpine design sniffer were functionally adequate.

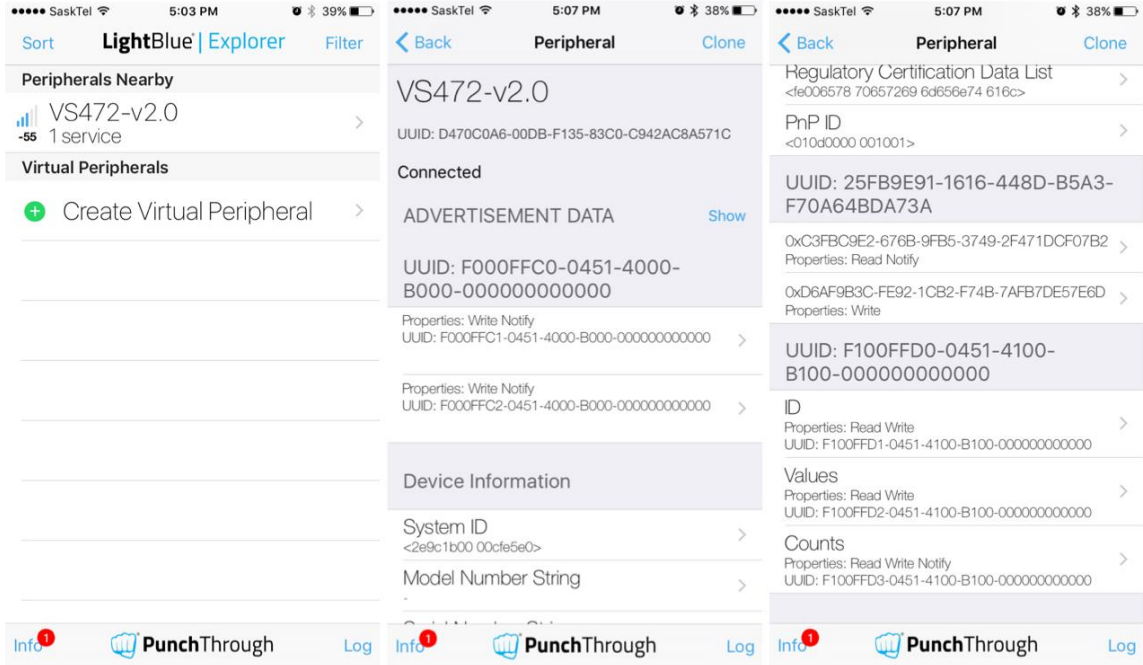


Figure 4.3 VS472 Connected to Generic BLE Apps

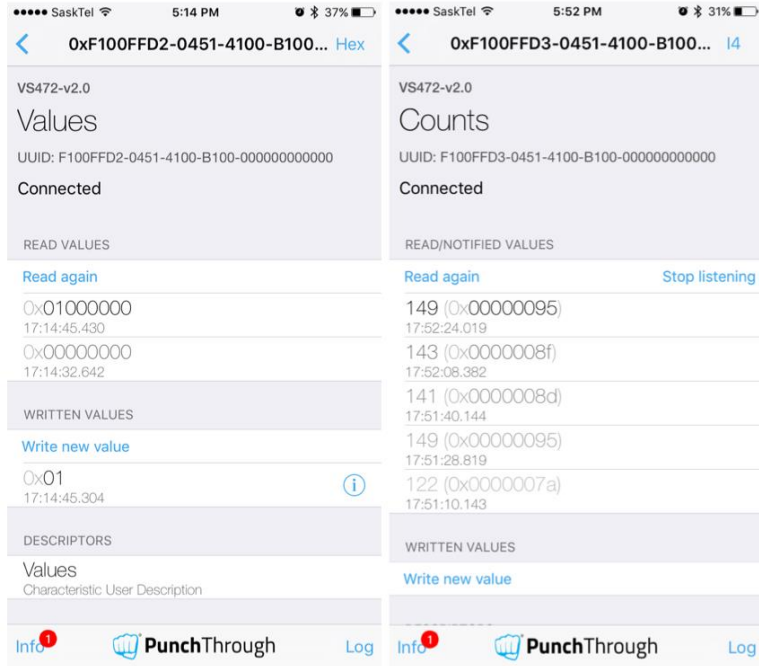


Figure 4.4 VS472 Characteristic Writing and Reading

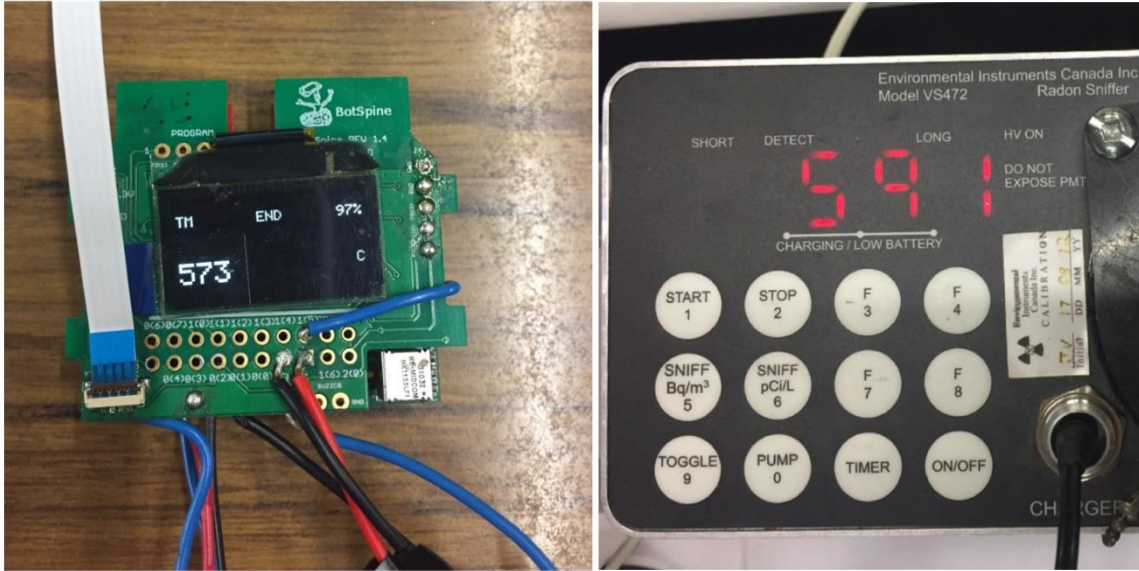


Figure 4.5 One Minute Counts Reading from BotSpine and previous Radon Sniffer

Calibration accuracy was assessed by comparing the calibrations of BotSpine and the original version of sniffer using exactly the same radiative source within the same measurement environment.

Figure 4.5 illustrates the results of pulse counts in one minute timer mode. BotSpine designed sniffer had 573 counts, while the old model of radon sniffer had 591 counts. Within the measurement error, it can be deduced that both devices have similar pulse counts in the same period of time, indicating that the pulse interrupts worked as expected on both hardware and firmware.

Although the pulse detection fluctuates along measurements, the radon concentration would eventually become stable at a level if the detection environment is certain and the radiative source is stable. To determine if the BotSpine design radon sniffer achieves radon activity concentration measurement requirement, the radon readings in sniffer mode from BotSpine design sniffer and old version VS472 sniffer are to be compared. Considering the fact that sniffers indicate radon concentration averages in a certain time period and the radon activities usually take time to become stable, a long measurement time frame in longer period average would reflect the radon activity trending in a more comprehensive and convincing manner.

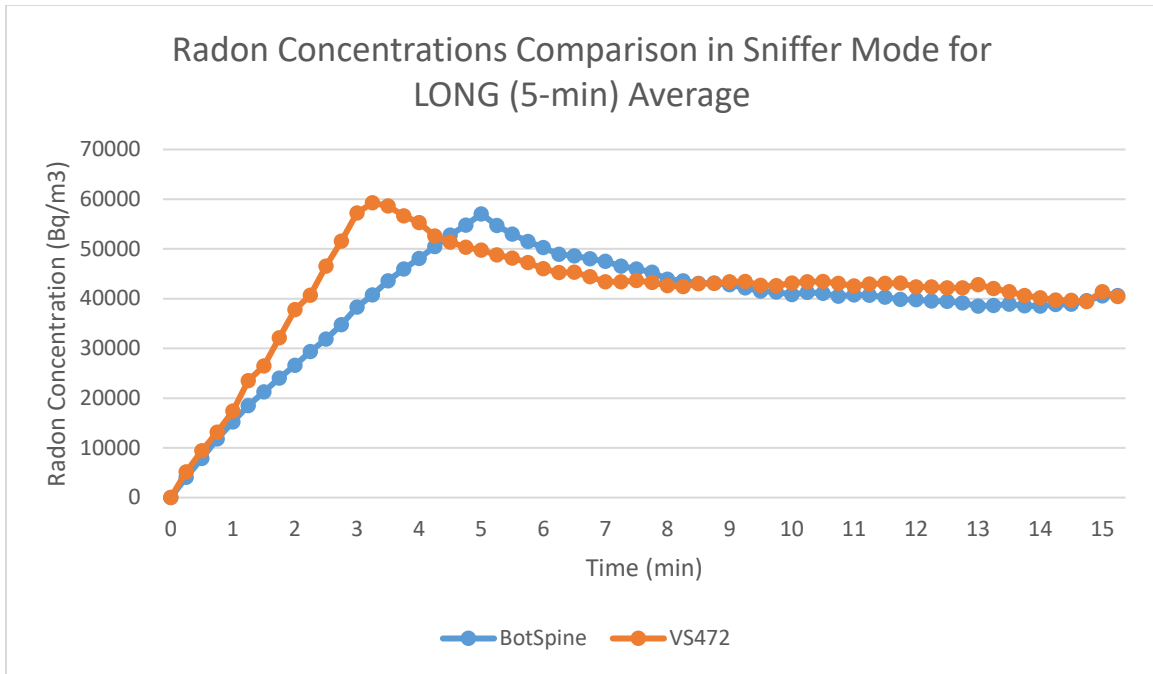


Figure 4.6 Radon Concentrations Comparison in Sniffer Mode for LONG (5-min) Average

Therefore, the LONG (5 minutes) average radon concentrations were measured for more than 15 minutes, running sniffer mode on both the BotSpine designed sniffer and original sniffer. Both devices use the same radiative source, placed in the same measurement environment.

Chart table 4.2 outlines the results of the readings. As discernable from the line chart, both five-minute averages generally boost up from 0 to nearly 60,000 Bq/m^3 in the first few minutes, then gradually decline to 40,000 Bq/m^3 and remain at a level of 40,000.

The difference is that the original VS472 model reaches the peak in 3 minutes, while it takes BotSpine designed sniffer 5 minutes to climb to the top. The logic behind the peak delay may be that there is a small air gap between the PMT and the radiative source in BotSpine sniffer measurement, while the original VS472 does not have any gaps in between. The air gap may cause the data peak, as it makes the alpha particles travel a slightly longer to be detected by PMT.

After all, neglecting the air gap error, the overall radon activity concentrations reflect identical trending and the resulting average concentrations are comparable.

Power consumption is not one of the performance assessments, as the power supplies and air pumps are completely different. The original VS472 uses 6.5 volts of battery and BotSpine sniffer uses

only 3.7 volts. The air pumps are driven by different voltage and regulators. Therefore, the operating current drains are incomparable.

In summary of BotSpine designed radon sniffer performance, the hardware and firmware functional tests are performed as designed. The BLE function of radon sniffer could be scanned and connectable. The built in BLE service and characteristics transfer the correct data between BotSpine sniffer and smart phones. The result detection is comparable to the original radon sniffer.

The radon detector rebuilds using BotSpine is less time and effort consumption which depicts BotSpine is facile development platform for IoT applications. The built radon detectors are now available to purchase in EIC.

5 Conclusion

To overcome the empty gap of universal control interface between smartphones and sensor/actuators, this thesis proposed BotSpine. BotSpine is composed of a PCB digital circuit board, an BASIC interpreter structure firmware installed and a software android app called “BotBasic”. After technology challenge investigation and target application potential, we chose one of the available wireless technologies, BLE, of smartphones as our communication methods due to its ultra-low energy efficiency and satisfying accuracy. According to currently available BLE microcontroller chips, CC2541 is a low cost and efficient system-on-chip selection. Based on the hardware selections, we designed BotSpine hardware board to be a small sized, easily accessible (20 external IOs) and multifunctional embedded digital board. Based on CC2541 BLE stack, we developed BotSpine firmware, of which the second version had an embedded BASIC code interpreter. We also developed software BotBasic on Android system to support smartphones to BotSpine board connection and control. At last, the radon sniffer implementation employed BotSpine performed the expected behaviours with notably improved development cost and effort.

Now with BotSpine, people can easily start building their customised projects by wiring BotSpine board to their electronics, robotics or sensors and start programing on their smart phones. BotSpine is designed as a generic development interface for users who pursue easy and rapid connection and control over their desired electronics. It is suitable and user friendly for starters, but still requires basic mechanical and electrical skills for ascertaining the circuit connections, as well as basic programming skills.

As BotSpine has been promoted and introduced to product developers and industrials, it has piqued the interest of people, with promising potential for various projects. BotSpine is of commercial use in EIC and their recent sale products are made from BotSpine, such as the radon sniffers in the implementation. The next step of BotSpine development entails identifying and working on areas that require improvement; this development includes extending 26 variables, clearing a portion of the flash with valid stored programs remaining, or expanding memory space through the use of external flash.

To keep BotSpine getting along with IoT, there will be some future developments on BotSpine. One of possible developing direction is replacing CC2541 MCU because CC254x series are no longer carried along by TI supports and TI has changed to focus BLE MCU on Arm microchips recently. Wireless MCUs are updated more frequently than they have come along and it is possible that in a near future, 8051 architecture and CC2514 may not meet the needs of BotSpine potential applications. Therefore, BotSpine may seek for a new MCU and more popular and potentially longer supported firmware architecture. Another future development direction could be how to make BotSpine more easily adapted to the Things network so that BotSpine can be accessed over the Internet out of BLE range.

References

- [1] He, J., Lo, D. C., Xie, Y. & Lartigue, J. (2016). Integrating Internet of Things (IoT) into STEM undergraduate education: Case study of a modern technology infused courseware of embedded system course. *2016 IEEE Frontiers in Education Conference*, 1-9. doi: 10.1109/FIE.2016.7757458
- [2] Morelli, B. (2014). The Internet of Things explodes. *IHS Quarterly, Q1 2014*, pp.9. Retrieved from <http://cdn.ihs.com/www/IHS-Quarterly-Q1-2014.pdf>
- [3] Shah, S. H. & Yaqoob, I. (2016). A survey: Internet of Things (IOT) technologies, applications and challenges. *2016 the 4th IEEE International Conference on Smart Energy Grid Engineering*, 381-385. doi: 10.1109/SEGE.2016.7589556
- [4] Smith, A. & Anderson, J. (2014, Aug 6). Predictions for the State of AI and Robotics in 2025. *AI, Robotics, and the Future of Jobs*, pp.3. Retrieved from <http://www.pewinternet.org/2014/08/06/predictions-for-the-state-of-ai-and-robotics-in-2025/>
- [5] Loianno, G., Cross, G., Qu, C., Mulgaonkar, Y., Hesch, J. A. & Kumar, V. (2015). Flying Smartphones: Automated Flight Enabled by consumer Electronics. *IEEE Robotics & Automation Magazine*, 22(2), 24-32. doi: 10.1109/MRA.2014.2382792
- [6] He, N., Li, Z., Jiang, C., Yang, C., Fan, S., ... Chen, C. (2011). A Sensing Platform to Support Smartphones Accessing into Wireless Sensor Networks. *2011 Eighth IEEE International Conference on Mobile Ad-Hoc and Sensor Systems*, 173-175. doi: 10.1109/MASS.2011.124
- [7] Alma, S., Jasmin, N., Nermin, S. & Aljo, M. (2016). Improving energy-efficiency of real-time health monitoring using wireless sensors and smartphones. *2016 39th International Conference on Telecommunications and Signal Processing*, 396-399. doi: 10.1109/TSP.2016.7760905
- [8] Zoller, S., Reinhardt, A., Wachtel, M. & Steinmetz, R. (2013). Integrating wireless sensor nodes and smartphones for energy-efficient data exchange in smart environments. *2013 IEEE International Conference on Pervasive Computing and Communications Workshops*, 652-657. doi: 10.1109/PerComW.2013.6529574
- [9] Nie, H., Joshi, R. & Li, Z. (2016). Demonstration of a joint power harvesting and communication technology for smartphone centric ubiquitous sensing applications. *2016 13th IEEE Annual*

Consumer Communications & Networking Conference, 252-253. doi: 10.1109/CCNC.2016.7444766

[10] Kumar, R. & Rajasekaran, M. R. (2016). An IoT based patient monitoring system using Raspberry Pi. *2016 IEEE International Conference on Computing Technologies and Intelligent Data Engineering*, 1-4. doi:10.1109/ICCTIDE.2016.7725378

[11] Matsuoka, H., Wang, J., Jing, L., Zhou, Y., Wu, Y. & Cheng, Z. (2014). Development of a control system for home appliances based on BLE techniques. *2014 IEEE International Symposium on Independent Computing*, 1-5. doi: 10.1109/INDCOMP.2014.7011751

[12] Electric Imp. *About Electric Imp*. Retrieved from <https://electricimp.com/aboutus/> on December 14, 2016

[13] Chui, M., Loffler, M. and Roberts, R. (2010). The Internet of Things. *McKinsey Quarterly, March 2010*. Retrieved from <http://www.mckinsey.com/industries/high-tech/our-insights/the-internet-of-things> on Jan 30, 2017.

[14] Lueth, K.L. (2014). Why the Internet of Things is called the Internet of Things: Definition, history, disambiguation. *IoT Analytics*. Retrieved from <https://iot-analytics.com/internet-of-things-definition/> on Jan 30, 2017.

[15] Tan, L. & Wang, N. (2010). Future Internet: The Internet of Things. *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 5, 376-380.

[16] INFSO D.4 networked enterprise & RFID, INFSO G.2 Micro & Nanosystems, & RFID working group of EPoSS. (2008, September 5). Internet of Things in 2020 -- Roadmap for the future. *Workshop "Beyond RFID – The Internet of Things"*, 1-32.

[17] Bandyopadhyay, D. & Sen, J. (2011). Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless Personal Communications*, 58(1), 49-69. doi: 10.1007/s11277-011-0288-5

[18] Pereira, C., Pinto, A., Aguiar, A., Rocha, P. Santiago, F. & Sousa, J. (2016). IoT Interoperability for Actuating Applications through Standardised M2M Communications. *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks*, 1-6. doi: 10.1109/WoWMoM.2016.7523564

- [19] Rose, K., Eldridge, S. & Chapin, L. (2015). The Internet of Things: An Overview – Understanding the Issues and Challenges of a More Connected World. *Internet Society*, 1-80.
- [20] Sanctis, M. D., Stallo, C., Parracino, S., Ruggieri, M. & Prasad, R. (2012). Interoperability solutions between smartphones and Wireless Sensor Networks. *2012 IEEE First AESS European Conference on Satellite Telecommunications*, 1-6. doi: 10.1109/ESTEL.2012.6400136
- [21] Kumar, Y.B. & Ilaiyaraja, S. (2014). Smart Interface for Sensors in Robots. *2014 Sixth International Conference on Advanced Computing*, 145-149. doi: 10.1109/ICoAC.2014.7229763
- [22] Doukas, C., Capra, L., Antonelli, F., Jaupaj, E., Tamilin, A. & Carreras, I. (2015). Providing generic support for IoT and M2M for mobile devices. *The 2015 IEEE RIVF International Conference on Computing & Communication Technologies – Research, Innovation, and Vision for Future*, 192-197. doi: 10.1109/RIVF.2015.7049898
- [23] Shen, C., Choi, H., Chakraborty, S. & Srivastava, M. (2014). Towards a rich sensing stack for IoT devices. *2014 IEEE/ACM International Conference on Computing-Aided Design*, 424-427. doi: 10.1109/ICCAD.2014.7001386
- [24] What is a Raspberry Pi? *Raspberry Pi Foundation*. Retrieved from <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/> on December 15, 2016
- [25] What is Arduino? *Arduino*. Retrieved from <https://www.arduino.cc/en/Guide/Introduction> on December 15, 2016
- [26] Jia, Q., Li, X. & Meijer, G.C.M. (2009). Trade-offs in the Design of a Universal Sensor Interface Chip. *2009 IEEE 8th International Conference on ASIC*, 871-874. doi: 10.1109/ASICON.2009.5351557
- [27] Kruger, H., Lebahn, F. & Ewald, H. (2011). Universal Low Power Smart Sensor Interface Using Two-wires for Data Transmission and Supply. *2011 Fifth International Conference on Sensing Technology*, 384-387. doi: 10.1109/ICSensT.2011.6137005
- [28] Reiter, G. (2014, June). Wireless connectivity for the Internet of Things. *Texas Instruments*, swry010.
- [29] Palle, D., Kommu, A. & Kanchi, R. R. (2016). Design and development of CC3200-based CloudIoT for measuring humidity and temperature. *2016 IEEE International Conference on*

Electrical, Electronics, and Optimization Technologies, 3116-3120. doi: 10.1109/ICEEOT.2016.7755275

[30] Wireless Connectivity Guide. *Texas Instruments*, 4Q 2014, slab056d.

[31] Boddan, P., Pajic, M., Pande, P.P. & Raghunathan, V. (2016). Making the Internet-of-things a reality: From smart models, sensing and actuation to energy-efficient architectures. 2016 International Conference on Hardware/Software Codesign and System Synthesis, 1-10. doi: <http://dx.doi.org/10.1145/2968456.2973272>

[32] Discover Bluetooth. *What is Bluetooth?* Bluetooth SIG. Retrieved from <https://www.bluetooth.com/what-is-bluetooth-technology/discover-bluetooth> on December 15, 2016

[33] Bluetooth core specification. *Specifications*. Bluetooth SIG. Retrieved from <https://www.bluetooth.com/specifications/bluetooth-core-specification> on December 15, 2016

[34] Bluetooth SIG (2014, Dec 2). Core System Package [Low Energy Controller volume]. *Specification of the Bluetooth System v4.2*, 6, 2544-2737.

[35] Bluetooth SIG (2014, Dec 2). Core System Package [Host volume]. *Specification of the Bluetooth System v4.2*, 3, 1693-2396.

[36] Monnier, O., Zigman, E. & Hammer, A. (2015, July). Wireless connectivity design considerations for the industrial IoT. *Understanding Wireless Connectivity in the Industrial IoT*, *Texas Instruments*, 3-5. Texas Instruments SWRY016.

[37] CC2541 Product Compare. *Texas Instruments*. Retrieved from <http://www.ti.com/product/CC2541/compare> on June 2, 2016

[38] (2013, June). 2.4-GHz Bluetooth™ low energy and Proprietary System-on-Chip. Texas Instruments. Rev. ed. SWRS110D.

[39] Qanbari, S., Pezeshki, S., Raisi, R., Mahdizadeh, S., Rahimzadeh, R., ... Dustdar, S. (2016). IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. *2016 IEEE First International Conference on Internet-of-Things Design and Implementation*, 277-282. doi: 10.1109/IoTDI.2015.18

- [40] (2009, April). CC2540/41 System-on-Chip Solution for 2.4GHz Bluetooth low energy Applications User's Guide. Texas Instruments. Rev. ed. SWRU191F.
- [41] Anderson, A. (2008, April). Small Size 2.4 GHz PCB Antenna. *Application Note AN043*. Texas Instruments. Rev. D. SWRA117D.
- [42] (2015, Sep.). CC2540 and CC2541 Bluetooth low energy software developer's Reference Guide. Texas Instruments. Rev. ed. SWRU271G.
- [43] Wilkinson, T. (2014, Aug). *BlueBasic: BASIC for Bluetooth*. Retrieved from <https://hackaday.io/project/2386-bluebasic-basic-for-bluetooth>
- [44] Dicola, T. (2014). Power Usage. *Embedded Linux Board Comparison*. Retrieve from <https://learn.adafruit.com/embedded-linux-board-comparison/power-usage> on Jan 31, 2017
- [45] Sample Projects. *BotSpine*. Environmental Instruments Canada. Retrieved from <http://botspine.com/sample-projects.html>

Appendix

```
NEW
PRINT "VS472 BASIC Version-0.2"

10 ADVERT GENERAL
12 ADVERT "f100ffd0-0451-4100-b100-000000000000"
13 ADVERT NAME "VS472-v2.0"
14 ADVERT END
40 GATT SERVICE "f100ffd0-0451-4100-b100-000000000000" ONCONNECT GOSUB 500
42 GATT CHARACTERISTIC "f100ffd1-0451-4100-b100-000000000000" "ID"
43 GATT READ WRITE A ONWRITE GOSUB 800 ONREAD GOSUB 500
45 GATT CHARACTERISTIC "f100ffd2-0451-4100-b100-000000000000" "Values"
46 GATT READ WRITE C ONWRITE GOSUB 700 ONREAD GOSUB 800
50 GATT CHARACTERISTIC "f100ffd3-0451-4100-b100-000000000000" "Counts"
51 GATT READ WRITE NOTIFY D ONWRITE GOSUB 500
70 GATT END
77 delay 500
80 BTPOKE LIM_DISC_ADV_INT_MIN, 3200
81 BTPOKE LIM_DISC_ADV_INT_MAX, 3200
82 BTPOKE GEN_DISC_ADV_INT_MIN, 3200
83 BTPOKE GEN_DISC_ADV_INT_MAX, 3200

97 DIM T(15)
98 T(3) = 80
99 T(11) = -1
100 DIM R(20)
102 DIM S(20)
104 DIM W(20)

220 PINMODE P1(7) OUTPUT
221 P1(7) = 0
223 PINMODE P0(5) OUTPUT
224 P0(5) = 1
226 delay 500
227 PINMODE P1(5) OUTPUT
228 P1(5) = 0
229 PINMODE P0(1) INPUT PULLUP
230 PINMODE P0(0) INPUT ADC
232 PINMODE P0(2) INPUT ADC
233 PINMODE P1(1) OUTPUT
234 DELAY 1000
235 P1(1) = 0
236 ANALOG REFERENCE, INTERNAL
//digipot settings
254 I2C MASTER P2(0) P1(6) pullup
258 I2C WRITE 0X58, 0X00, T(3)
260 I2C WRITE 0X58, 0X80, T(3)

409 Interrupt attach P0(1) falling gosub 1400
450 GOTO 5000

500 RETURN
//ID#1 mode select
```



```

700 IF A == 0
705 T(0) = C
710 IF T(0) > 2
713 T(0) = 0
714 END
715 END
//ID#2 timer
720 IF A == 1
725 T(9) = C
727 IF T(9) > 60
728 T(9) = 60
730 END
735 END
//ID#3 Battery percentage
740 IF A == 2
745 T(10) = C
755 END
799 RETURN
//ID#1 mode select
800 IF A == 0
805 C = T(0)
815 END
//ID#2 timer
820 IF A == 1
825 C = T(9)
835 END
//ID#3 Battery percentage
840 IF A == 2
845 C = T(10)
855 END
899 RETURN

1400 T(5) = 1
1410 TIMER 1, 200 REPEAT GOSUB 2300
1490 Return

1600 Q = P0(0) * 165 /100
//[95 % - 100 %]
1602 IF Q > 4000
1604 T(10) = (Q - 200) / 40
//[75 % - 95 %]
1606 ELIF Q < 4000
1614 T(10) = (Q - 2400) / 17
//[0% - 75%]
1615 ELIF Q < 3700
1620 T(10) = (Q - 3600) * 10 / 12
1628 END
//finding T(10)attery percentage
1632 IF T(10) > 100
1634 T(10) = 100
1635 RETURN
1636 ELIF T(10) < 5
//shut down on very low T(10)attery
1640 GOTO 2573
1642 RETURN
1644 ELIF T(10) < 40
//beep on low battery

```

```

1664 ELSE
1672 END
1699 RETURN

1700 N = D * 1000000
1702 U = X + (J - X) * 10/179
1704 V = Y + (X - Y) / 154
1706 Z = Z + (Y - Z) / 114
1710 J = N - U - Z
1712 X = U
1714 Y = V
1720 E = J/1000000
1725 F = E * 568
1729 RETURN
//Mode0-5min cal
1800 T(11) = T(11) + 1
1806 IF T(11) == 20
1808 T(11) = 0
1810 END
1812 IF E > 16777216
1814 E = 16777216
1816 END
1820 R(T(11)) = E / 65536
1825 W(T(11)) = (E % 65536) / 256
1827 S(T(11)) = E % 256
1830 FOR P = 0 TO 19
1832 I = R(P)*65536 + W(P)*256 + S(P) + I
1834 DELAY 5
1836 NEXT P
1840 G = I * 29
1870 I = 0
1890 RETURN
//Total avg since start
1900 K = K + E
1903 IF K > 2147483000
1905 K = 2147483000
1910 END
1915 L = L + 1
1920 H = K * 568 / L
1950 RETURN
//Mode0-15sec acc
2100 read #counter 0, D
2102 counter 0, clear
2104 GOSUB 1700
2126 GOSUB 1800
2130 GOSUB 1900
2150 IF T(6) == 1
2155 GOSUB 2700
2158 END
2160 RETURN
//Model
2200 read #counter 0, D
2202 counter 0, clear
2205 M = M + D
2206 IF M > 2147483000
2207 M = 2147483000
2208 END

```

```

2210 T(8) = T(8) + 1
2212 IF T(8) == 4
2214 T(8) = 0
2216 T(7) = T(7) + 1
2218 END
2220 IF T(7) == T(9)
2222 P1(7) = 0
2224 counter 0, disable
2226 TIMER 0 STOP
2228 P1(5) = 0
2230 DELAY 50
2232 T(4) = 0
2234 GOSUB 2700
2236 RETURN
2238 END
2240 IF T(6) == 1
2245 GOSUB 2700
2248 END
2250 RETURN

2300 T(5) = T(5) + 1
2304 IF P0(1) == 1
2305 TIMER 1 STOP
2306 GOSUB 2400
2310 ELSE
2320 END
2330 RETURN

2400 IF T(5) < 10
2402 IF T(1) == 1
2405 IF T(0) == 0
2410 T(2) = T(2) + 1
2412 IF T(2) > 3
2414 T(2) = 0
2416 END
2420 ELSE
2421 IF T(6) == 0
2422 T(6) = 1
2423 ELSE
2424 T(6) = 0
2425 END
2426 END
2427 GOSUB 2600
2428 RETURN
2429 END
2500 ELSE
2510 IF T(1) == 0
2512 T(1) = 1
2514 GOSUB 3100
2516 T(6) = 1
2518 P1(5) = 1
2520 K = 0
2521 L = 0
2525 IF T(0) == 2
2530 TIMER 0, 15000 REPEAT GOSUB 2700
2535 ELIF T(0) == 1
2537 M = 0

```

```

2539 IF T(9) == 0
2541 T(9) = 1
2545 END
2550 T(7) = 0
2551 T(8) = 0
2552 P1(7) = 1
2553 DELAY 50
2554 counter 0, enable
2555 TIMER 0, 15000 REPEAT GOSUB 2200
2556 T(4) = 1
2558 ELSE
2560 T(11) = 0
2561 T(2) = 1
2562 I = 0
2565 P1(7) = 1
2566 DELAY 50
2567 counter 0, enable
2568 TIMER 0, 15000 REPEAT GOSUB 2100
2569 END
2570 RETURN
2572 ELSE
2573 T(1) = 0
2575 GOSUB 3150
2579 P1(7) = 0
2581 counter 0, disable
2582 TIMER 0 STOP
2583 DELAY 50
2584 T(2) = 0
2585 T(6) = 0
2589 P1(5) = 0
2597 END
2598 END
2599 RETURN

//screen toggle
2600 IF T(6) == 0
2602 OLED OFF
2604 ELSE
2606 IF T(0) == 0
2608 GOSUB 2700
2610 ELSE
2612 OLED RENDER
2614 END
2616 END
2620 RETURN

//screen update
2700 OLED OFF
2702 OLED CLEAR
2704 DELAY 100
2706 OLED FONT 1
2708 OLED ROW 1
2710 OLED JUST L
2720 IF T(0) == 0
2722 IF T(2) == 0
2724 RETURN
2726 ELSE

```

```

2730 OLED "SN"
2732 DELAY 20
2734 OLED JUST C
2736 IF T(2) == 1
2738 OLED "SHR"
2740 ELIF T(2) == 2
2742 OLED "LNG"
2744 ELIF T(2) == 3
2746 OLED "TTL"
2748 END
2750 DELAY 20
2752 END
2760 ELIF T(0) == 1
2762 OLED "TM"
2764 DELAY 20
2766 OLED JUST C
2768 IF T(4) == 1
2770 OLED T(7), "min"
2772 ELSE
2774 OLED "END"
2776 END
2778 DELAY 80
2780 ELIF T(0) == 2
2782 OLED "PM"
2784 DELAY 20
2786 END
//write battery%
2790 GOSUB 1600
2792 OLED JUST R
2794 OLED T(10), "%"
2796 DELAY 50
//write units
2800 OLED ROW 5
2804 IF T(0) == 1
2806 OLED "C"
2808 ELIF T(0) == 0
2810 OLED "Bq/m3"
2812 END
2814 DELAY 50
//write values
2850 OLED FONT 2
2851 OLED JUST L
2852 IF T(0) == 1
2854 OLED M
2856 ELIF T(0) == 2
2858 OLED "PUMPING"
2860 ELSE
2862 IF T(2) == 1
2864 OLED F
2866 ELIF T(2) == 2
2868 OLED G
2870 ELIF T(2) == 3
2872 OLED H
2874 END
2875 END
2878 DELAY 50
2889 OLED RENDER

```

```
2890 RETURN
//screen "START"
3100 OLED CLEAR
3102 DELAY 100
3104 OLED ROW 1
3106 OLED JUST C
3108 OLED FONT 0
3110 OLED "START"
3112 DELAY 2000
3114 OLED CLEAR
3116 OLED OFF
3120 RETURN
//screen "STOP"
3150 OLED CLEAR
3152 DELAY 100
3154 OLED ROW 1
3156 OLED JUST C
3158 OLED FONT 0
3160 OLED "STOP"
3162 DELAY 2000
3164 OLED CLEAR
3166 OLED OFF
3170 RETURN

5000 PRINT "END OF VS472"
autorun on
PRINT "LOADING DONE"
mem
reboot
```