



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi

**THE DEVELOPMENT OF A WIRELESS MOBILE RFID READER SYSTEM FOR  
REAL-TIME TRACKING BASED ON EMBEDDED ARCHITECTURE**

**SITI NORAINAH BINTI SEMUNAB**



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi

**THESIS SUBMITTED IN FULFILLMENT OF THE REQUIREMENT FOR THE  
DEGREE OF MASTER OF SCIENCE (COMPUTER AIDED DESIGN  
TECGHNOLOGY)  
(MASTER BY RESEARCH)**

**FACULTY OF ART, COMPUTING AND CREATIVE INDUSTRY  
UNIVERSITI PENDIDIKAN SULTAN IDRIS**

2018



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



## ABSTRACT

The aims of the study is to develop a system that able to overcome the limitations of keeping track inventories on the manufacturing plant. The proposed system is named Wireless Mobile RFID Reader (WiBRED) in order to improve the accuracy and traceability of materials in the industrial warehouse management process. This novel system was developed based on the Waterfall model involving the integration of active and passive Radio Frequency Identification (RFID) and IOIO is a board specially designed to work with Android 1.5 and later device to support wireless communication over a smart phone running on Android operating system. This system was implemented in a Wireless Mesh Network (WMN) environment in which its performance was assessed in terms of maximum read range for indoor environment, multi tags communication efficiency, power consumption, and tag collection time. The findings of the assessment showed that maximum read range reach up to 3.5meter for passive wireless communication and 74meter for active wireless communication. Multi tags communication efficiency resulted more than 75% of data received. The duration of tags collection time for 25 tags is 192.5millisecond and the power consumption is 96.611milliampere. Conclusion, WiBRED is capable in improving the monitoring of warehouse management and control the material flow in the manufacturing industry when combining passive RFID and active RFID connected to Android smart phone and can perform in WMN platform. The implications of the findings suggest that this system can be used to improve existing practice of material flow monitoring of manufacturing processes through greater flexibility and mobility in eliminating errors and non-productive labor and delays caused by repetitive re-keying production data.





## PEMBANGUNAN SENI BINA TERTANAM RFID AKTIF MUDAH ALIH TANPA WAYAR UNTUK SISTEM PENGESANAN MASA NYATA

### ABSTRAK

Tujuan kajian ini adalah untuk membangunkan satu sistem yang dapat mengatasi batasan penyimpanan inventori di kilang pembuatan. Sistem yang dicadangkan ini dinamakan *Wireless RFID Reader (WiBRED)* adalah untuk meningkatkan ketepatan dan kebolehesanan bahan dalam proses pengurusan gudang industri. Keaslian sistem ini dibangunkan berdasarkan model Air Terjun yang melibatkan gabungan *Radio Frequency Identification (RFID)* aktif dan *RFID* pasif yang tertanam dengan peranti *IOIO* di mana *IOIO* adalah papan yang direka khas untuk bekerja dengan peranti *Android 1.5* dan ke atas untuk menyokong komunikasi tanpa wayar dengan telefon pintar *Android*. Sistem ini dilaksanakan dalam persekitaran *Wireless Mesh Network (WMN)* iaitu prestasinya dinilai dari segi julat bacaan maksimum untuk persekitaran tertutup, kecekapan komunikasi multi tag, anggaran penggunaan kuasa litar sistem dan masa pengumpulan tag yang dicadangkan. Keputusan penilaian menunjukkan bahawa julat bacaan maksimum adalah mencapai 3.5meter untuk komunikasi tanpa wayar pasif, manakala 74meter untuk komunikasi tanpa wayar aktif. Kecekapan komunikasi pelbagai tag menghasilkan lebih daripada 75% data yang diterima. Tempoh masa pengumpulan tag untuk 25 tag adalah 192.5milisaat dan penggunaan kuasa adalah 96.611miliampere. Kesimpulannya, *WiBRED* mampu meningkatkan pemantauan pengurusan gudang dan mengawal aliran bahan dalam industri perkilangan apabila menggabungkan *RFID* pasif dan *RFID* aktif yang boleh disambungkan ke telefon pintar *Android* dan boleh dilakukan di platform *WMN*. Implikasi dari penemuan ini menunjukkan bahawa sistem *WIBRED* ini boleh digunakan untuk menambahbaik pemantauan aliran bahan yang sedia ada di dalam proses pembuatan kerana sistem *WIBRED* yang lebih fleksibel dan mudah alih dalam menghapuskan kesilapan buruh dan kelewatan yang tidak produktif yang disebabkan oleh data pengeluaran yang berulang.



## TABLE OF CONTENTS

	<b>Pages</b>
<b>DECLARATION OF ORIGINAL WORK</b>	ii
<b>DECLARATION OF DISSERTATION</b>	iii
<b>ACKNOWLEDGMENTS</b>	iv
<b>ABSTRACT</b>	v
<b>ABSTRAK</b>	vi
<b>TABLE OF CONTENTS</b>	vii
<b>LIST OF TABLES</b>	xiii
<b>LIST OF FIGURES</b>	xv
<b>LIST OF ABBREVIATIONS</b>	xix
<b>LIST OF APPENDIXES</b>	xxii
<b>CHAPTER 1 INTRODUCTION</b>	
1.1 Background Research	1
1.2 Problem Statement	4
1.3 Research Questions	7
1.4 Research Objectives	7
1.5 Research Motivation	8

1.6 Research Contributions	10
----------------------------	----

## CHAPTER 2 LITERATURE REVIEW

2.1 Research Background	11
-------------------------	----

2.2 Overview of Embedded System	13
---------------------------------	----

2.3 Introduction of RFID	15
--------------------------	----

2.3.1 History of RFID	15
-----------------------	----

2.3.2 RFID System	17
-------------------	----

2.3.2.1 RFID Tag	17
------------------	----

2.3.2.2 RFID Reader	21
---------------------	----

2.3.2.3 Host Computer	21
-----------------------	----

2.4 Wireless Communication	22
----------------------------	----

2.4.1 Wi-Fi Technology	23
------------------------	----

2.4.1.1 Wi-Fi Architecture	25
----------------------------	----

2.4.2 Bluetooth	26
-----------------	----

2.4.2.1 Bluetooth Connections	28
-------------------------------	----

2.4.3 Zigbee	30
--------------	----

2.4.3.1 Zigbee Architecture	32
-----------------------------	----

2.4.3.2 Zigbee Topologies	33
---------------------------	----

2.4.4 Comparison of Wireless Communication	35
--------------------------------------------	----

2.5 IOIO OTG Board	36
2.5.1 USB OTG	39
2.6 Mobile Device Operating System	43
2.6.1 Symbian	44
2.6.1.1 Symbian OS Architecture	45
2.6.2 Blackberry	48
2.6.2.1 BlackBerry OS Architecture	49
2.6.3 Windows Phone	51
2.6.3.1 Windows Phone Architecture	52
2.6.4 iOS	53
2.6.4.1 iOS Architecture	54
2.6.5 Android	55
2.6.5.1 Android Architecture	56
2.6.6 Comparison of Operating System	58
2.7 Warehouse Management Process in Manufacturing Industry	62
2.8 Research Gaps	67

## **CHAPTER 3 DESIGN AND DEVELOPMENT**

3.1 Overview	69
3.2 Research Procedure	70

3.3 Hardware Development Design	74
3.3.1 Phase 1	79
3.3.1.1 IOIO-OTG with Smart Phone Integration	81
3.3.2 Phase 2	86
3.3.2.1 Passive Reader	87
3.3.3 Phase 3	89
3.3.3.1 Relay Controller	90
3.4 Software Design and Development	93
3.4.1 Phase 1	97
3.4.2 Phase 2	101
3.4.2.1 Main	101
3.4.2.2 Thread	105
3.4.3 Phase 3	106
3.4.3.1 Decoding Message	110
3.4.4 Phase 4	112
3.4.4.1 WiBRED Application Layout	113
3.5 Summary	119

## CHAPTER 4 RESULTS AND DISCUSSIONS

4.1 Overview	120
4.2 Application Interface Setup	121
4.3 Anti-Collision Test	123
4.3.1 Stationary Mode	125
4.3.2 Mobile Mode	127
4.4 Comparison of Results of the Anti-Collision Test	128
4.5 Maximum Read Range Measurement	130
4.5.1 Stationary Mode	131
4.5.2 Mobile Mode	132
4.5.3 Comparison of Station and Mobile Read Range	133
4.6 Wireless Mesh Network Test	135
4.7 Tags Collection Time	137
4.8 Energy Analysis	139
4.8.1 Calculated Current Consumption	142
4.8.2 Measured Current Consumption	145
4.8.3 Rechargeable Battery Lifetime Estimation	146
4.9 Comparison of WiBRED with Existing Systems	146
4.10 Summary	148



## **CHAPTER 5 CONCLUSION AND FUTURE WORK**

5.1 Conclusion 151

5.2 Future Work 153

**REFERENCES** 154

**APPENDICES**

## LIST OF TABLES

Table No.		Pages
2.1	The Milestone in the Development of RFID Technology	16
2.2	Differences among the four types of RFID tags	19
2.3	Frequencies and Data Rates of Zigbee of the Technology	32
2.4	Comparison of Wireless Sensor Network Technologies	35
2.5	USB Connector	40
2.6	Description of USB Pins	41
2.7	The Comparison of Mobile Operating Systems	61
2.8	Supply Chain Management Processes and RFID Benefits	66
3.1	Specifications of the Key Components	70
3.2	Java Classes	101
3.3	The Details of The ID Channel	107
3.4	The Descriptions of The Packet Message	110
3.5	The Sample Data Stored in The Database	111
4.1	Antenna Parameters	124
4.2	Percentage of Data Received for Stationary Mode	126
4.3	Percentage of Data Received for Mobile Mode	128
4.4	The NLOS and LOS Propagations Based on Stationary Mode	132
4.5	The NLOS and LOS Propagations Based on Mobile Mode	132
4.6	Self-Healing Capability of WMN	137

4.7	Tags Collection Time Based on Stationary and Mobile Modes	138
4.8	The Current Consumption During Transmission and Reception	142
4.9	Calculated Current Consumption Per Circuit	143
4.10	Percentage of Time Usage for Each Mode	144
4.11	The Calculated Current Consumption of The Tag in a Period of 60 S	144
4.12	The Measured Current Consumption	145
4.13	The Measured Current Consumption Of The Tag In a Period Of 60 S	146
4.14	Comparison Of Wibred with Existing Systems	147

## LIST OF FIGURES

Figure No.		Pages
1.1	The Cost of Materials of Canon's Lens Production Based on Conventional System	4
1.2	The Survey Result of Malaysia Companies Regarding The use of RFID Technology	9
2.1	Block Diagram of a Generic Embedded System	14
2.2	Core Component of RFID System	17
2.3	Wi-Fi Network	24
2.4	Wi-Fi Architecture	26
2.5	Bluetooth Network	27
2.6	Bluetooth Topology	29
2.7	The Zigbee Architecture	31
2.8	Zigbee Topologies	33
2.9	Ioio Board	38
2.10	USB OTG Cable Orientation of an a Device and A B Device Acting as a Host a Slave, Respectively	42
2.11	USB OTG Cable Orientation of a B Device And an a Device Acting as a Host a Slave, Respectively	42
2.12	Symbian OS Architecture	47
2.13	BlackBerry 10 Device Architecture	50
2.14	Microsoft's Windows Phone Architecture	53
2.15	The Architecture of Apple's iOS	55
2.16	Android Architecture	58

2.17	The market share of mobile operating systems	60
2.18	The warehouse management process	63
2.19	The Flows of Materials and Information through a factory	64
2.20	Flowchart for the Basic Material Flow	66
3.1	The Research Methodology of the Study Based on Waterfall Model	73
3.2	The Process of the Hardware Development	75
3.3	Comparison of Existing System and Proposed System	76
3.4	The Block Diagram of the Proposed System	77
3.5	The Architectural of Proposed System Design	78
3.6	The Block Diagram of the Hardware Development	79
3.7	Phase 1 of Hardware Development	80
3.8	<b>IOIO-OTG with Smart Phone Integration</b>	<b>82</b>
3.9	Enabling the Developer Options	83
3.10	Enable USB Debugging	83
3.11	UART Communication Test	84
3.12	Selections of Preferred Application	85
3.13	Phase 2 of the Hardware Development	86
3.14	Connection of Passive Reader with PC	87
3.15	List of Firmware Command	87
3.16	Response of SelectTag Command	88
3.17	Scanning Tag with the PC using X-CTU firmware	89
3.18	Phase 3 of the Hardware Development	90
3.19	Schematic of Relay Controller	91
3.20	Communication in Mobile Mode	92

3.21	Communication in Stationary Mode	92
3.22	The Processes of the Software Development	94
3.23	The Block Diagram of the Software Development	97
3.24	The Android Layout of Development in Phase 1	98
3.25	The Flow of Communication between Android Application and Electronic Circuit	99
3.26	Looping Test for Transmit and Reception Process	100
3.27	Taxonomy of MainActivity Java Class	102
3.28	The Layout of MainActivity.java	103
3.29	The Flowchart of handleMessage class	104
3.30	The Taxonomy of Read Tag ID Thread	105
3.31	The Taxonomy of Read Tag Info Thread	106
3.32	Details of ID channel	107
3.33	The Flow of the Database Development	109
3.34	The Parts of the Packet Message	110
3.35	The Descriptions of TR and TD in the Database	112
3.36	The WiBRED Icon Displayed on the Interface	113
3.37	The WiBRED Application Layout	114
3.38	Red Indicate No Accessory Connected	115
3.39	Green Indicate an Accessory is Connected	115
3.40	The Flow of Operational Process of the WiBRED Application	117
3.41	To Scan at Tagged Object	118
3.42	A Display of the Description of a Tagged Object	118
4.1	Database Configuration	122

4.2	The General Hardware Setup	122
4.3	Anti-Collision Test Setup	123
4.4	Antenna 1 and Antenna 2 Used in the Test	124
4.5	Results of Anti-Collision Test Based on Stationary Mode	126
4.6	The Results of Anti-Collision Test Based on Mobile Mode	127
4.7	Comparison of Results of the Anti-Collision Test Based on Antenna 1	129
4.8	Comparison of Results of the Anti-Collision Test Based on Antenna 2	129
4.9	Test Setup for Read Range Measurement	130
4.10	The Read Range Result of Passive RFID Using Antenna 1	133
4.11	The Read Range Result of Passive RFID Using Antenna 2	134
4.12	The Read Range Result of Active RFID	134
4.13	Layout of Wireless Mesh Sensor Network in Multi-Hop Environment	136
4.14	Tags Collection Time Based on Stationary and Mobile Modes	138
4.15	The Setup of the Energy Consumption Measurement	140
4.16	The Waveform of Scan Tag Process	141

### LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
AIDC	Automatic Identification and Data Capture
AP	Access Point
API	Application Program Interface
ATM	Automated Teller Machine
BS	Base Station
BSS	Basic Service Set
CE	Compact Edition
CPU	Central Processing Unit
DES	Data Encryption Standard
DS	Distribution System
ESS	Extended Service Set
HAL	Hardware Adaptation Layer
HNP	Host Negotiation Protocol
I2C	Inter-Integrated Circuit
IBM	International Business Machines
ID	Identification
IERC	Internet of Things European Research Cluster
IOIO-OTG	IOIO-On-The-Go
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JVM	Java Virtual Machine



LAN	Local Area Networks
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LOS	Line-Of-Sight
MAC	Medium Access Control Layer
MCU	Microcontroller Unit
MIDP	Mobile Information Device Profile
NLOS	Non-Line-Of-Sight
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
PHY	Physical Layer
PWM	Pulse Width Modulation
RAM	Random-Access-Memory
RF	Radio Frequency
RFID	Radio Frequency Identification
RIM	Research in Motion
ROM	Read-Only-Memory
SCO	Synchronous Connections Oriented
SDK	Software Development Kits
SIG	Special Interest Group
SPI	Serial Peripheral Interface
TCL	Tool Command Language
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface

USB	Universal Serial Bus
USB-OTG	USB On-The-Go
WAP	Wireless Application Protocol
WiBRED	Wireless Mobile RFID Reader
Wi-Fi	Wireless Fidelity
WIP	Work-In-Progress
WLAN	wireless local area networks
WMN	Wireless Mesh Network
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
WUSB	Wireless Universal Serial Bus
ZC	Zigbee Coordinator
ZED	Zigbee End Device
ZR	Zigbee Router

## LIST OF APPENDIXES

- A Source Code of MainActivity.java
- B Source Code of Getting Tag ID
- C Source Code of Tag Info Thread
- D Source Code of Tag Info Thread
- E Source Code of Tag Description Dialog Box
- F Source Code of Database
- G Measurement Results



## CHAPTER 1

### INTRODUCTION



#### 1.1 Background Research

Internet of Things (IoT) is a global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities (Jia, Feng, Fan, & Lei, 2012). It will offer specific object identification and connection capability as the basis for development of independent cooperative services and applications, which is characterized by high degree of autonomous data capture, event transfer, network connectivity and interoperability.





According to the IEEE Journal OF Internet of Things, an IoT system is a network of networks where, typically, a massive number of objects, sensors and devices are connected through communications and information infrastructure to provide value-added services via intelligent data processing and management for many different applications (Porkodi & Bhuvaneswari, 2014). For example, Radio Frequency Identification (RFID), Zigbee and Wireless Sensor Network (WSN) are used for wireless communication among diverse systems. In addition, The Internet of Things of European Research Cluster (IERC) provides a succinct definition of IoT, stating that it is a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols, where physical and virtual “things” have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network (Gaikwad, Gabhane, & Professor, 2015).

In this regard, the characteristics of RFID serve as a prerequisite for the IoT. Admittedly, RFID is not new technology since it has been popularly used in the early twentieth century. The popularity of RFID lies in its practicality in a wide variety of applications. As such, it can virtually provide almost limitless potentials for future applications. Specifically, RFID technology can be efficiently used to monitor manufacturing products that constantly moving and have low visibility (Bachelor, 2014). Hence, it is not surprising that the use of RFID technology has been pervasive in many industries. More specifically, the technology has almost been fully utilized in the





manufacturing industry, in view of the diversity and complexity of manufacturing products that need to be managed efficiently.

For example Wal-Mart has been using RFID technology since 2003, the impact of which has made the company's operations become more efficient. This technology is poised to become more dominant in this decade as many industries across the globe will experience fast and high growth. For example, the Asia-Pacific region is expected to experience high economic growth in the next 10 years, with many expensive and massive projects being carried out by a strong legion of private companies and government enterprises (Bachelor, 2014). For example, Canon is one of the leading and largest camera producers in the world that is using the RFID technology in their manufacturing process.



In lens production, considerable quantities of waste material resulted from the grinding process. Previously, based on its conventional system, about 1% of defective products could be recorded. Figure 1.1 shows a figure highlighting various costs along the process chain at Canon. For each machining process, the costs of raw materials, system, and the disposal were recorded and distributed between outputs. As shown, 32% of the costs were attributed to material loss. The figure shows that within six months the company managed to cover the cost of deployment of RFID system by only focusing on supervisory labour cost. Clearly, further innovative use of RFID can lead to fewer downtimes, higher productivity, and reduced maintenance cost.



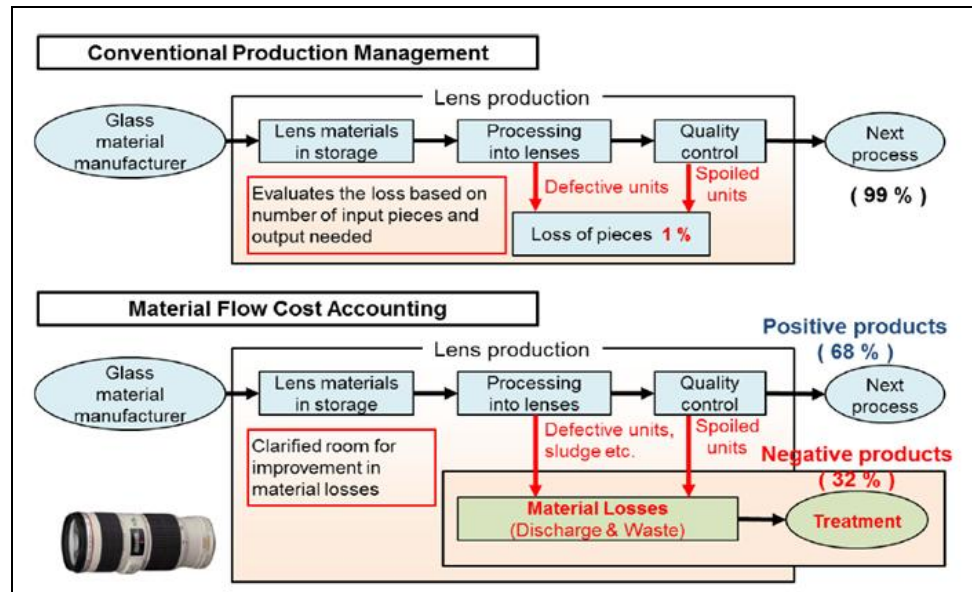


Figure 1.1. The Cost of Materials of Canon's Lens Production Based on Conventional System (Resources, 2013)

## 1.2 Problem Statement

Typically, a firm's stocks consist of a variety of inventories, including finished goods, work-in-progress (WIP) materials, and raw materials (Jou, Wee, & Chen, 2009). As such, keeping track of such materials is very critical to the company's operational efficiency. Nevertheless, this has become a major problem to many manufacturing companies, given their inability to gather and control accurate product and process data, causing massive loss of profits and competitiveness (North, 2013). Nowadays, the manufacturing industry is facing many challenges due to a host of problems, notably because of a lack of reliable tracking systems to control, monitor, and manage the flow materials and items (Ruankaew & Williams, 2013). In fact, material tracking and control systems are new to



the manufacturing industry, as many companies strive to improve their manufacturing productivity and quality of products.

Without any anomalies in the expected physical material flows, the amount of good available in an inventory system would increase each time an expected replenishment is received and decrease each time a demand is satisfied. The existence of several unknown outputs and inputs, such as missing, misplaced, or inaccurate items (which are attributable to unreliable suppliers), the actual material flow would differ from the nominal one (Ruankaew & Williams, 2013). Thus, perfect synchronization between the physical flow and the associated data recorded in the information system is needed to verify if events happened as planned (without any loss or delay of products) and to



identify the reasons for any deviations that took place.

In recent years, the use of mobile phones has been increasing rapidly, far surpassing the use of personal computers (PCs). Therefore, it is not only appropriate but also timely for manufacturing companies to capitalize on the accessibility and mobility of mobile devices to keep track and obtain product data quickly and accurately. With mobile information system, users are no longer required to physically go to the various process stations of a manufacturing shop floor to collect data or information of product parts, items, or components. In this study, the researcher propose a novel monitoring system, designed and developed using a multi-disciplinary approach with the use of an embedded system, RFID technology, Zigbee technology, wireless mesh network (WMN), android operating system, user interface, and Java Application. Specifically, the development of







the proposed system involved the integration of several technologies, notably hardware, mobile application (software), and networking. The system was applied for warehouse management process to automatically and remotely trace and monitor product data without any human interventions, which helped automate and improve the traceability and acquisition of important information.

Currently, most existing systems rely on barcodes and handheld RFID devices, making the tracking process agonisingly slow and susceptible to errors. Such problem is inevitable because barcodes depend on limited line of sight scanning to trigger an event, thus restricting the traceability of materials. Moreover, only a single barcode can be scanned at a given time within a short range of distances. To make matter worse, existing systems do not have a database that allows item tracking at various levels, and they are inherently susceptible to environmental damage and have limited visibility.

Therefore, the proposed system would be able to overcome the above limitations by using the latest Radio Frequency-Identification (RFID) technology. Essentially, an RFID device has a tag affixed to a product that can be identified and tracked using radio waves. In this regard, handheld communication devices are deemed inappropriate for such tracking in complex warehouses as they do not support real-time database transmission and have limited memory space to store large quantity of data. As a workaround, USB devices can be used to transfer important product data to PCs. Additionally, the use of Wi-Fi as a wireless platform can make the implementation of such a tracking system quite expensive.





### 1.3 Research Questions

In this study, the researcher formulated two main research questions to guide the study as follows:

1. Is the proposed tracking system applicable for the manufacturing industry?
2. How does the design of the proposed tracking system support the communication between Android-based mobile phones and RFID sensors using the mesh network platform?



### 1.4 Research Objectives



The main aim of this research is to design and develop an embedded architecture of the proposed system based on the integration of Android mobile phones, passive RFID, and Zigbee technology in the WSN as the platform for traceability of objects or parts. Specifically, the researchers identified three research objectives as follows:

- a. To design and develop the proposed system that can serve as a feasible solution to tracking problem in the manufacturing industry.
- b. To determine the capability of the proposed system based on 2.4 GHz Zigbee to communicate with Android-based smart phone in a mesh network.





- c. To test the performance of the proposed system in a real manufacturing environment.

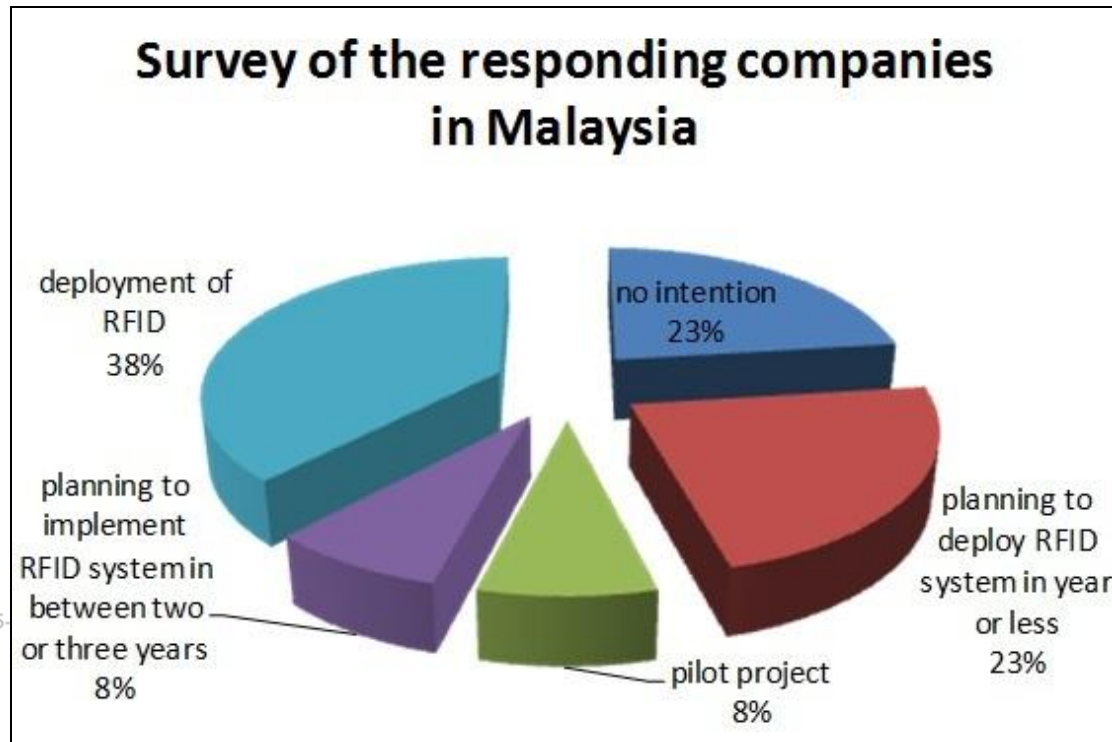
## 1.5 Research Motivation

According to the result survey conducted by Sulaiman, Umar, Tang, & Fatchurrohman(2012), 38% of the surveyed companies indicated that they fully deployed RFID systems in their factories, and another 23% of the companies indicated they were planning to deploy such systems. In addition, 8% the companies stated that they intended to use such systems in the near future. In contrast, the remaining 23% of the companies indicated that they did not intend to deploy such RFID systems.

These finding clearly show that the adoption of RFID technology in Malaysia in quite low. As such, more efforts are required to encourage and help more manufacturing companies in Malaysia to adopt the RFID technology. On the positive side, the findings also indicate that a sizeable number of companies has some plans to introduce such technology in the near future, thus relevant parties should focus such efforts on these companies. Arguably, such companies might have realized the immense impacts that the RFID technology would have on the manufacturing productivity. In view of this positive finding, this research was carried out to help improve material flow traceability and accuracy of a manufacturing company by automating the warehouse inventory tracking



and management process. Figure 1.2 summarizes the survey results of Malaysian companies regarding the use of RFID technology (Sulaiman et al., 2012).



*Figure 1.2.* The Survey Result of Malaysia Companies Regarding The use of RFID Technology (Sulaiman et al., 2012)



## 1.6 Research Contributions

The following are the contributions made by the present research.

### 1. Reducing fixed labor cost

By using WIBRED, manufacturing companies can eliminate transcription errors and non-productive labor and delays caused by repetitive re-keying of production data into enterprise systems. Such a novel automatic system helps eliminate manual data collection, data review, transcription, or re-entry, which effectively minimize the associated fixed and non-productive labor costs.



### 2. Enhanced knowledge

The creative features of this innovative system are based on four combinations of technologies, enabling efficient, automated, and precise object tracking. Admittedly, many industrial applications have been using RFID technology in systems to track movements of human and product. However, this system can further improve such tracking by using embedded passive RFID with Zigbee technology, WSN and Android OS technology, which effectively helps create an effective solution to existing problems. More importantly, the design of the system is based on real-time, distributed principles to deliver latest information, data, or instructions to users, thus improving control and reducing labor cost.





## CHAPTER 2

### LITERATURE REVIEW



#### 2.1 Research Background

In this chapter, the researcher discusses the technology used in the development of the tracking system, namely an embedded system, RFID, Android operating system, WMN technology, and Zigbee technology protocol. Additionally, related works of existing real-time locating systems and existing mobile RFID readers are discussed, especially in terms of the differences in the features between existing mobile RFID readers and the proposed solution. In fact, the proposed system was developed for industrial warehouse application: therefore, technologies related to such application is discussed in detail.





According to Liukkonen, Havia, & Hiltunen(2012), modern manufacturing is constantly facing new challenges due to the evolving production requirements, increasing product variety and complexity, miniaturisation of component and products, new environment regulations, and increasing time-based competition. In this regard, many manufacturing companies have to deal with a host of challenges, such as information gaps between individual manufacturing units, poor visualisation of production stages, and the state of warehouse.

Of late, RFID technology is making a strong presence in manufacturing, given its capability to enhance various manufacturing processes, making them more efficient and productive. In their paper, Huang, Saygin, & Dai(2012) argue that RFID can be used to reduce inventory levels, cut down lead times, and facilitate enterprise-wide operational visibility throughout the entire product life cycle. Moreover, with such technology, information about an object's current location, condition, and history can be stored and retrieved real-time, resulting in better visibility of products that leads to improved decision making (James, Cheng, & Huang, 2012).

Industry 4.0 is the name of the current trend of automation and data exchange in manufacturing technologies, with the numeral "4.0" indicating the fourth industrial revolution. Alternatively, it also means the "Industrial Internet" or the 'Digital Factory'. Fundamentally, Industry 4.0 focuses on the end-to-end digitisation of all physical assets and their integration into digital ecosystems with value chain partners, as opposed to Industry 3.0 that was mainly focused on the automation of single machines and





processes. According to Francisco Almada-Lobo(2015), Industry 4.0 envisages an ecosystem of “smart factory,” in which cyber-physical systems monitor the physical processes of the factory and make decentralized decisions. Hence, the proposed system was designed and developed with similar characteristics of the Industry 4.0, which can help enhance interoperability among connected machines, devices, and people, information transparency, technical assistance, and decentralized decision-making (Deloitte, 2015; Dr. Reinhard Geissbauer, Vedso, & Schrauf, 2016).

## 2.2 Overview of Embedded System



Clearly, many workers and practitioners can recognize the increasing number of intelligent functions of devices that help simplify their daily tasks. Nevertheless, according to Sangiovanni-Vincentelli, Zeng, Natale, and Marwedel(2013), a majority of personnel are almost oblivious to the number of problems faced by the industry in coping with the increasing complexity of embedded functionality. Specifically, there is an urgent need to enhance contents and add new intelligent functions to manufacturing processes to achieve reduced manufacturing lead-time, robustness, and future extensibility. As such, the use of embedded systems can help relevant industries to develop their own devices based on the needs of their daily operations.





An embedded system can be defined as a combination of hardware and software that forms a part of some larger system and are generally designed to perform one or many tasks according to a fixed set of rules, programs, or plans of a working organization. Notably, an embedded system consists of a specially designed micro-processor, programmable read-only-memory (ROM) or random-access-memory (RAM), and other circuitry. A more general-purpose definition of an embedded system is that it is a device that controls, monitors or assists the operation of equipment, machinery, or plant. In recent years, embedded systems have become an important feature in many products, such as home appliances, automobiles, automated teller machines(ATMs), missile guidance systems, and nuclear power plants, which are mostly controlled by such system (Barr & Massa, 2009).

Figure 2.1 shows the block diagram of a generic embedded system. In this study, the researcher focused on industrial manufacturing by proposing a new architecture of embedded system.

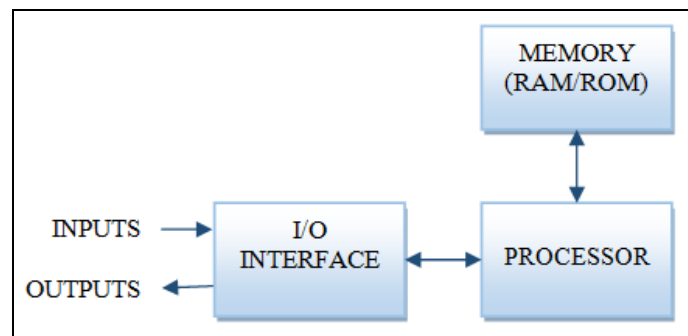


Figure 2.1. Block Diagram of a Generic Embedded System (Medavarapu, 2004)



## 2.3 Introduction of RFID

Actually, RFID technology has been used in so many applications for over 60 years. Essentially, it is a wireless technology or system that uses transmitted radio signals to automatically tag, identify, track, and trace movements of an object (Abdullah, Ismail, & Abdul, 2015). It functions in different frequency ranges based on the different types of tag used. Specifically, RFID technology is classified as a wireless Automatic Identification and Data Capture (AIDC), with which users can send and receive data with no physically contacts occurring between interrogators and tags. Remarkably, it can hold more information than data other carrier systems, such as Bar-Code system (Zare Mehrjerdi, 2011).



### 2.3.1 History of RFID

The roots of RFID technology can be traced back to World War II. The Germans, Japanese, Americans and British were all using radar to warn of approaching planes while they were still miles away. However, there was a main problem in differentiating between an enemy's plane and one's own plane. Fortunately, the Germans discovered that if pilots rolled their planes as they returned to base, radio signals would be reflected back (Srivastava, 2006). Table 2.1 shows the Milestone in the Development of RFID technology.



Table 2.1

## The Milestone in the Development of RFID Technology

Decade	Advancement
1920 <sub>s</sub>	⇒ Development of radar and early RFID, which was a combination of radio broadcast technology and radar
1940 <sub>s</sub>	⇒ The system was able to identify a friend's or a foe's military planes in the second World War
1960 <sub>s</sub>	⇒ Tagging was developed further to improve the safety and security in transporting nuclear materials
1970 <sub>s</sub>	⇒ The first US patent for active RFID tag with rewritable memory ⇒ Passive transponder used to unlock a door without a key ⇒ The technology was transferred to the public sector by Los Alamos Scientific ⇒ Civilian use of the system based on the results of two companies' research
1980 <sub>s</sub>	⇒ First automated toll payment system based on RFID. ⇒ Passive RFID system based on ultra-high frequency (UHF) radio waves to track and identify cows ⇒ First low frequency (125 kHz) systems which enable smaller transponders ⇒ First high frequency (13.56 MHz) systems which offered greater range and faster data transfer rates
1990 <sub>s</sub>	⇒ First UHF system by IBM ⇒ Auto-ID center was established at the Massachusetts Institute of Technology in 1999, using low-cost tags in supply chain by David Brock and Sanjay Sarma
2000 <sub>s</sub>	⇒ Air interface protocols (Class 1 and Class 0), and the EPC numbering schema were developed ⇒ EPCglobal was established to commercialize EPC technology
2010 <sub>s</sub>	⇒ Enhancement in the miniaturization and technology development ⇒ Reduction in the cost of RFID technology development ⇒ Utilization of the technology in numerous fields

Adaptation from (Mika, 2014).

## 2.3.2 RFID System

In general, an RFID system comprises three components, namely a host computer, a coordinator, and a tag. An RFID tag or a transponder can be attached to a product as a means of identification. This tag contains an integrated circuit for storing information, including a serial number, configuration instructions, activity history, modulating and demodulating a radio frequency (RF) signal, and other specialized facilities (Cornel Turcu, Turcu, & Graur, 2008). Figure 2.2 shows the core components of RFID system.

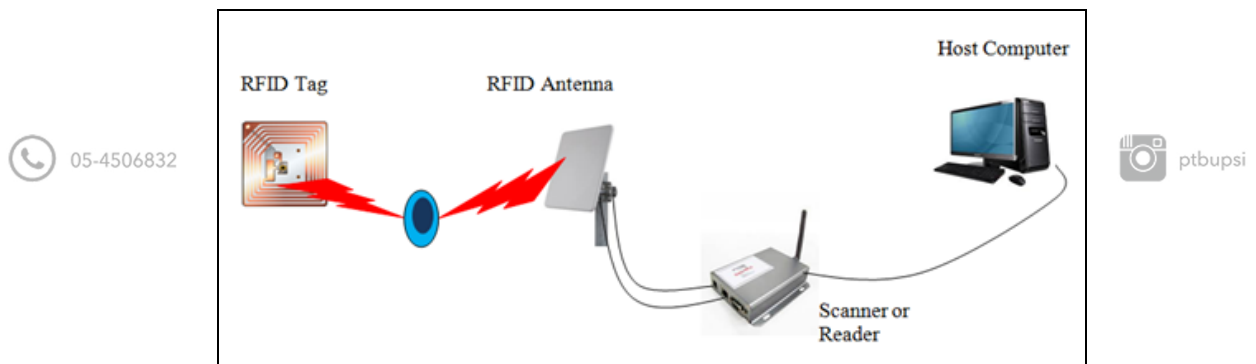


Figure 2.2. Core Component of RFID System (Abdullah et al., 2015)

### 2.3.2.1 RFID Tag

An RFID tag is a tiny radio device that is also referred to as a transponder, a smart tag, a smart label, or a radio barcode. The tag comprises a simple silicon microchip (typically, less than half a millimetre in size) attached to a small flat aerial and mounted on a substrate. The whole device can then be encapsulated in different materials, such as

plastic, depending on the intended use. The finished tag can be attached to an object, such as an item, a box, or a pallet, and read remotely to ascertain its identity, position, or state.

The circuit is attached to a miniature antenna with a label to permit a tag to be attached to a desired physical object. The RFID tag transmits its data in response to an interrogation received from a read-write device called an RFID reader or an interrogator. This device decodes the tag signal and transfers the data to a computer through a cable or wireless connection. The tags and readers are designed with a specific operating frequency. Based on wireless communication, all data may be read from a distance between the RFID chip and the RFID reader. The reading range varies in accordance with the operating frequency, the size of the reader antenna, the orientation of the RFID tag toward the antenna, the tag position with respect to the antenna core, and the tag type. There are three types of tags, namely passive tag, active tag, semi-passive tag, (Zare Mehrjerdi, 2011). The three types of tags are explained below, and their differences are summarized in Table 2.2.

### **1. Passive tags**

Passive RFID tags do not contain a battery; hence, they draw the required power from the radio wave transmitted by the reader. The reader transmits a low power radio signal through its antenna to the tag, which in turn receives it through its own antenna to power the integrated circuit (chip). They have a smaller memory capacity and are considerably lower in cost, making them ideal for tracking lower cost items. Since the 1980s, logistic

and consumer goods industries have depended on passive RFID solution due to low cost of tag and great versatility of the technology (Hakala, 2013).

## 2. Active tags

Active tags are battery powered. They broadcast a signal to the reader and can transmit it over the great distances (100+ meters making it ideal to track high-value goods, such as vehicles and large containers of goods. Shipboard containers are a good example of an active RFID tag application.

## 3. Semi-passive tags

Semi-passive tags use a battery to run its chip's circuitry, but it communicates by drawing power from the reader (Zare Mehrjerdi, 2011). The frequency range is higher compared to passive tags.

Table 2.2

Differences among the Four Types of RFID Tags

Feature	Active	Passive	Semi-passive
<b>Power source</b>	Battery	Induction from electro-magic	Battery and induction
<b>Read distance</b>	Up to 30 meters	3-7 meters	Up to 30 meters
<b>Frequency</b>	High	Medium	High
<b>Information</b>	32 kb or more	2 kb Read Only	32 kb or more

**Storage**

(Read/Write)

(Read/Write)

Generally, active and semi-passive tags are more expensive than passive RFID tags because the former contain more hardware. As such, active and semi-passive tags are reserved for costly items that are used to read over great distances. Yet, this flexibility does have a cost, as active tags require more maintenance and have a limited life span based on on-board power supplies, normally lasting from 5 to 10 years. In contrast, passive RFID tags have lower production costs, allowing them to be applied to less expensive items. In fact, improved passive tag technology has been the driving force for the current wave of RFID adoption, spurred by substantial reduction in cost and increase in operating range. In some cases, active tags and tags with sensors can be used to monitor product quality. Another factor that also influences the cost of RFID tags is data storage. In general, there are four classes of tag as follows:

1. CLASS 0 (Read-only) Tags are the simplest type of tags. They are programmed with unique information (ID number) stored on the read-only chips, which cannot be altered after manufacturing (Deavours, 2005).
2. CLASS 1 (With Read-Write) tags allow users to add information to the tag or write over existing information when the tag is within the range of the reader. Read-Write chips are more expensive than Read-only chips (Deavours, 2005).





3. CLASS 2 (Read Write) tags are the most flexible type of tag, enabling users to read and write data into the tags' memory (Cristina Turcu, 2011).
4. CLASS 3 (Read-Write with on board sensors) tags either semi-passive or active tags, which may contain sensors for recording parameters, such as temperature and pressure, and they can record readings into tags' memory (Vhatkar & Bhole, 2010).

### 2.3.2.2 RFID Reader



An RFID reader is also called an interrogator or a scanner, and its complexity and configuration depend on the functions it need to carry out, which can vary quite significantly from one application to another (Duroc & Kaddour, 2012). It sends RF data to and receives RF data from the tag via antennas. Moreover, the reader may have multiple antennas to send and receive radio waves.

### 2.3.2.3 Host Computer

Data received by the readers are passed to a host computer, which may run specialist RFID software or middleware to filter the data and route such data to the correct application to be processed into useful information.







## 2.4 Wireless Communication

Wireless communication is the transfer of information between two or more points that are not connected by an electrical conductor. In the last 14 years, such communication has become an important feature in the manufacturing of commercial products, and it has also emerged as a popular research topic. Evidently, wireless technologies have made significant progress in recent years, allowing many applications to support high-speed data communications and transmission using sophisticated mobile devices and smart objects. To this end, various applications based on such technologies have been developed to support a diverse range of communications (Saad, Cheikh, Mostafa, & Abderrahmane, 2014). Interestingly, these technologies can be used in many situations where mobility is essential and the use wires is not practical. Lately, the emergence of radio frequency wireless technologies has rendered expensive wiring virtually eliminated.

Of late, wireless networks have been widely deployed in communication industries, because wireless communication removes the restriction of wired connections and supports fast access to the Internet. Essentially, wireless technology enables efficient connectivity between two or more computers to communicate using standard network protocols (Tomai & Toma, 2009).

Globally, wireless technologies have been used in many applications throughout the world. For example, wireless communication via satellites helps connect geographically dispersed countries or places more efficiently. For areas that are close





with one another (such schools, colleges, offices, factories, and industries), wireless sensor networks, such as Bluetooth, Wireless Fidelity (Wi-Fi) and Zigbee, can be used to communicate or transfer data among such various entities. In this regard, the main advantages of wireless sensor networks are their reliability, authenticity, and lower cost, compared to those of the wired technologies.

### 2.4.1 Wi-Fi Technology

Wi-Fi is a wireless technology that allows electronic devices to exchange data over a network, such as the internet (Khanna & Gupta, 2013; Lee, Su, & Shen, 2007) . Such a network is based on wireless local area networks (WLAN) technology using the IEEE standard 802.11, including 802.11a, 802.11b, 802.11g and 802.11n,withwhich centralized router devices that can share Wi-Fi signal (Abinayaa & Jayan, 2014; Anamika Vatsal & Fatima, n.d.; Khanna & Gupta, 2013; Nagarajan & Dhanasekaran, 2015). In this respect, Wi-Fi Alliance, which is the owner of such technology, promotes these standards with the aim of improving the interoperability of wireless local area network products (Tomai & Toma, 2009) Without doubts, Wi-Fi has been an integral component of a communication infrastructure, which is attributed its robustness, expandability, cost-effectiveness, broad coverage area, some non-line-of-sight (NLOS) transmission capacity, small disturbance of links, and mesh topology (Lee et al., 2007). Moreover, it can support high data-transfer rate up to 300 Mbps and throughput rate of 100 to 150 Mbps, depending on the standard used.



Nonetheless, in an indoor environment, Wi-Fi technology faces a common problem called multipath interference due to reflection of signals from the furniture, walls and other obstacles. Despite this problem, Wi-Fi remains popular for home and business networks, because it allows local area networks (LANs) to operate without cables and wiring as illustrate in Figure 2.3. Furthermore, it provides wireless broadband internet access for many modern devices, such as laptops, smart phones, and tablets. However, increasing the number of devices in a single Wi-Fi connection, the strength of signals of each device becomes weak (Khanna & Gupta, 2013).

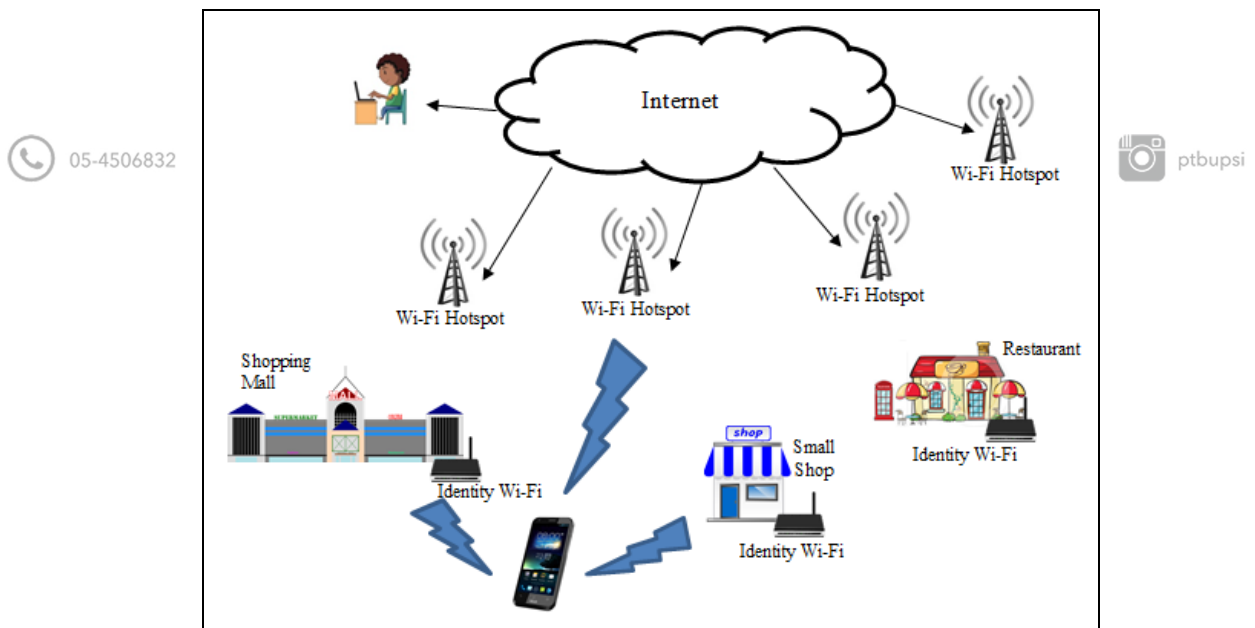


Figure 2.3. Wi-Fi Network (Abinayaa & Jayan, 2014)

### 2.4.1.1 Wi-Fi Architecture

Figure 2.4 shows the architecture of a Wi-Fi LAN consisting of the following components:

1. End user: The device must be based on IEEE 802.11 board, such as radio network interface cards: Personal Digital Assistant (PDA), smart phone and laptop.
2. Base Station/Access Point (BS/AP): a device with antenna that sends and receives radio packets over wireless environment from the end user devices.
3. Basic Service Set (BSS): The cell coverage formed by radio antenna of the Access Point device.
4. Distribution System (DS): The items that provide Internet access to Aps, with all devices communicate and interconnect with each other through a network router.
5. Extended Service Set (ESS): The component that contains all BSSs, APs and DS from a Wi-Fi LAN.

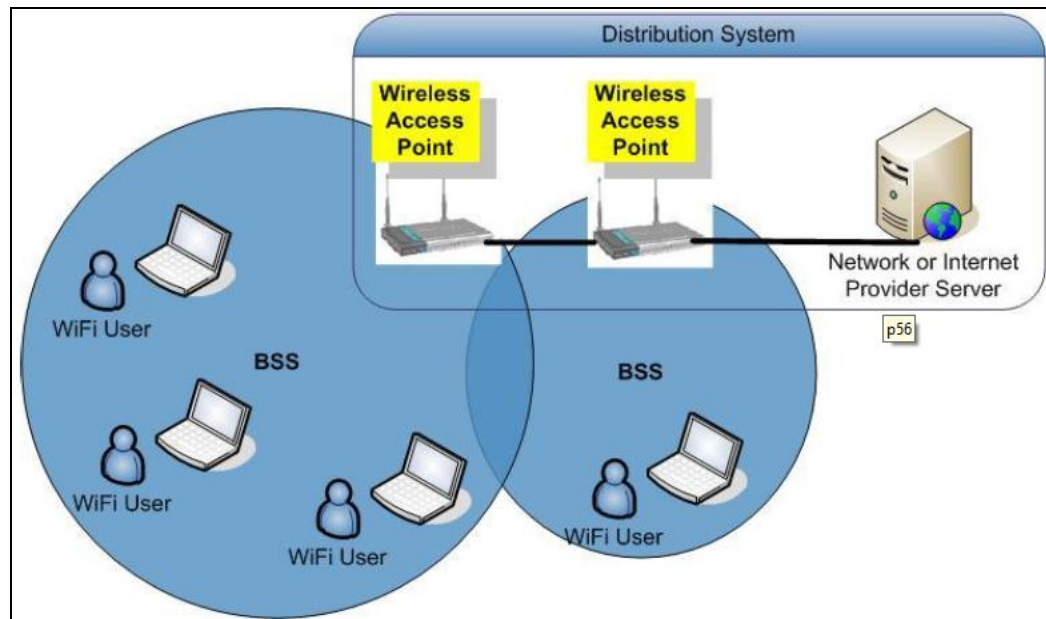


Figure 2.4. Wi-Fi Architecture (Tomai & Toma, 2009)

## 2.4.2 Bluetooth

Bluetooth is a robust wireless technology that requires low power and supports short-range wireless connection between several types of devices, including mobile phones, computers, entertainment systems, and other electronics (Aju, 2015; Idris & Muhammad, 2016; Lee et al., 2007). Such devices need to be placed approximately within 10 meters to each other, and the typical data rate is 2 Mbps (Abinayaa & Jayan, 2014; Idris & Muhammad, 2016). Such devices need to be placed approximately within 10 meters to each other, and the typical data rate is 2 Mbps (Idris & Muhammad, 2016). Typically, Bluetooth signals operate in the 2.4 GHz frequency band, where every device using Bluetooth has a small microchip that can send both data and voice signals (Nagarajan &

Dhanasekaran, 2015). Actually, Wireless Personal Area Network WPAN has been adopted solely for replacing cable technology.

In view of its capability, the use of Bluetooth technology is increasingly expanding at a rapid rate, making its way into several domains, such as automation, health and fitness, mobile telephony, personal computer (PC), and other peripheral devices (Aju, 2015) as shown in Figure 2.5. Bluetooth operates in 2.4 GHz frequency band all over the world. From the historical perspective, Bluetooth takes its name from Harald Bluetooth, and it was developed by an Ericsson-led consortium, including Toshiba, International Business Machines (IBM), Nokia and Intel. In early January 2000, the technology was further promoted by the Bluetooth Special Interest Group (SIG) comprising 1371 members (Verma, Singh, & Kaur, 2015).



Figure 2.5. Bluetooth Network (Aju, 2015)



### 2.4.2.1 Bluetooth Connections

Two connectivity topologies are defined in Bluetooth, namely the piconet and scatternet (Aju, 2015). Based on piconets, it can support up to eight active devices, with a maximum of three synchronous connections oriented (SCO) link (SCO) link. Essentially, the piconet is a WPAN formed by a Bluetooth device serving as a master in the piconet, with one or more Bluetooth devices serving as slaves. Each piconet is defined by the frequency-hopping channel based on the address of the master, where all devices participating in communications in the piconet are synchronized using the clock master of the master (Lee et al., 2007). Bluetooth devices can be connected via point-to-point connection and point-to-multipoint connection to form a piconet (Patel, Patel, & Patel, 2011). Bluetooth radios connect with each other in piconets, which are formed by the master radio simultaneously connecting up to eight slave radios. Scatternet comprises of two or more piconets. It facilitates high densities of communicating devices, making it possible for dozens of piconets to co-exist and independently communicate in close proximity without significant performance degradation. It supports communication more than 8 devices. Scatternet can be formed when a member of one piconet (either the master or one of the slaves) elects to participate as a slave in a second, separate piconet. Figure 2.6 shows the Bluetooth topology of such connections. The Bluetooth radios are symmetric such that any Bluetooth radio can become the master or slave radio, and the piconet configuration is determined at the time of such formation. Typically, the connecting radio will become the master; however, the "master/slave swap" function can reverse such roles.



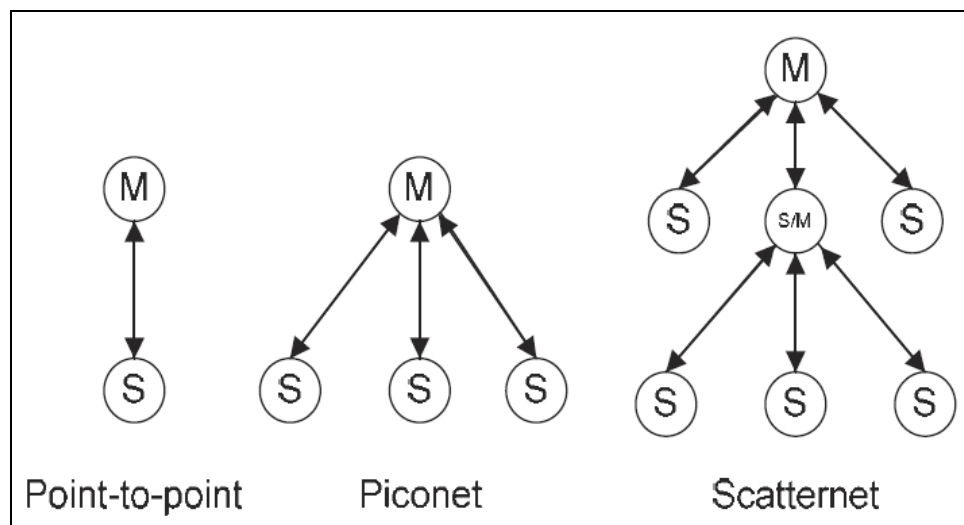


Figure 2.6. Bluetooth Topology (Patel et al., 2011)

Upon successful connection, a Bluetooth device can be in any of the four following states: Active, Hold, Sniff, and Park mode, with the last three being power-saving modes (Patel et al., 2011).

1. Active mode: A Bluetooth device actively participates on the channel.
2. Hold mode: A slave can perform processes, such as scanning, paging, inquiring or attending another piconet or entering a low power sleep mode.
3. Sniff mode: The duty cycle of the slave's listen activity can be reduced, thus enabling the master to only transmit in pre-specified time slots.
4. Park mode: This mode helps conserve power, which would be appropriate for a device in the piconet that only needs to be randomly accessed.



### 2.4.3 Zigbee

Zigbee (IEEE 802.15.4) is a wireless technology that is based on the Institute of Electrical and Electronics Engineers (IEEE) standard 802.15.4, which defines the physical (PHY) and Medium Access Control (MAC) layers, and works on a low data rate standard (Abdulla, 2012). According to (Narayanan, Muthumanickam, & Nagappan, 2015) Zigbee has many advantages compared to Bluetooth technology, characterised by its diverse transmission range of 10 to 100 meter, low cost, and low power consumption at low data rate (Bal, 2014).

Essentially, it uses inexpensive components that can run using 2.4 GHz radio frequency, with data throughput of up to 250 kbps for transferring data between Zigbee modules. Moreover, Zigbee wireless devices can operate for many years with the use of battery power (Bal, 2014). As such, Zigbee technology is suitable for a wide range of monitoring activities, including building automation, health monitoring, automated meter reading equipment, grain storage, remote controls, heating and cooling control devices, fans, and structural integrity (Kim & Ayurzana, 2009).

Figure 2.7 shows Zigbee architecture consisting of four layers. The top two layers are the Application and the Network and Security layer. These layers' specifications are governed by Zigbee Alliance, which provides several manufacturing standards for relevant industries. The bottom two layers are the MAC and PHY layers. These layers' specifications are based on the IEEE 802.15.4-2006 standard to ensure uninterrupted

coexistence of devices without any interference with other wireless protocols, such as Wi-Fi (Netalkar, Kaushal, Shekar, & Shet, 2014).

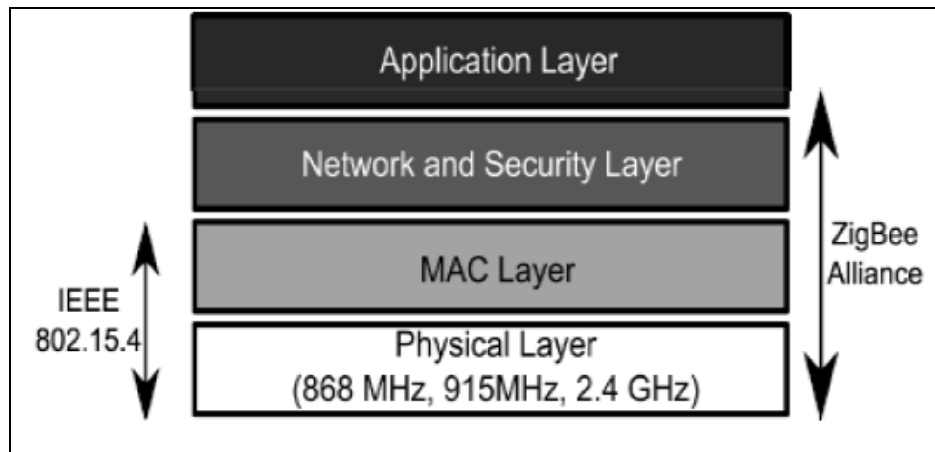


Figure 2.7. The Zigbee Architecture (Netalkar et al., 2014)

Given that Zigbee standard is based on IEEE 802.15.4, implementing a Zigbee network entails compliance with similar requirements of the 802.15.4 network. With IEEE 802.15.4 specification, the design of PHY level can help support low-cost and top-level integrated demand. Table 2.3 summarizes the working band and data transmission rate of the technology.

Table 2.3

Frequencies and Data Rates of Zigbee of the Technology			
Band(ISM)	Geographic Coverage	Burst Data Rate	No of Channels
2.4 GHz	World Wide	250 kbps	16
868MHz	Europe	20 kbps	1
915MHz	Americas	40kbps	10
Adaptation from(Dharmistha & Vishwakarma, 2012)	Adaptation from(Dharmistha & Vishwakarma, 2012)	Adaptation from(Dharmistha & Vishwakarma, 2012)	Adaptation from(Dharmistha & Vishwakarma, 2012)

#### 2.4.3.1 Zigbee Architecture

Fundamentally, the architecture of a Zigbee system consists of three different types of (Netalkar et al., 2014) as follows :

1. *Zigbee Coordinator (ZC)*: As the most capable device, ZC acts as a root and a bridge of the network that controls the entire network formation and maintains the security of the network. There is only one ZC in each network, and it is the device that initiates the formation and establishment of such network.
2. *Zigbee Router (ZR)*: ZR is an optional network that acts as intermediary devices element whose major function is to extend the range of the network. It keeps a routing table and controls the addressing/routing issues for the connected end devices.

3. *Zigbee End Device (ZED)*: A ZED performs specific sensing or control functions. It also has the capability to communicate with the above two types of device, which functions as parent nodes. This relationship allows the node to be asleep for a significant amount of time, effectively extending the battery power. While ZEDs are allowed to periodically cycle in a low-power sleep mode in networks, ZRs and ZCs must always, in general, remain awake. With regard to the network layer, both ZCs and ZRs participate in multi-hop routing activities, and ZEDs only process messages to and from their own associated parent device within their radio transmission range (C. Li, Wang, & Guo, 2010).

There are three network topologies that are specified for the Zigbee network, namely the star, tree, and mesh topology (Aju, 2015; Dharmistha & Vishwakarma, 2012) as shown in Figure 2.8.

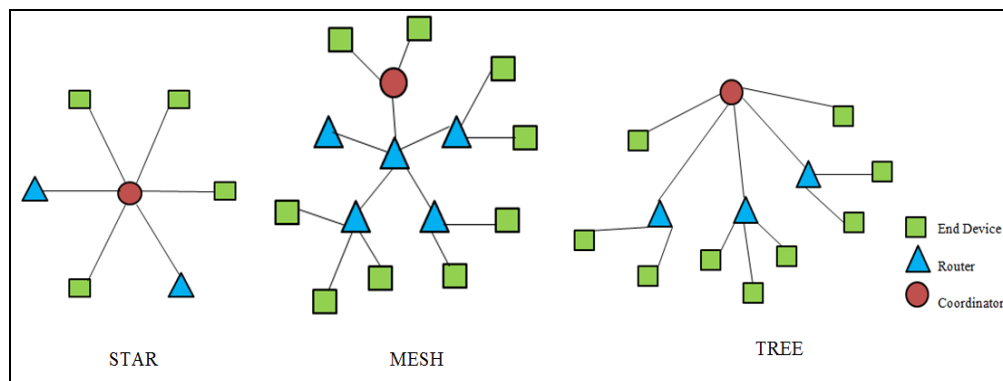


Figure 2.8. Zigbee Topologies

## 1. Star Topology

In addition to end devices, routers can also be used. However, only the router's application will be used, not its function. Therefore, the end devices or routers become the children of the coordinator. The advantage of this topology lies in its simplicity. By contrast, the disadvantage of this topology is that it does not provide an alternative route for packet transmission and reception, thus increasing the workload of the coordinator and causing congestion in the network.

## 2. Mesh Topology

This topology comprises the coordinator, routers, and end devices, with the routers extending the network range over which packets will pass through multiple hops to reach destinations and enabling communication between any source and destination in the network. As such, this topology is also called peer-to-peer multi-hop network. The advantage of this topology is that it provides alternative paths for a packet to reach its destination if a previously selected path failed to function. This topology is also called as a "self-healing" network, making the addition or removal of nodes easier.

## 3. Tree Topology

For the tree topology, the coordinator at the top is connected to several routers and end devices, rendering them as the coordinator's children. Moreover, a router can connect to several other routers and end device, thus extending the network. Only the coordinator and routers can have children and become a parent. In

contrast, the end devices are not allowed to have children, thus disabling them to become a parent in this topology. Similar to the star topology, this topology provides no alternative paths to nodes to reach their destinations, as a malfunctioned parent will disable the communication between nodes in the network, even if such nodes are geographically close. Furthermore, a defective router will prevent its children from communicating with the rest of other nodes in the network (Al-Harbawi, Rasid, & Noordin, 2009)

#### 2.4.4 Comparison of Wireless Communications

Table 2.4 shows the comparison of four different wireless communications in terms of their frequency, maximum data rate, maximum transmission distance, tracking location, and supported topologies. Even though WiMax, Wi-Fi, Bluetooth, and Zigbee operate at the same frequency of 2.4 GHz, no interference among them occurs because their protocols are not the same.

Table 2.4

Comparison of Wireless Sensor Network Technologies

Name	WiMax(4G)	Wi-Fi	Bluetooth	Zigbee
<b>Protocol</b>	IEEE 802.16m	IEEE 802.11 G	IEEE 802.15.1	IEEE 802.15.4
<b>Frequency</b>	2.4 GHz, 10-66GHz	2.4 GHz	2.4 GHz	2.4 GHz, 915 MHz, 868 MHz
<b>Maximum Data Rate</b>	300 Mbps	54 Mbps	732 kbps	250 kbps

<b>Nominal Range</b>	0.3 – 49 km	10-100 m	10 m	10 - 1000 m
<b>Supported Topologies</b>	Star, Mesh, Point-to-point	Tree	Tree	Point-to-point, Star, Mesh
<b>Access Protocol</b>	Request/Grant Internet,	CSMA/CA Data network,	CSMA/CA Cable replacement	CSMA/CA Monitoring, Control
<b>Application</b>	Monitoring, Network Service	Internet, Monitoring		

Adaptation from (Saad et al., 2014; W.Fisher, 2013)

Among all four wireless communications, Zigbee technology is selected because of the read range, access protocol and supported topologies. Eventhough WiMax has longer read range and support mesh and star topologies, but it did not support CSMA/CA access protocol. CSMA/CA is effective in avoiding data collision and reliable in transmission whereas the next data will only be send when the previous data reach its

destination safely.

## 2.5 IOIO OTG Board

IOIO (pronounced “yo-yo”) is a board that connects Android devices (running on Android operating system (OS) version 1.5 and greater) to external hardware, such as sensors, servos, touch displays, and built-in sensors (camera, global positioning system, and accelerometer). With such connectivity, an Android application can control these hardware through universal serial bus (USB), Bluetooth, other wired or wireless connection (Matumo & Kisangiri, 2014; Syafeeq, Shazli, & Daud, 2015; Umare &



Padole, 2015). Furthermore, IOIO is unique as it does not require any firmware configuration.

Recently, the development of IOIO-On-The-Go (IOIO-OTG), which is more versatile than previous IOIO boards, helps leverage the USB On-The-Go specification by connecting a host or as a slave to an Android device with a USB On-The-Go (USB-OTG) cable and a software library (Java.jar file), enabling an Android application to manage all communications that significantly simplifies the coding (Umare & Padole, 2015).

Lately, many of today's portable devices have begun using USB electrical interface as a way of connecting to a PC. Although USB works well as a desktop interface, however it is not well suited for portable devices as when the PC acts as a host (master), all other devices will be reduced to peripherals (slave). This is inevitable because a host cannot connect to other hosts, and likewise a peripheral cannot connect to other peripherals. Otherwise, large currents will be required for such connection, and the USB connectors will be too large for portable devices (Remple, 2003).

The main difference between previous IOIO boards (DEV-11343 ROHS) and IOIO-OTG (DEV-13613 ROHS) is the ability of the latter to leverage the USB-OTG specification to serve as a host or an accessory. Figure 2.9 shows the latest IOIO board. A switch on the board can be used to switch the roles between a host and an accessory, but most of the time the board can be set on auto mode, allowing it to detect its role in the connection (Syafeeq et al., 2015). The following are the features of the latest IOIO board:





1. USB Host (Master): When the application is running on an Android device, the IOIO-OTG will run on its own power source.
2. USB Accessory (Slave): When the application is running on a Windows, Linux, or OSX machine, the IOIO-OTG will run on the device mode and present itself as a virtual serial port.

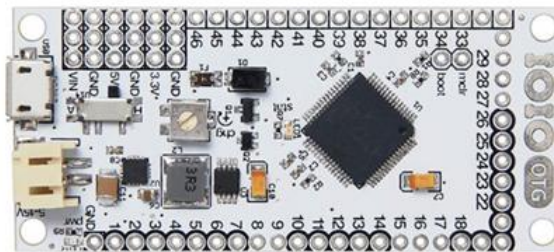


Figure 2.9. IOIO Board

Essentially, the heart of the IOIO board is a microcontroller in which codes are stored in flash memory by a process called microcontroller programming. Moreover, IOIO can interact with peripheral devices in the same way as most microcontroller units (MCUs) do. Additionally, digital Input/Output, pulse width modulation (PWM), Analog Input, inter-integrated circuit (I2C), serial peripheral interface (SPI), and universal asynchronous receiver/transmitter (UART) control can all be used with the IOIO (Umare & Padole, 2015).

The IOIO firmware code comprises two main parts, namely bootloader and application. The bootloader is the first code that runs every time the IOIO is restarted. It



establishes data connection with the Android device, and then uses the data connection to check for the existence of new application codes on the Android device. Running the IOIOLib codes in the application firmware, IOIO will be able to communicate with Android devices by sending appropriate commands (Syafeeq et al., 2015).

### 2.5.1 USB OTG

USB 1.1, USB 2.0, USB OTG, Wireless USB, and OTG are technical specifications used in the communication industry. Initially, the original USB 1.0 specification was released in January 1996, and in April 2000, it underwent a major revision resulting in a new revised specification called USB 2.0. Later, in July 2003, USB OTG addendum was released, defining a new class of devices for portable, battery-powered products with limited host capabilities. Finally, in May 2005, Wireless USB (WUSB) specification was released (Harmon, 2007).

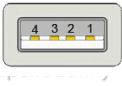

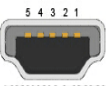


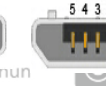

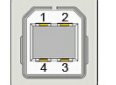
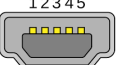


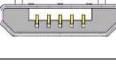
USB OTG is an add-on to the USB 2.0 specification that defines a new class of devices that extends the functionality of a peripheral product to include limited host capabilities. As the name implies, the original target of the specification was consumer portable devices with which end users could share data when a computer was not available (Harmon, 2007).

Various USB connectors that can be mounted on a USB host or on a USB peripheral called “receptacle”, and the USB connectors can be attached to cables using a



“plug” as shown in Table 2.5 shows. A non-USB OTG device can always act as a USB host or an A-device with an A-receptacle, to which only a Type-A plug can be inserted. Similarly, a non-USB OTG of B-device can always act as a USB host with a B-receptacle, to which only a Micro-B plug can be inserted. Furthermore, a USB OTG device can act as either a USB host (A device) or a USB peripheral (B-device), with both having a Micro-AB receptacle to which a Micro-A plug and Micro-B plug can be inserted, respectively (Lokhande, Bamnote, Ingle, & Dharkar, 2014).

Table 2.5  
USB Connector

		Plug					
							
Receptacle		USB 1.x/2.0 Standard Type A	USB 1.x/2.0 Standard Type B	USB 1.x/2.0 Mini A	USB 1.x/2.0 Mini B	USB 2.0 Micro A	USB 2.0 Micro B
	Type A	YES					
	Type B		YES				
	Mini A			YES			
	Mini B				YES		
	Micro AB					YES	YES
	Micro B						YES

Adaptation from (Lokhande et al., 2014)

Table 2.6 shows the descriptions of the USB pin of standard USB and Mini/Micro USB. Clearly, the two types of USB pins differ in the number of pins, with the standard and Mini/Micro USB consisting of 4 and 5 pins, respectively. In addition, Min/Micro USB has an additional identification function, providing it with OTG capabilities to swap two functions depending on its application (Lokhande et al., 2014).

Table 2.6

Description of USB Pins

Pin		Name	Description
Standard	Mini/Micro		
1	1	VCC	+5 V
2	2	D -	Data -
3	3	D +	Data +
			OTG Identification:
		ID	<ul style="list-style-type: none"> <li>• <b>Host:</b> connected to ground</li> <li>• <b>Slave:</b> not connected Ground</li> </ul>
4	5	GND	
	Shell	Shield	

Adaptation from (Lokhande et al., 2014)

Moreover, cable orientation determines the role of each OTG device as either a host or a peripheral at the connection, the role of which can be reversed via a dynamic switching method called Host Negotiation Protocol (HNP). In this respect, a USB OTG cable has the ability to switch such roles. Figure 2.10 shows the switching role of USB OTG cable orientation, with the red square indicating the USB OTG. Given that the Micro AB plug can fit both Micro A and Micro B receptacles, A device and B device can therefore function as either a host or a slave, depending on the receptacle being connected to each one. Figure 2.10 shows an A device and a B device functioning as a host and a

slave, respectively. Figure 2.11 shows a B device and an A device functioning as a host and a slave, respectively. As acknowledged, USB OTG cable can help eliminate the steps of the must pass of each device and extend the functionality of smart phones, rendering such devices more adaptable to mobile computing (Harmon, 2007; Remple, 2003).

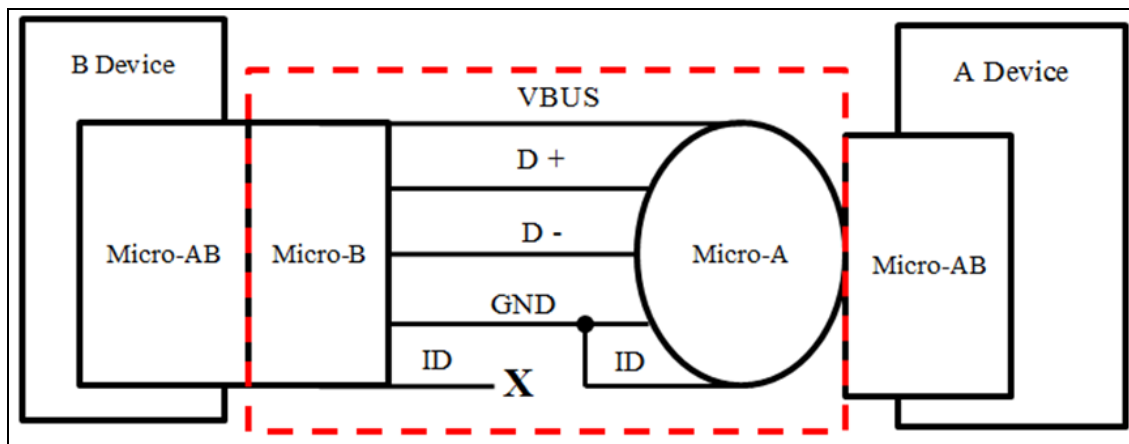


Figure 2.10. USB OTG Cable Orientation of an A Device and a B Device Acting as a Host A Slave, Respectively (Harmon, 2007)

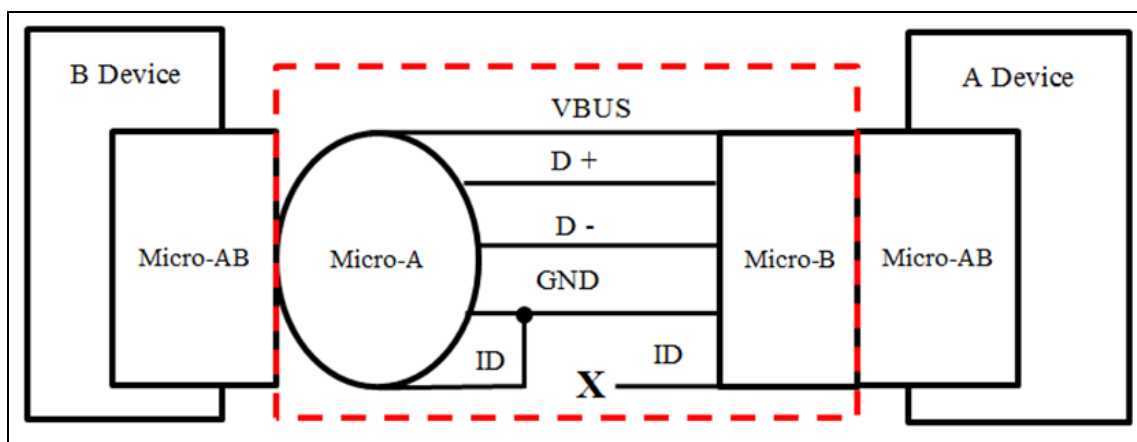


Figure 2.11. USB OTG Cable Orientation of a B Device and an A Device Acting as a Host A Slave, Respectively (Harmon, 2007)



## 2.6 Mobile Device Operating System

In the early years of mobile computing, mobile phones were only capable of sending and receiving text messages and calls for communication purposes. As anticipated, mobile technology has improved, and will continue to improve, with each improvement making mobile devices more powerful, affordable, and appealing. For example, not only their screens are getting bigger and better in quality but also their internal hardware is getting more complex and powerful, effectively making them a formidable rival to laptops and computers (Padhya, Desai, Pawade, & Student, 2007). Thus, it is hardly surprising that mobile phone has become an integral part of the people's life. To date, the use of mobile devices has become pervasive and almost ubiquitous, manifesting in various brands and running on several operating systems (Joseph, Professor, Kurian, Manager, & Mahindra, 2013).

Obviously, the mobile operating system is the heart of smart phones, and currently there are eight operating systems used, namely iOS, Windows Phone, Blackberry, Android, Tizen, Sailfish OS, Firefox OS, and Ubuntu Touch. In fact, these mobile operating systems are reliable and powerful, comparable to those of computer operating systems. Essentially, the mobile operating system performs activities similar to those of a computer operating system, such as office production activities using outlook and communicator. Among the eight mobile operating systems, Apple's iOS, Google Android, Windows Mobile, Symbian and Blackberry remain the most popular and dominant systems used for mobile devices, such as smart phones and tablets in Malaysia





(Statista, 2018). In this information-driven era, they will continue to exert their influence over a long period as evidenced by their continual phenomenal growths and potentials (Padhya et al., 2007).

An OS is the most critical software element on any running processor-based device. The OS manages the hardware and software resources within a device, performs and manages basic tasks such as the recognition of input from the device keyboard and generation of output to the device's screen and ensures different programs running at the same time do not interfere with each other. Hence, it is responsible for the management of memory and for communication within the device (Jindal & Munjal, 2012).



### 2.6.1 Symbian

In 2008, Symbian Ltd developed an open source operating system and licensed it to several phone manufacturers, notably Nokia. Primarily, Symbian OS is designed to run on minimum power and on low memory. Furthermore, it is a multitasking operating system, and it has less dependency on peripherals (Jindal & Munjal, 2012). In compliance with the agreed-upon standards of Symbian OS, applications running on various technologies would be robust, portable, and interoperable. Moreover, memory management is optimized for embedded software environment, and application support for international environment is enabled by the built-in Unicode character sets. Symbian





uses microkernel approach to help manage system resources, such as memory, and carry out time-slicing of applications and system tasks (Joseph et al., 2013).

Essentially, Symbian OS is built upon a common architecture consisting of a nanokernel/microkernel core with basic localization and screen drivers. Base services sit above the kernel, including low-level libraries, media frameworks, XML, file system management, and hardware abstraction. OS services provide communication, telephony, networking, multimedia, and graphics.

The above elements support the Application Services layer with application-facing APIs for development and an interface layer to manage the user interface (UI). A Java Virtual Machine (JVM) is also included above the OS services layer. Nokia provides Software Development Kits (SDKs) for Symbian development that supports a variety of languages, including C++ and Java (Okediran O, Arulogun O, & Ganiyu R, 2014). Hence, such built-in multi-language support helps makes Symbian a more advantageous compared to other OSs (Kumar Maji, Hao, Sultana, & Bagchi, 2010).

### 2.6.1.1 Symbian OS Architecture

In mobile computing, complex OSs will usually contain many important elements, such as UI elements, to make mobile devices more robust. For example, Symbian OS integrates the functionality of three different UI options. In essence, the Symbian's







system model is segmented into 3 main layers as shown in Figure 2.12 (Jindal & Munjal, 2012; Kumar Maji et al., 2010; L. Li, 2007) as follows:

*OS Layer:* This layer includes the Hardware Adaptation Layer (HAL) that abstracts all higher layers from actual hardware and Kernel, including physical and logical device drivers (Kumar Maji et al., 2010). It also provides programmable interface for hardware and OS (through frameworks, libraries, and utilities) and higher level OS services for communications, networking, graphics and multimedia.

*Middleware Layer:* This layer provides services (independent of hardware, applications, or user interface) for applications and other higher-level programs. Services can be application-specific, such as messaging and multimedia, or generic to a particular device, such as web services, security, device management, and Internet Protocol (IP) services.

*Application Layer:* This layer contains all the Symbian applications, such as multimedia applications, telephony, and IP applications. In fact, Symbian OS is specially designed for data-enabled mobile phones. According to Symbian Limited, such design was needed as scaling down PC operating systems or expanding existing light-weight operating systems would lead to too many fundamental compromises. Hence, Symbian OS is designed to fulfil several special requirements of mobile phones as follows:

- Devices are small



- The target is for consumer mass market
- Devices can be used when connected to the wireless network and other devices
- Manufacturers must be able to use it on diverse range of products
- Hardware designs, user interfaces, and networks.
- It must be open for the third party development of additional applications and services
- Handling user data must be reliable
- Communication in situations lacking resources
- All device resources, especially power consumption and memory, must be used efficiently

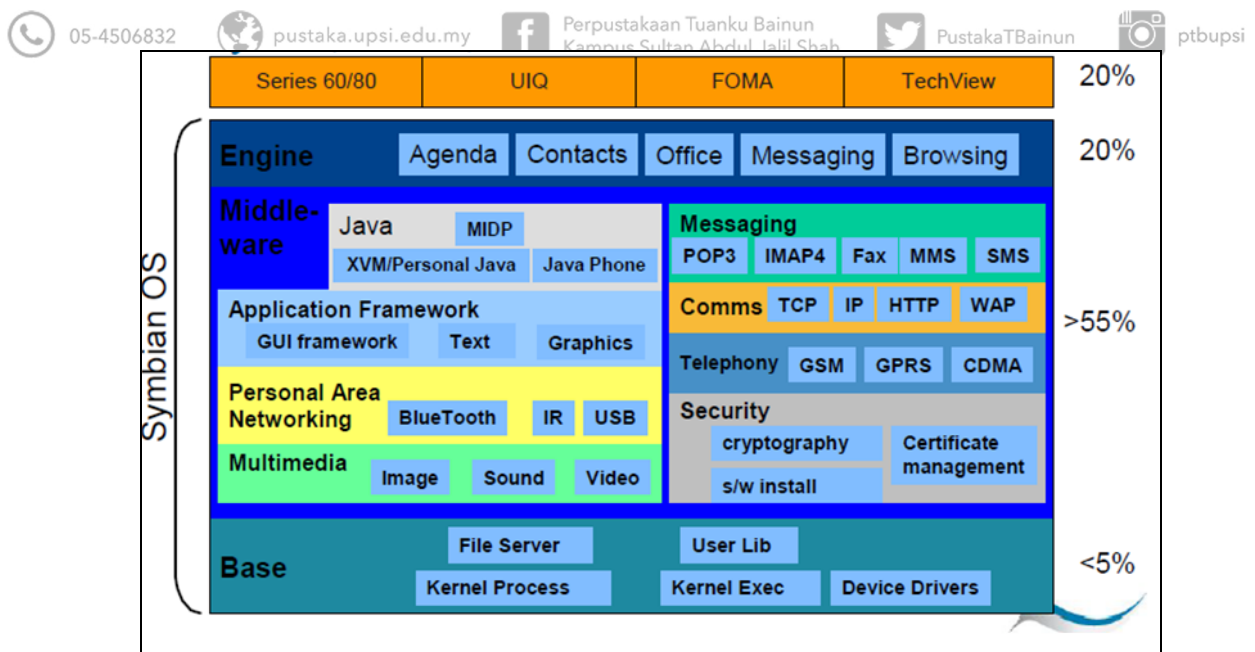


Figure 2.12. Symbian OS Architecture (L. Li, 2007)



## 2.6.2 BlackBerry

BlackBerry is a mobile operating system developed by BlackBerry Ltd, which was initially released on 30<sup>th</sup> January, 2013. Unlike Symbian, BlackBerry is a closed source or a proprietary operating system. It is programmed in C, C++ and Java programming languages (Padhya et al., 2007). The BlackBerry OS is the proprietary mobile operating system developed by RIM (Research in Motion), exclusively for its BlackBerry smart phones and mobile devices. It offers native support for corporate mail via Mobile Information Device Profile (MIDP), which enables effortless wireless synchronisation with Microsoft Exchange, Lotus Domino and email, contacts, calendar, notes and others, through the BlackBerry Enterprise Server. This OS additionally supports Wireless Application Protocol (WAP) 1.2. Uniquely, its network architecture is different from other operating systems (Joseph et al., 2013).

BlackBerry provides end-to-end encryption, using two encryption options such as Advanced Encryption Standard (AES) and Triple Data Encryption Standard (Triple DES). Data sent to the BlackBerry smart phone are encrypted by BlackBerry Enterprise Server using the private key retrieved from the user's mailbox. The encrypted information travels securely through the network to the smart phone where it will be decrypted. Additional authorization is also available when users access the application data or corporate intranets (Joseph et al., 2013). However, the use of BlackBerry 10 has increased slightly in some regions, but its global growth has gradually decreased, dropping from 0.5% to 0.3% to 0% (Padhya et al., 2007).



### 2.6.2.1 BlackBerry OS Architecture

Figure 2.13 shows the core components of the architecture of BlackBerry 10 device.

1. CPU embedded bootloader: The CPU bootloader verifies the digital signature of the bootloader code before running it.
2. Bootloader: The bootloader verifies the digital signature of the OS before running it.
3. Microkernel: The microkernel is the minimal amount of software that the OS requires to run.
4. Radio: The radio includes the drivers, stacks, and services required to support the radio subsystems for voice, data, and other services.
5. Drivers and BSP: The drivers and BSP include the drivers and board ring up logic to support the device hardware.
6. OS: The OS processes that exist outside of the kernel.
7. Platform and application services: Platform and application services include security management, software installation and management, background services for applications, media services, and more. Platform and application services are required because applications will not be able to run services in the background or to gain access to protected system components and services.
8. Application runtimes: Runtimes include virtual machines, libraries, services, mapping layers, and more, with all applications run in isolated

sandboxes. BlackBerry10 devices support applications built with the native SDK, Android, and HTML5.

9. App: The applications can be preloaded, user-installed, or deployed by users' organization.

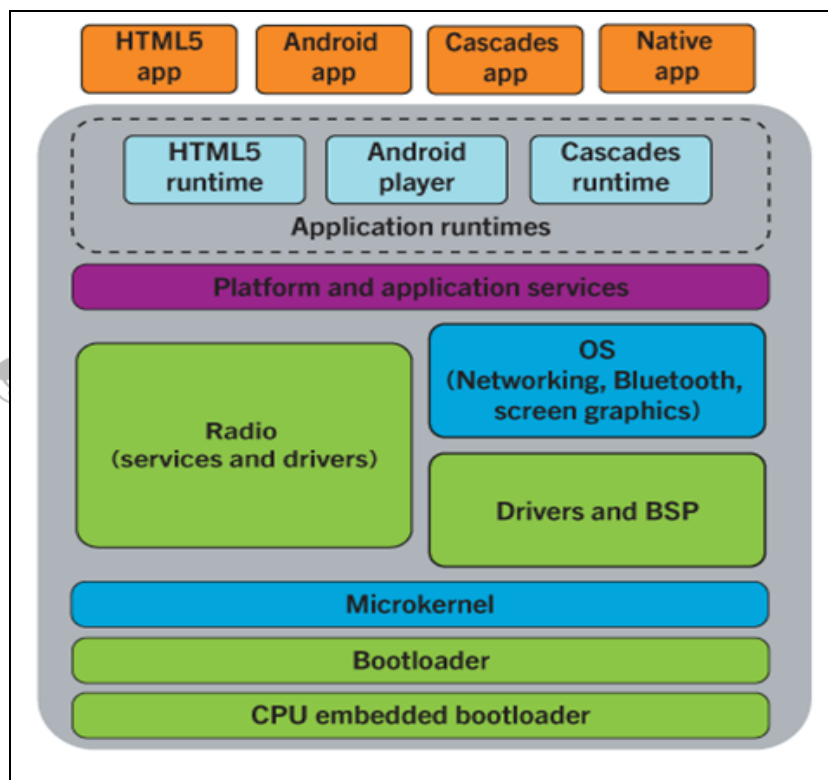


Figure 2.13. BlackBerry 10 Device Architecture (BlackBerry, n.d.)

### 2.6.3 Windows Phone

Windows is a mobile operating system developed by Microsoft Corporation and designed primarily for touch screen devices, such as smart phones and tablets. Initially, it was released on 8<sup>th</sup> November 2010, and its latest OS version released on 20<sup>th</sup> November 2015 is known as Windows (Padhya et al., 2007). Currently, Windows is the most popular computer operating system. In the last five years, Microsoft Corporation has focused its efforts in the development of mobile operating systems (Joseph et al., 2013).

Arguably, today's Windows phones has the world's best UI, allowing users to start an application with just a slight touch of their finger. Previously, such ease of use was not possible, given that earlier versions of Windows phone OS were not compatible with relatively large screens of Phablets and Tablets. Now, Nokia is working aggressively to develop 6-inch Phablets, which will run on newer Microsoft OS. Currently, some of the best Windows smart phones include Nokia Lumia 1020, Nokia Lumia 928, and HTC Mobile Radar (Divyap & Venkata Krishnakumar, 2016).

Interestingly, Windows mobile offers a new user interface with 'Metro' design. For example, Windows CE (Compact Edition) was specifically designed for handheld devices using Windows Application Program Interface (API). Later, in June 2012, the company introduced newer versions of Windows 7 and Windows 8 mobile OS that support many enhanced features, such as multi-core processor support, hi-fi screen resolution, and larger storage support (Joseph et al., 2013).



### 2.6.3.1 Windows Phone Architecture

Windows Phone 7's architecture requires a hardware layer that meets Microsoft's minimum system requirements, such as ARM7 CPU, a DirectX 9-capable GPU, 256MB RAM and 8GB of flash memory, a 5-megapixel camera, a multi-touch capacitive display, an A-GPS, an accelerometer, a compass, proximity and light sensors, and six physical buttons (for back, start, search, camera, power/sleep, and volume). The Windows Phone kernel handles low-level device driver, including basic security, networking, and storage. It has three libraries (for an App Model for application management, a UI model for user-interface management, and a Cloud Integration module for web search via Bing), location services, and push notifications embedded into the J4 kernel. Application-facing APIs include Silverlight, XNA, HTML/JavaScript and the Common Language Runtime (CLR) that supports C# or VB .Net applications.

The kernel itself is a proprietary Windows OS designed for embedded devices that combine Windows Embedded CE 6.0 R3 and Windows Embedded Compact 77. Moreover, Windows Phone 8 has replaced the Windows CE kernel with newer kernel based on Windows NT. In part, such replacement helps the former OS to mimic Windows 8 desktop OS, allowing easier porting of applications between the two operating systems. Figure 2.14 depicts the architecture of Windows phone (Okediran O et al., 2014).



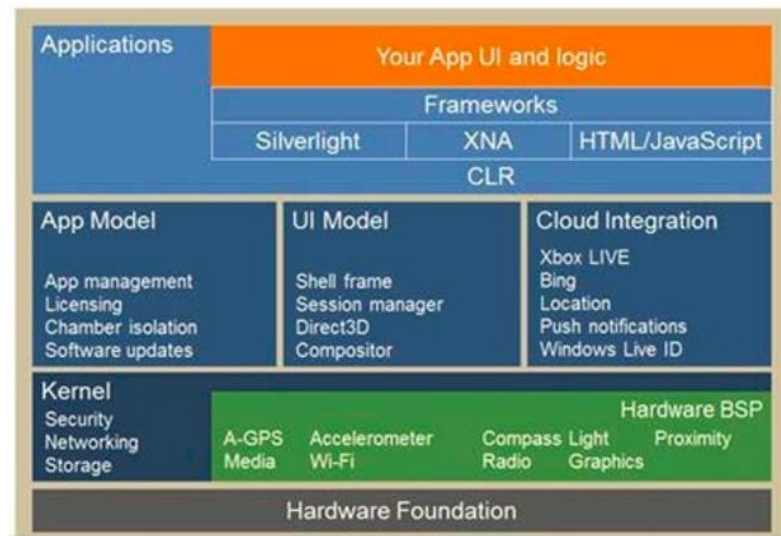


Figure 2.14. Microsoft's Windows Phone Architecture (Okediran O et al., 2014)

iOS is a powerful, expensive mobile operating system developed by Apple Inc. and is designed primarily for touch screen devices, such as smart phones and tablets, whose native language is C. iOS was initially released on 29th June, 2007, and the latest OS version known as iOS 9.1 was launched on 8th December, 2015 (Padhya et al., 2007). Currently, iOS is one of the leading mobile operating systems that is highly stable, secure, and user friendly (Divyap & Venkata Krishnakumar, 2016; Joseph et al., 2013). In addition, iOS works with Microsoft Exchange and standards-based servers to deliver over-the-air push emails, calendars, and contacts. More importantly, it protects users' data by encrypting information in three separate areas, namely in transmission, at rest on the device, and when backed up to iTunes (Joseph et al., 2013).



### 2.6.4.1 iOS Architecture

iOS was derived from Mac OS X, and thus shares the basic Darwin foundation, which is an open source POSIX-compliant UNIX OS. As such, iOS can be considered a variant of UNIX consisting of four abstraction layers, such as Core OS, Core Services, Media, and Cocoa Touch6 (Okediran O et al., 2014) as depicted in Figure 2.15.

- i. *Core OS*: This layer contains the kernel of the operating system, which includes basic low-level features: system support—threads, sockets, IO, DNS, math, memory—general security services—certificates, private/public keys, and encryption—external hardware management, Bluetooth, and sound and image processing.
- ii. *Core Services*: This layer provides fundamental system-services, which are subdivided into different frameworks based on C and Objective C. It comprises basic application services, including accounts, contacts, networking, data management, location, calendar events, store purchasing, SQLite, and XML support.
- iii. *Media Layer*: This layer contains several high-level frameworks to support 2D and 3D graphics, audio and video technologies.

- iv. *Cocoa Touch*: This layer consists of UIKIT, which is an Objective C-based framework that provides a number of functionalities that is necessary for the development of an iOS Application, such as the User Interface Management. It also includes APIs for building applications that support multitasking processes, touch input, notifications, interface views, and access to device data.

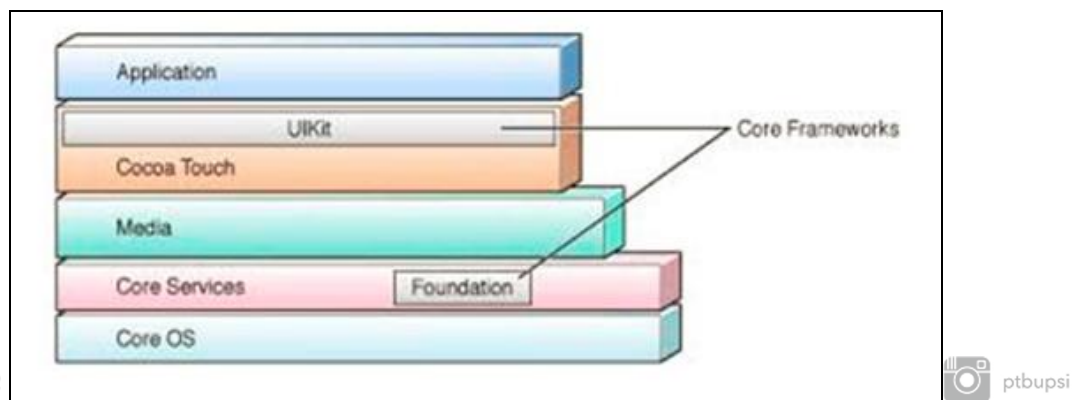


Figure 2.15. The Architecture of Apple's iOS (Okediran O et al., 2014)

### 2.6.5 Android

According to (Wang, Duan, Ma, & Wang, 2011), Android is a mobile operating system developed by Android Inc. (which was the acquired by Google and later by the Open Handset Alliance) running on the Linux kernel. Essentially, it is a software stack for mobile devices that includes an operating system, middleware, and applications.



Android, Inc. was founded on October, 2003, in Palo Alto, California, United States by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development WebTV) (Bazard & Bhardwaj, 2011). Since its original release, Android has undergone a series of updates, typically focusing on fixing bugs as well as adding new features. Generally, each new version of the Android operating system is developed under a code name based on a dessert item.

### 2.6.5.1 Android Architecture



Figure 2.16 shows the architecture of Android operating system consisting of four layers, namely Application, Application framework, Hardware Abstraction Layer, Libraries and Android RunTime, and Linux Kernel described as follows:

#### 1. Linux Kernal

It acts as the heart of the whole system that provides various functionalities, such as memory management, process management, device management, security settings in android system, and the entire essential device driver for the hardware with which it interacts (Bala, Sharma, & Kaur, 2015).



## 2. Android Runtime

It includes a set of core libraries that provides most of the functionalities available in the core libraries of the Java programming language. Every android application runs on its own process with Dalvik Virtual machine (Gandhewar & Sheikh, 2010).

## 3. Application Framework

A software framework that is used to implement a standard structure of an application for a specific operating system. This framework can reassemble functions used by other existing applications with the help of managers, content providers, and other services programmers, (Gandhewar & Sheikh, 2010).

## 4. Android Application

It comprises both the native applications and third party applications. Native applications support the basic Android implementation, such as SMS client application, Dialer, Web browser, and Contact manager. The third party applications are largely installed by developers and programmers, the debugging or testing of which are performed by users (Bala et al., 2015).

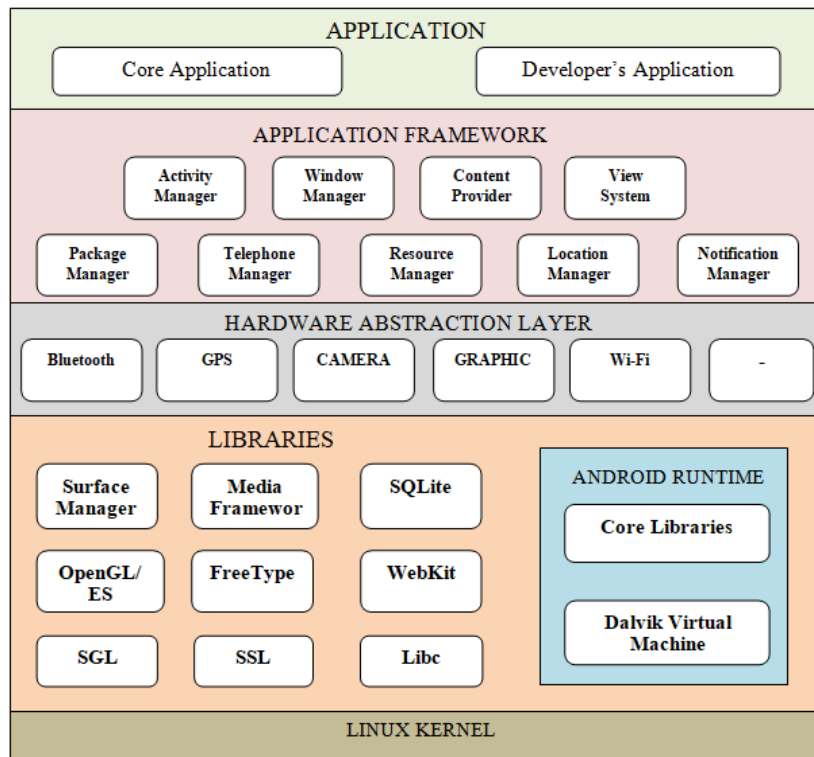


Figure 2.16. Android Architecture (Gandhewar & Sheikh, 2010; Wang et al., 2011)

## 2.6.6 Comparison of Operating System

The comparative analysis and market share analysis carried out in the fourth quarter of 2014 showed Android and Windows Phones to be superior compared to other OSs. Currently, Android is the best Smartphone OS in the world today, and lately it has been used as an educational tool. Being an open source operating system, Android can freely and readily help users to install third party applications. Given such unconstrained installation, the system is susceptible to cyber threats or malware attacks, such as virus,



worms, spyware, adware, and Trojan horse. As a workaround, experts insist on detecting malware before installing an application.

From the historical perspective, mobile devices has added many of the core components of a regular phone system, such as Central Processing Unit (CPU), memory, and Liquid Crystal Display(LCD) screen, and telephone switching, to its design as early in the 1990s (Hall & Anderson, 2009), The first addition was to store and access contacts. Further improvements in mobile phone technology were carried out with the introduction of virtual applications in the early part of this century, which was greatly driven by the gadget convergence movement. Figure 2.17 shows the percentage of market share of various mobile operating systems, clearly indicating that Android had the highest market share at 81.3%, followed by Apple iOS and Microsoft Windows Phone at 13.4% and 4.1%, respectively. At 1% and 0.2%, Blackberry 1 and other OSs had the least market share, respectively.



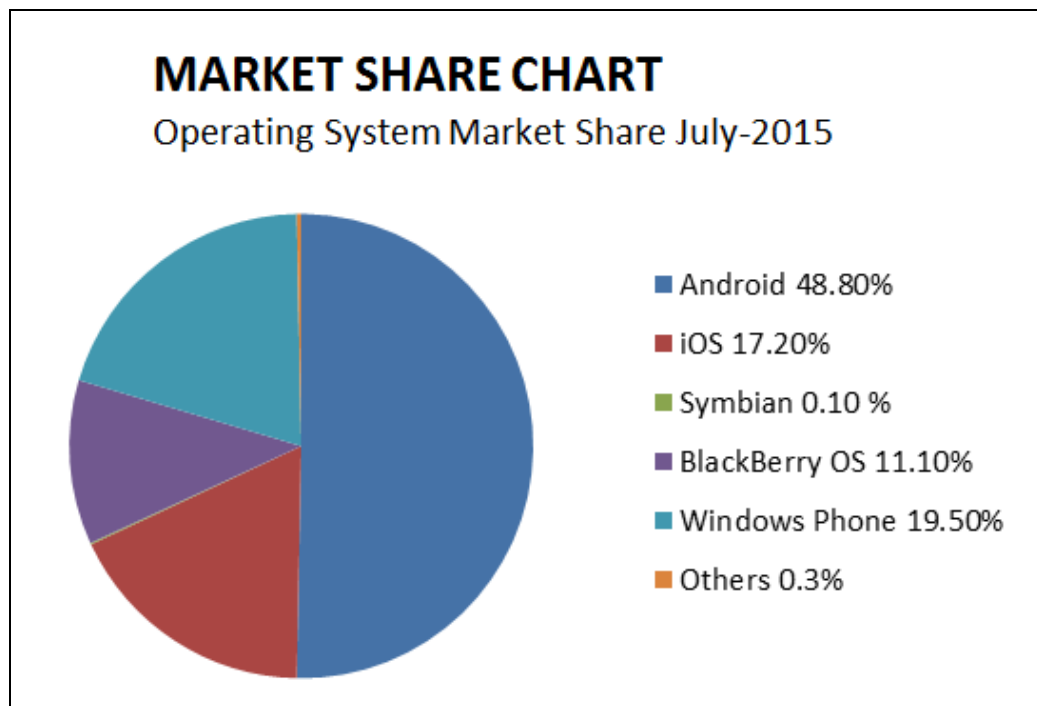


Figure 2.17. The Market Share of Mobile Operating Systems (Joseph et al., 2013)

Table 2.7 shows the comparison of smart phone operating systems of Android, iOS, Symbian, Blackberry and Windows Phone. Evidently, Android dominated the smart phone market with almost half of the share at 48.8%. Such finding is hardly surprising given that it is an open-source operating system that is free. Moreover, Android developers have developed many applications with special functions, which further strengthen its dominance in the market (Hall & Anderson, 2009). Interestingly, Android allows developers to write application codes in Java language that control mobile devices using Google-developed Java libraries.

Table 2.7

## The Comparison of Mobile Operating Systems

Operating System	Android	iOS	Symbian	Blackberry	Windows Phone
<b>Runs on</b>	Smart phones, Tablet, Computer, TVs, Cars and wearable devices	iPhone, iPad, iPod Touch	Smart phones	Smart phones	Zune software (not since windows 8)
<b>Source Model</b>	Open source and in most devices with proprietary components	Closed Source	Closed source, previously open source	Closed source	Closed source
<b>SDK Platform</b>	Windows XP, Vista and 7; Linux, Mac OS X	Mas OS X Snow Leopard 10.6.4	Windows XP Professional SP2; Vista & 7 for some SDKs	32-bit Windows XP, Vista and 7	Windows Vista & 7
<b>OS Family</b>	Linux	Darwin	RTOS	QNX	Windows CE 7 Windows NT 8
<b>Market Size</b>	Very High	High	Very low	Low	Medium
<b>Virtual Machine</b>	Allowed	Not allowed	Allowed	Allowed	Allowed
<b>Application store</b>	Google Play (Android Market)	App store	Nokia Ovi Store	BlackBerry app world	Windows Market Place

Adaptation from (Divyap & Venkata Krishnakumar, 2016; Okediran O et al., 2014; Padhya et al., 2007)



One of the most important criteria for the development of the proposed system was that Android allows the use of virtual machine, which is an open-source model used in most devices with proprietary components. In fact, Android has been designed to serve as a modern mobile platform that is truly open, on which innovative Android applications can be developed using advanced hardware and software, thus bringing in good value to customers.

## 2.7 Warehouse Management Process in Manufacturing Industry

According to Mika (2014), warehouse application is one of the earliest ways of employing RFID technology in manufacturing. Fundamentally, warehouse management includes all kind of operations related to manufacturing logistics, such as inventory tracking, identification of components, control of materials flows, and management of picking, receiving and shipping of materials. As stressed by Richards(2014), a warehouse should be viewed as a temporary place to store inventory and as a buffer in supply chains, with the primary aim to facilitate the movement of goods from suppliers to customers and to match product availability with consumer demand.

Figure 2.18 show the process of warehouse environment and management. Of late, supply chains have to rely on a number of technologies to achieve a higher level of performance in satisfying consumer needs. The warehouse is divided into specified areas according to technological operations in order to automate inbound, outbound and in-

stock operations. After receiving goods, the goods are put-away according to its shelf to make it easier for the workers when taking the goods for packing. After the goods are done packing, then it is ready for shipping.



Figure 2.18. The Warehouse Management Process

Figure 2.19 shows an example of the flow of materials and information through a factory. Obviously, materials flow through many stages of manufacturing, from procurement to finished goods, with inventories being prepared for shipment. The material flow is essentially a process of tracking the journey that materials take, right from the time they arrive at a manufacturing site to the point where customers receive the

finished product or item. In today's manufacturing, materials flow is an important concept that is an integral part of advanced modelling of supply chain management.

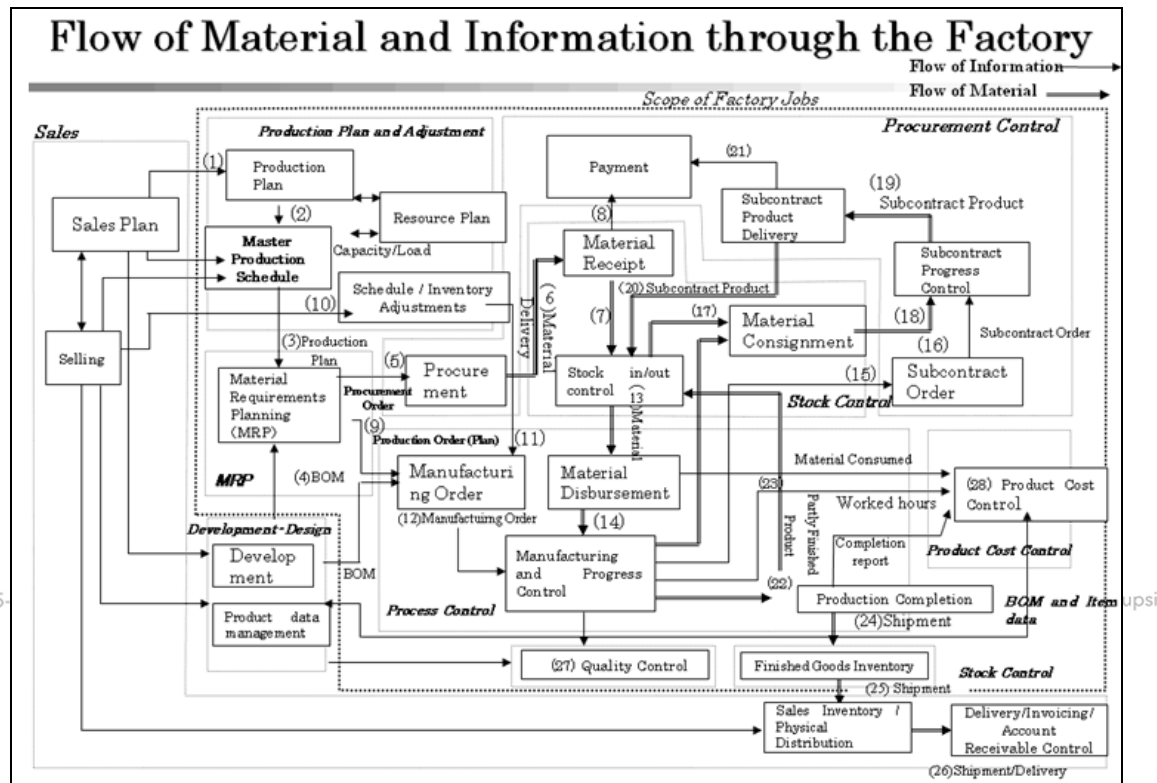


Figure 2.19. The Flows of Materials and Information Through a factory (Matsubayashi & Hiroshi, 2003)

As acknowledged by many, industrial material flow can easily become very complex, entailing detailed record of materials passing through various stages of the manufacturing process. As such, data need to be current and accurate, enabling manufacturing personnel to make informed, correct decisions based on the latest version of information. More importantly, such data should be in such a form that it can be used efficiently and effectively at the right time and place. In fact, production monitoring

requires efficient collection and distribution of data of real-time of events to the shop floor.

In this regard, traceability is the ability to verify the history, location, or application of an item by means of documented or recorded identification using identification tags (such as RFID tags and barcode). Technologies that can be applied in traceability solutions include RFID (Radio Frequency Identification), Barcode, and GPS. Effectively, traceability can be applied in various industries to track and trace products or raw materials from point-to-point across the supply chain. By implementing traceability solutions, companies can achieve better transparency, improved efficiency, and enhanced security. In general, the most common areas in which traceability solutions are applied include logistics, supply chain, and food processing. Figure 2.20 shows the basic flow of materials. Initially raw materials are being purchased. The receiving raw material will be stored in warehouse of the raw materials before being process. Production will process the raw materials into finished goods. Quality assurance will ensure the finished goods have no mistakes and defects. Before shipped the finished goods, the finished goods are stored in the warehouse of the finished goods.

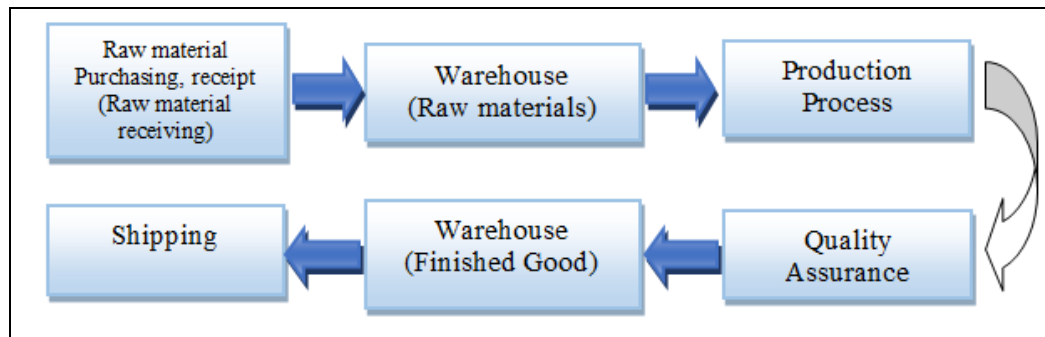


Figure 2.20. Flowchart for the Basic Material Flow

According to Attaran (2012), the application of RFID in the supply chain management process would help manufacturing companies to reap ten benefits as summarized in Table 2.8.

Table 2.8

*Supply Chain Management Processes and RFID Benefits*

Supply Chain Management Processes	RFID Benefits/ Success Variables
I. Demand Management	<ul style="list-style-type: none"> <li>• Fast and accurate information retrieval</li> <li>• Better decisions</li> </ul>
II. Order Fulfilment	<ul style="list-style-type: none"> <li>• Enhanced visibility along the supply chain</li> <li>• Better-quality information</li> </ul>
III. Manufacturing Flow	<ul style="list-style-type: none"> <li>• Accurate asset tracking</li> <li>• Enhanced process automation</li> </ul>
IV. Reverse Logistics	<ul style="list-style-type: none"> <li>• Improved productivity</li> <li>• Improved quality and reliability</li> </ul>
V. Supplier Relationship Management	<ul style="list-style-type: none"> <li>• Reduced operating costs</li> <li>• Improved competitive position</li> </ul>

Adaptation from Attaran, 2012.



## 2.8 Research Gaps

As discussed, embedded active RFID and wireless mesh network are two distinct wireless technologies that can help improve manufacturing environment. In particular, such embedded system can help manufacturing personnel in detecting the movement of secure containers and in managing asset management, inventory tracking, warehouse security management, and others. With RFID technology, manufacturing companies can gain improved accuracy, enhanced work-in-process traceability, better tracking and control of assets, reduced corrective costs, and minimal human intervention. Based on the literature review, many of industrial applications of RFID technology are largely focused on product tracking systems. In contrast, the proposed system was based on the integration of RFID technology, Zigbee, and WSM that resulted in an embedded system called WIBRED, with enhanced tracking and locating capabilities. With such capabilities, the proposed system can be applied in various application areas, notably in manufacturing warehouse in which real-time mobile tracking to search and locate missing or misplaced manufacturing objects or parts in a plant can be carried out remotely and accurately. Specifically, WIBRED can function as an automatic tracking system to help improve the materials flow process, by tracking and monitoring movements of production components, parts, or assemblies in real time.

Many R&D members now interested to investigate in mobile RFID performance. The Zigbee technology integrated with mobile communication system and embedded passive and active RFID gave birth to mobile RFID to provide services to users in





manufacturing industry. The purpose of this work is to design new mobile RFID technology namely WIBRED works on WMN platform for warehouse monitoring purpose. Furthermore this work also to design automatic identification and tracking system for material flow process based on the problem occurs. Then, the automatic tracking system will be validated either the system will be functional or otherwise. Additionally this invention enhances the capabilities of RFID adopted through WSN platform. These materials are tracked in the real time movement and be analysed on the efficiency of the automatic system.





## CHAPTER 3

### DESIGN AND DEVELOPMENT



#### 3.1 Overview

The proposed system has two parts, namely the hardware and the software. In this chapter, the researcher discusses the methods involved in the development of the hardware and software, including the main components used for such development.

Initially specifications based on the idea of the development and the main components were drafted. Essentially, the proposed system was designed to run on two modes, namely station mode and mobile mode. The specifications of the key components of the proposed system are listed in Table 3.1.





Table 3.1

Specifications of the Key Components	
Key Components	Specifications
Smart Phone	Android OS (Version 1.5 above)
IOIO -OTG	N/A
Passive Reader	NOVA
Relay Controller	N/A
Wireless Transceiver	XBee
PC	Windows Vista/7/8/10 (32-bit or 64-bit versions) Mac OS X v10.6 and higher versions (64-bit only)

Software development of the system consists of two parts. The first part involved the use of Java language of Android Studio Software to develop an Android application for temporary UART communication. The second part involved the development of a database using Tool Command Language (TCL) software.

### 3.2 Research Procedure

The waterfall model is a relatively linear sequential design approach for certain areas of engineering design. Waterfall model was proposed by Royce in 1970 which is a linear sequential software development life cycle (SDLC) model. This model is named “Waterfall” because its diagrammatic representation looks like a cascade of waterfall (Akshita Dubey, Amisha Jain, & Aditi Mantri, 2015; Saxena & Upadhyay, 2016). Waterfall model is best used when there is clear depiction of what the final product should be (Saxena & Upadhyay, 2016). Figure 3.1 shows the research methodology used

to develop the proposed system based on Waterfall model consisting of several stages as follows:

## 1. Requirement

In this phase, a literature review and an informal discussion with the industry expert were conducted to determine problems encountered by the industry in tracking objects in manufacturing plants. Guided by the above findings, the researchers formulated several problem statements, research goals, and objectives (see Chapter 1 for details). Discussions of research on existing products in the market and technology review have been discussed in Chapter 2.

## 2. Design

The related information gathered from the literature was used to determine the appropriate research methodology in designing the proposed system (the details of which are discussed in Chapter 3). In addition, the key components of the proposed system hardware had to be decided to help determine that the method of integration of relevant technologies. As for the software, the type of software developer suitable for the design of the mobile application and database was also required to be determined in view of the many types of software available.

### 3. Implementation

This development phase comprises two parts, namely hardware development and software development. The steps of both developments had to be determined to ensure smooth integration of such steps to help achieve the target objectives. Furthermore, the development of the proposed system was carried out with high degree of flexibility to allow any modifications to be made to the system.

### 4. Verification

The proposed system could be implemented such that it could run on mobile and fixed mode, depending on the intended requirements. Irrespective of the mode used, the system would be used tracking application.

### 5. Maintenance

In this phase, experimental testing of the proposed technology and prototype evaluation based on real world manufacturing process were performed. From the testing, the results were analysed to know measure the performance of the proposed system (see Chapter 4 for the details of the experimental setup and the discussion).

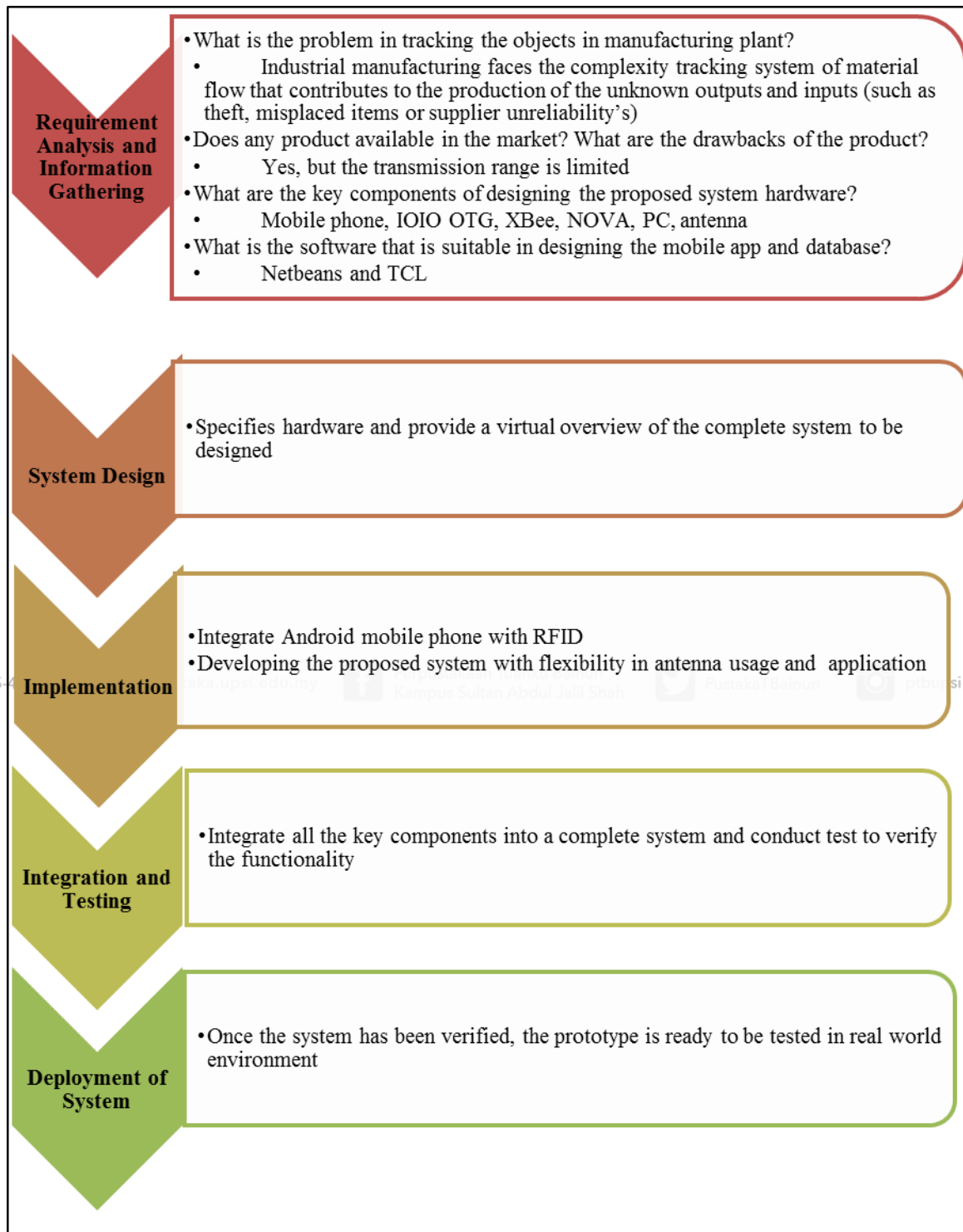


Figure 3.1. The Research Methodology of the Study Based on Waterfall Model

### 3.3 Hardware Development Design

In any system development, a feasibility study is essential to help the researcher to determine the correct components to purchase. Initially, an assembly of components of the breadboard needed to be determined to ensure the functionalities of the proposed system could be transferred to the Veroboard. Later, the performance of the hardware was verified through hardware debugging and hardware testing performed on the Veroboard to ensure the target objectives could be achieved. Figure 3.2 shows the process of the hardware development.

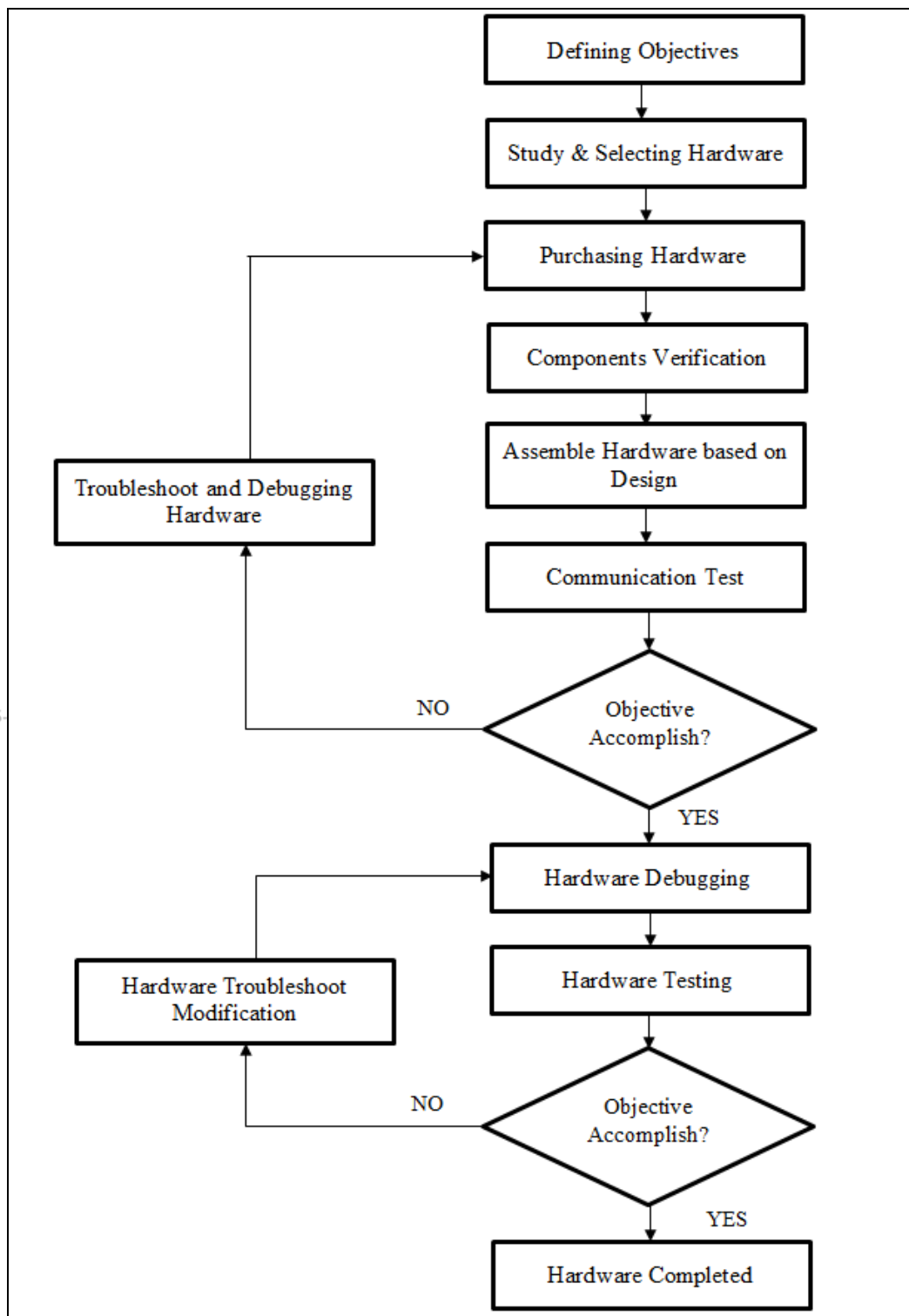


Figure 3.2. The Process of the Hardware Development

Figure 3.3 shows the differences in the attributes between the proposed system and the existing system (Vhatkar & Bhole, 2010). The existing system with the RFID Reader B-SL architecture functions by delivering internal location-based services for mobile devices using passive RFID technology. The combination of wireless and RFID technology enables an RFID reader to track and send information by PDAs or mobile devices via Wi-Fi.

A better solution to developing and implementing the proposed system would be based on Wireless Mesh Sensor (WMS) and Android mobile application. With such implementation, the proposed system could be efficiently used in manufacturing departments for traceability and monitoring purposes. To facilitate such use, the proposed system required data centralization, through which every department of the manufacturing company would be able to share information in real time.

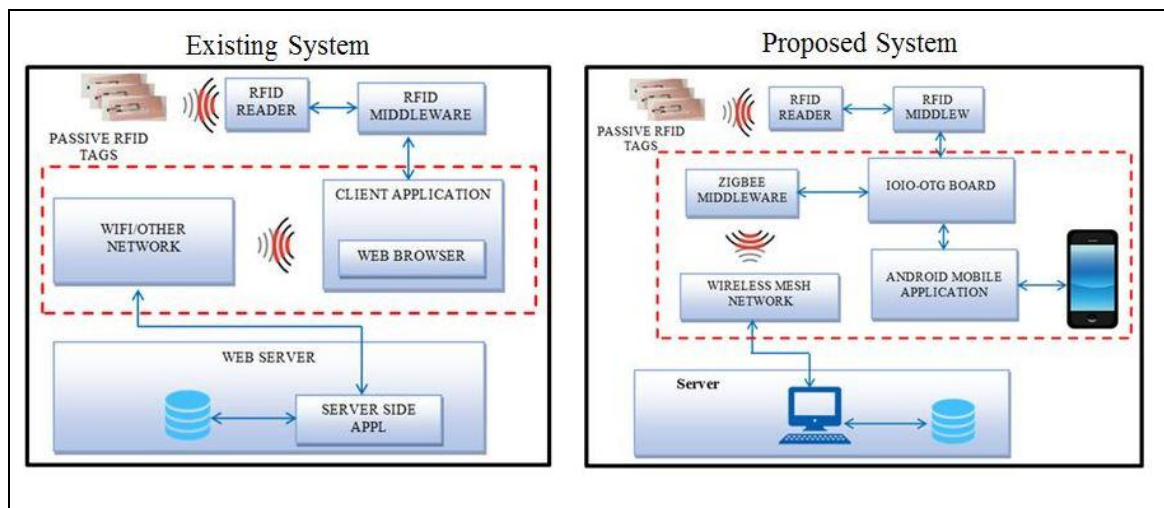


Figure 3.3. Comparison of Existing System and Proposed System

Figure 3.4 shows the block diagram of the architecture of the proposed system for smart data processing adopted for mobile tracking application. Arguably, designing a new embedded system for complex tracking of material flows would entail a new different approach that would focus on seeking and identifying manufacturing objects or parts search. Such process could be effectively carried out with the use of a common database, through which relevant data would be transmitted and received.

Essentially, the proposed system consists of three main parts, namely input, process, and output. The input and output of the proposed system are the signals from the passive tags and item description of the tagged object, respectively. In between the input and the output, lies the process part comprising smart phones, IOIO board, passive reader, relay controller, wireless transceiver, and database.

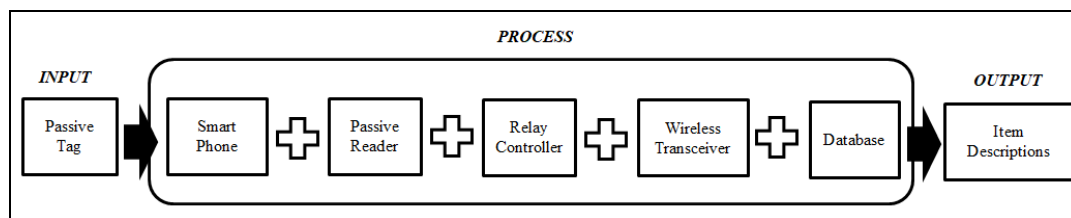


Figure 3.4. The Block Diagram of the Proposed System

The proposed system is powered by a 5 VDC power bank to IOIO board, with voltages ( $V_{IN}$ ) range between 5 VDC and 15 VDC (Ben-Tsvi, 2014). Therefore, the wireless transceiver is powered with 3.3 VDC, while the relay controller board and passive reader are powered by a 5 VDC power bank as depicted in Figure 3.5.



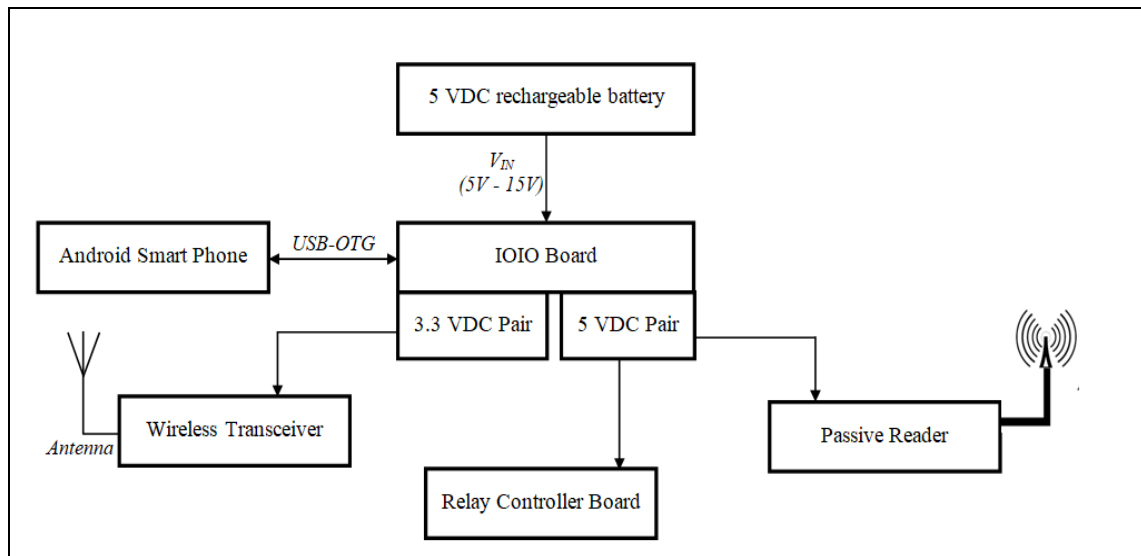


Figure 3.5. The Architectural of Proposed System Design

Figure 3.6 shows the block diagram of the development of the proposed system, including both the hardware and software. In developing the proposed system, five development phases were performed. Phase 1 included the communication between a smart phone and an IOIO-OTG board, while Phase 2 involved the communication among a smart phone, an IOIO-OTG board, and a passive reader. Phase 3 involved the communication between a passive reader and a wireless transceiver. In Phase 4, database setting and database scripting were carried out, and the final phase saw the combination of all the key components to ensure successful communication of the proposed system. The details of these phases are discussed in the following subsections.

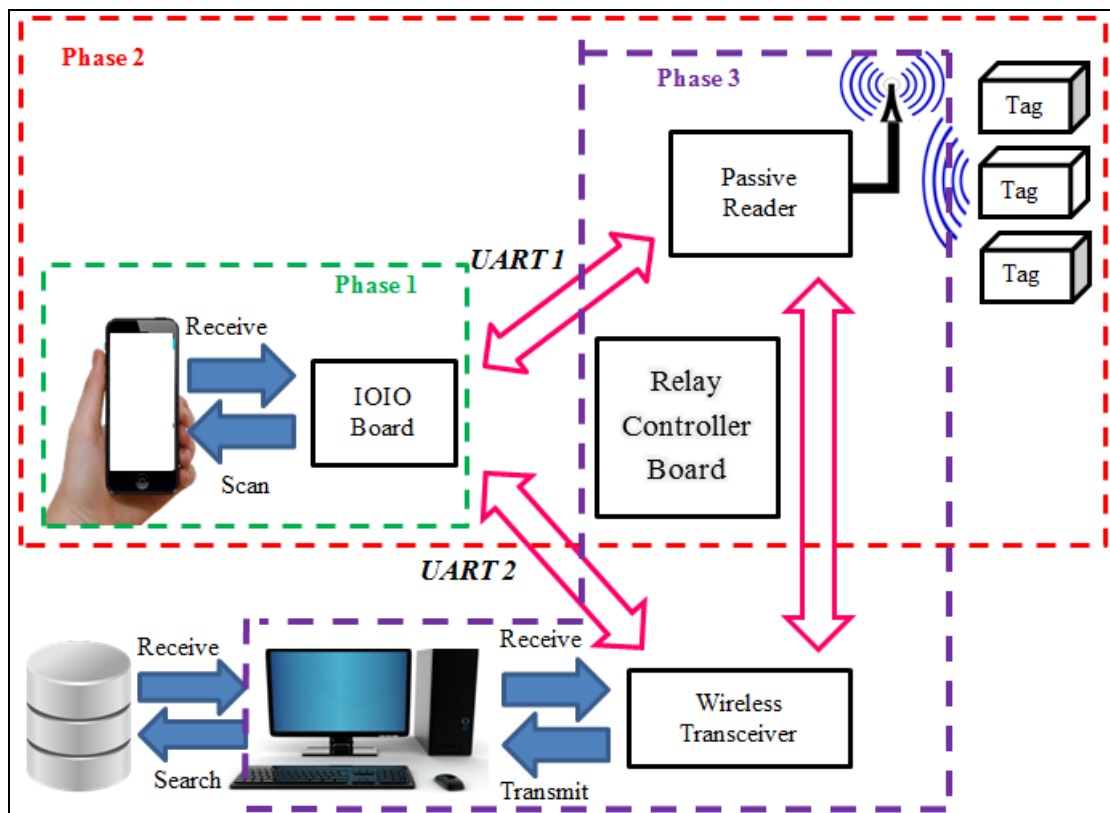


Figure 3.6. The Block Diagram of the Hardware Development

### 3.3.1 Phase 1

Phase 1 comprises key components, namely a smart phone and an IOIO board, with the latter serving as a medium to enable communication between the smart phone and a passive reader as illustrated in Figure 3.7. Theoretically, both components do not communicate with each other because the smart phone is not equipped with UART communication. Therefore, IOIO-OTG acts as a USB host and is connected to Android devices with a USB slave.

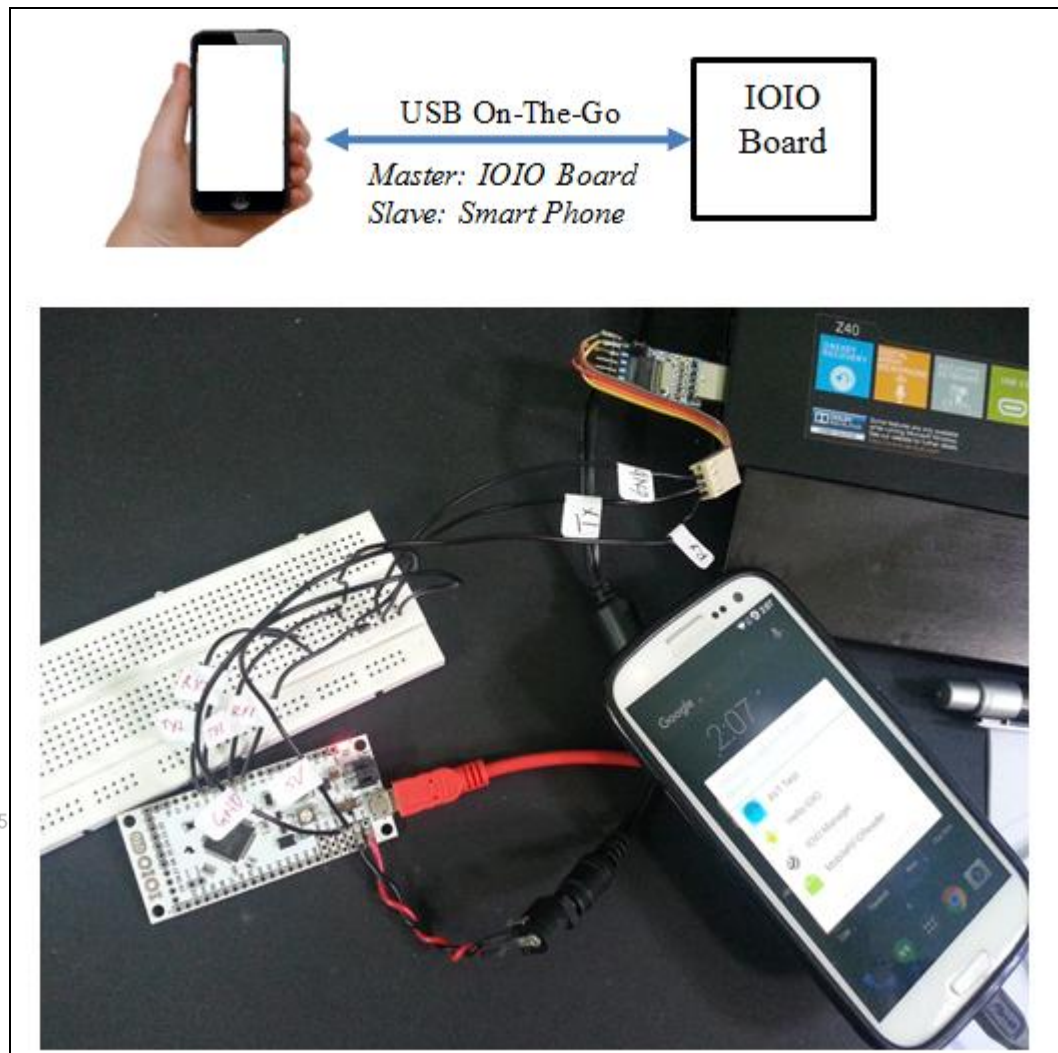


Figure 3.7. Phase 1 of Hardware Development

In general, an OTG USB can serve as either a *Slave* or a *Master*. In contrast, an IOIO-OTG USB and a smart phone can serve as a *Master* and a *Slave*, respectively. In these roles, the former and latter can send and receive data to be controlled, respectively. Nevertheless, not all smart phones are equipped with USB OTG to serve as a USB



Master. Therefore, in this study, an IOIO-OTG was used as an intermediary between a transceiver and a smart phone such that they could communicate with one another.

### 3.3.1.1 IOIO-OTG with Smart Phone Integration

Figure 3.8 shows the illustration of the hardware development. Importantly, an Android device must be able to run on USB Debugging Mode in Android after connecting the device directly to a computer with a USB cable. Such mode helps facilitate a connection between an Android device and a computer with Android SDK (software development kit). As shown, such connection was established to test the UART communication of the IOIO-OTG with the Android device.

To establish communication, the IOIO board was powered with 5 VDC and connected to the Android device using IOIO-OTG USB. As the proposed system required two UART connections, the UART tests were carried out for two UARTs. Two UARTs are needed because UART1 is for connection between IOIO board and passive reader while UART2 is connected between IOIO board and wireless transceiver. UART1 was connected to pin 10 and 11, while UART2 was connected to pin 13 and 14. To test whether the UART communication was successful, a computer was used because it could support such communication with the use of a number of software applications, such as X-CTU, HyperTerminal, Tera Term, and PuTTY.



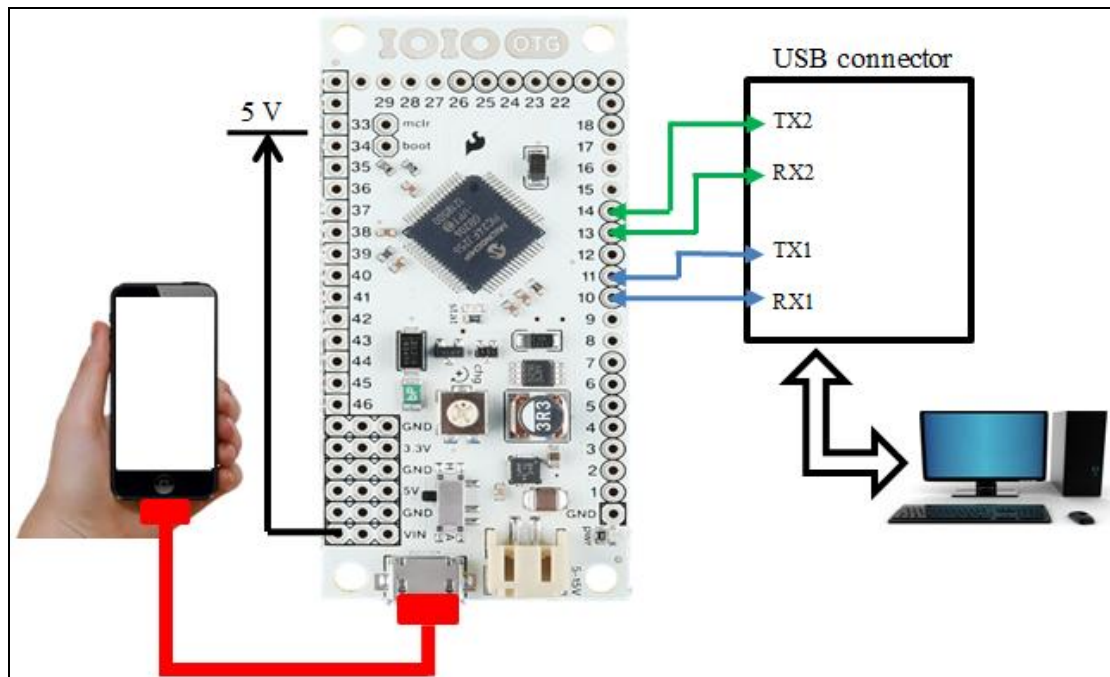


Figure 3.8. IOIO-OTG with Smart Phone Integration

Certainly, USB Debugging Mode can help improve access to Android devices when coding new applications, as programmers will definitely not write codes directly on such devices. Instead, they will develop an appropriate environment for the Android SDK to write codes of applications on a computer. Using the USB Debugging Mode, the programmer can transfer the applications to a device for testing. This direct transfer is made possible because USB Debugging Mode establishes a direct connection between an Android device and a computer, making both to be ready for high-level actions. Some Android devices do not automatically load the Developer Options. Therefore the programmer needs to activate such options by tapping the text “Build Number” seven times as shown in Figure 3.9. Figure 3.10 shows the flow of USB Debugging Mode on Android devices.

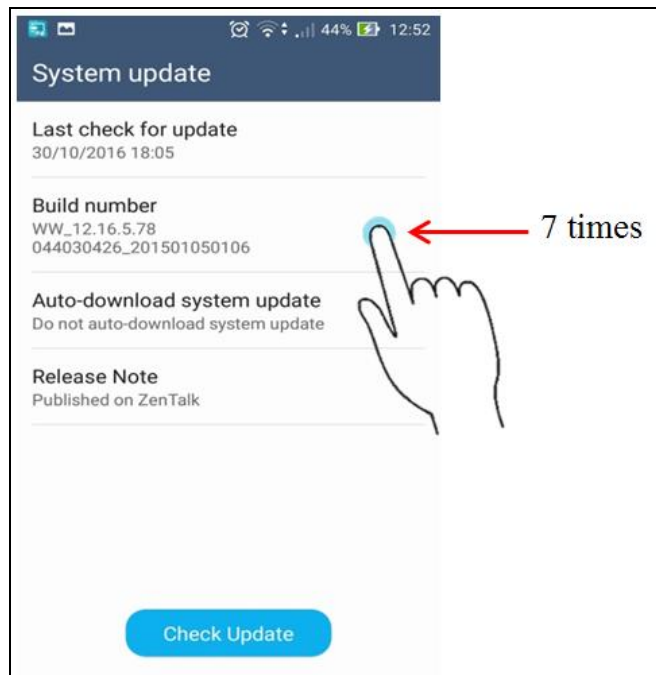


Figure 3.9. Enabling the Developer Options

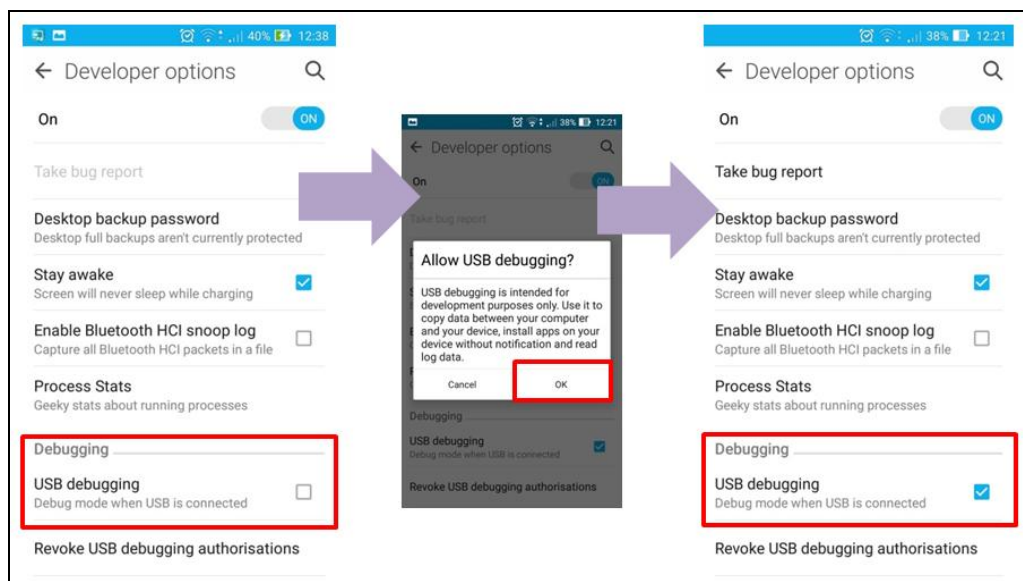


Figure 3.10. Enable USB Debugging



Figure 3.11 shows the setup of the UART communication test performed in this study. This test of an Android application helped ensure that data could be transmitted and received accordingly to establish successful communication. In fact, only after correct messages and commands had been successfully received, the development proceeded to Phase 2, which is the second phase of the development. In this study, Android Studio was used to develop the Android applications.

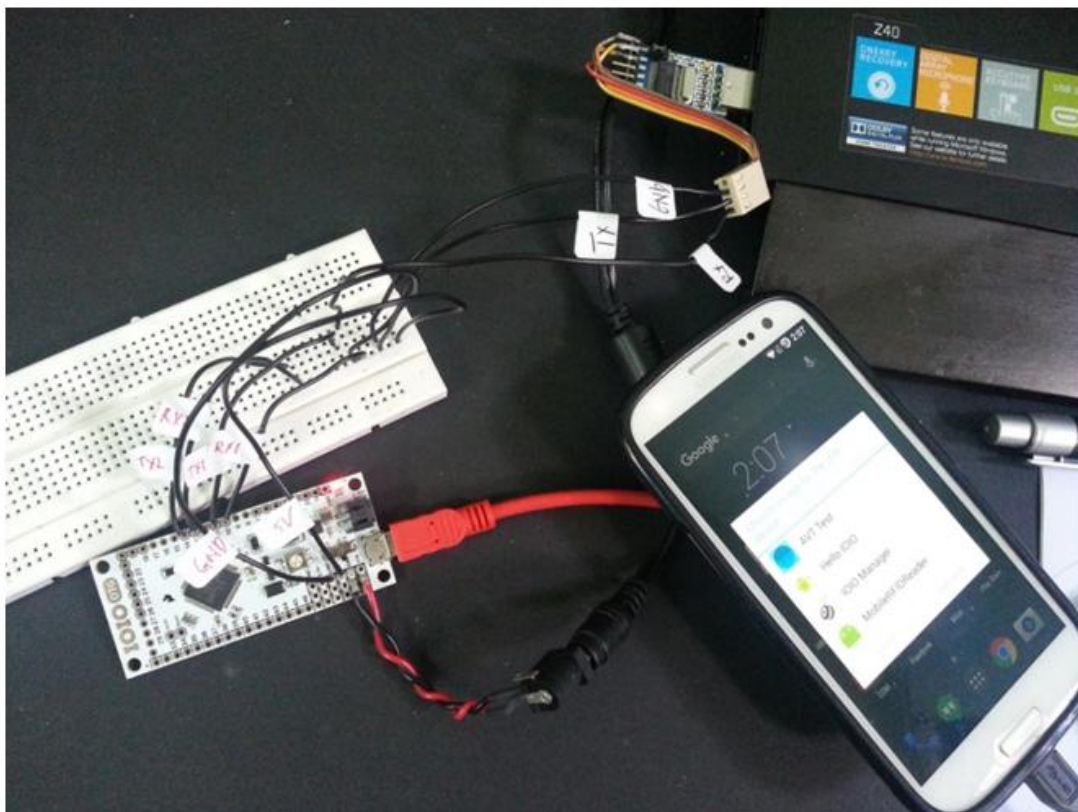


Figure 3.11. UART Communication Test

To test communication between the Android device and the IOIO board, a simple application of UART communication was developed. With more than one USB OTG

application, a layer would appear on the screen to allow users to select their preferred applications shown in Figure 3.12, or a default application would be automatically displayed. For this study, the developed application could support Android OS 6.0 and higher versions of the OS.

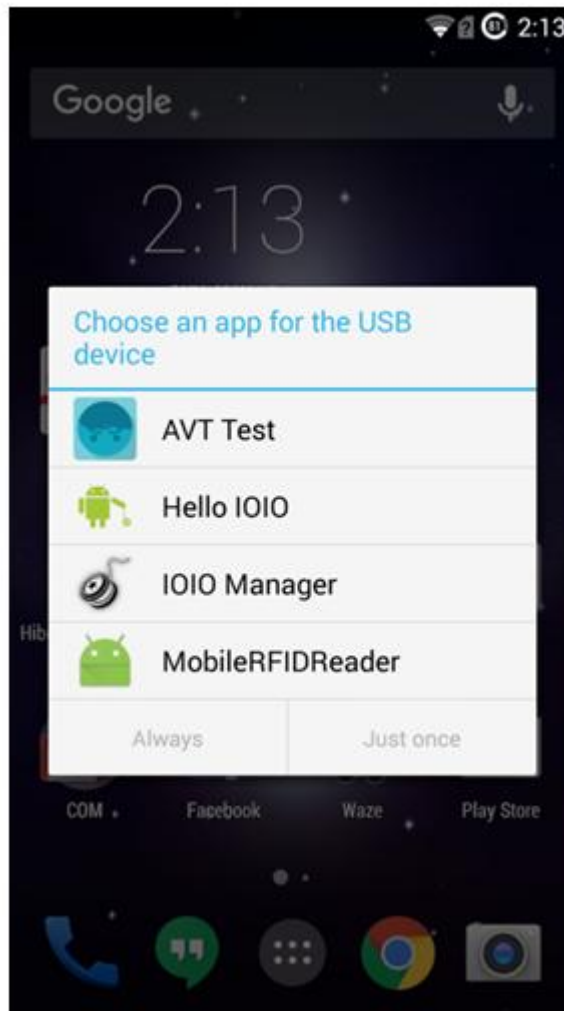


Figure 3.12. Selections of Preferred Application



### 3.3.2 Phase 2

Phase 2 involved the use of key components of smart phone, an IOIO board, a passive reader, and a PC. Figure 3.13 shows the connection among the three key components. For the IOIO board, it comprises two different outputs, which are 3.3 V and 5 V; while its input power ranges from 5 to 15 VDC. Therefore, IOIO board consumes 5 VDC from the power bank or adapter, while passive reader is powered with 5 VDC from the output power source of IOIO board.

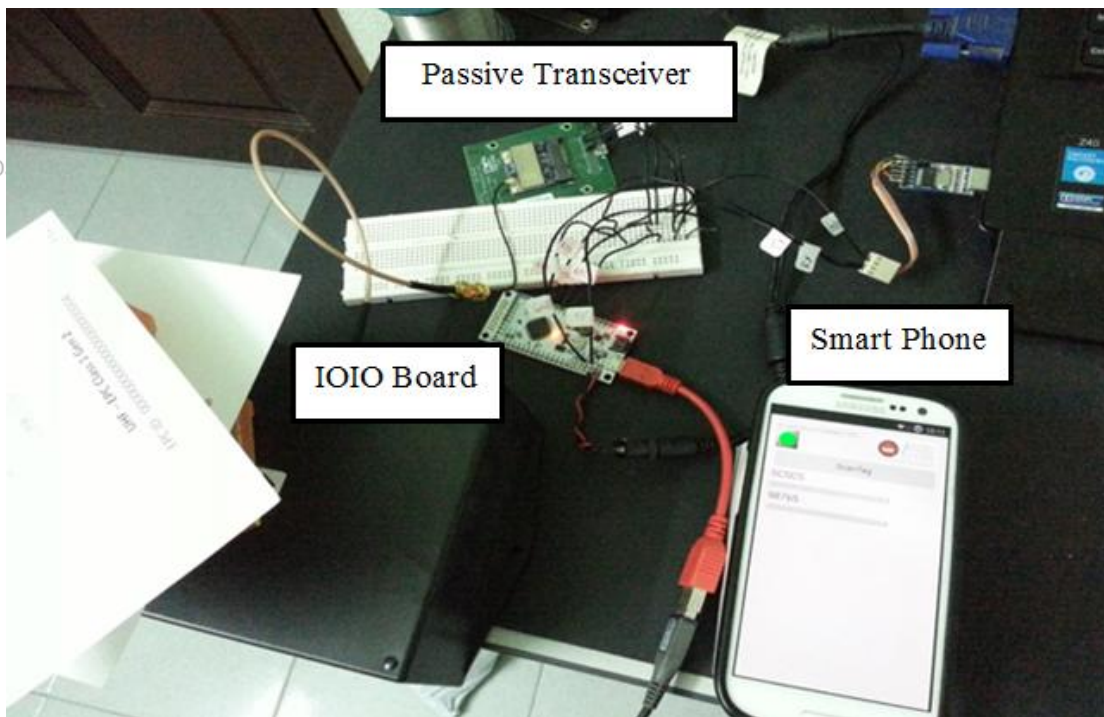


Figure 3.13. Phase 2 of the Hardware Development

### 3.3.2.1 Passive Reader

Before developing the prototype, the passive reader command had to be determined, requiring a connection as illustrated in Figure 3.14.

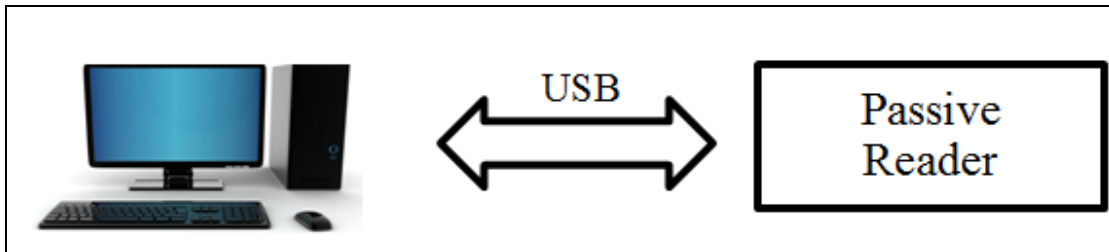


Figure 3.14. Connection of Passive Reader with PC

Figure 3.15 shows the List of Firmware Command, which is equipped with request command for programming applications.

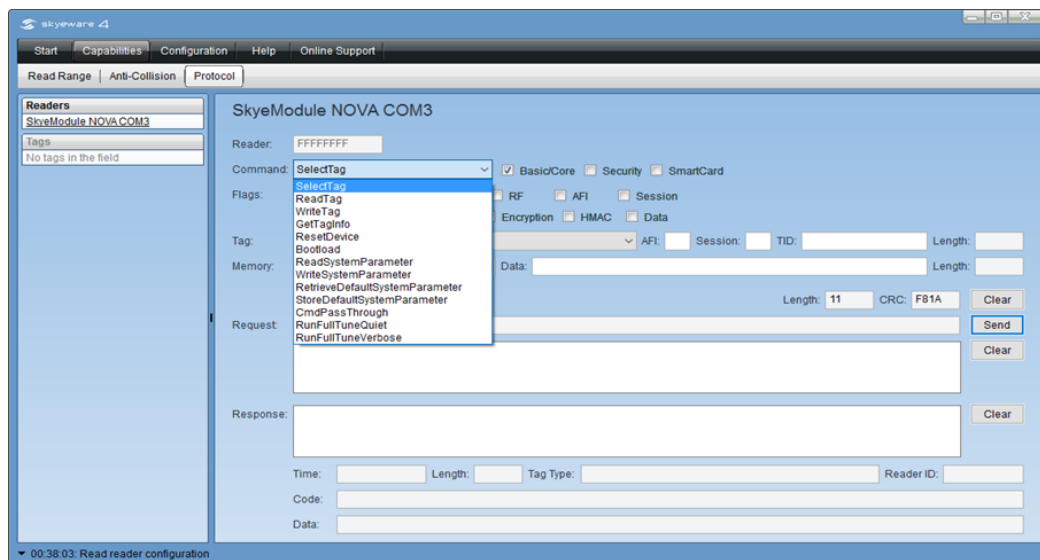


Figure 3.15. List of Firmware Command



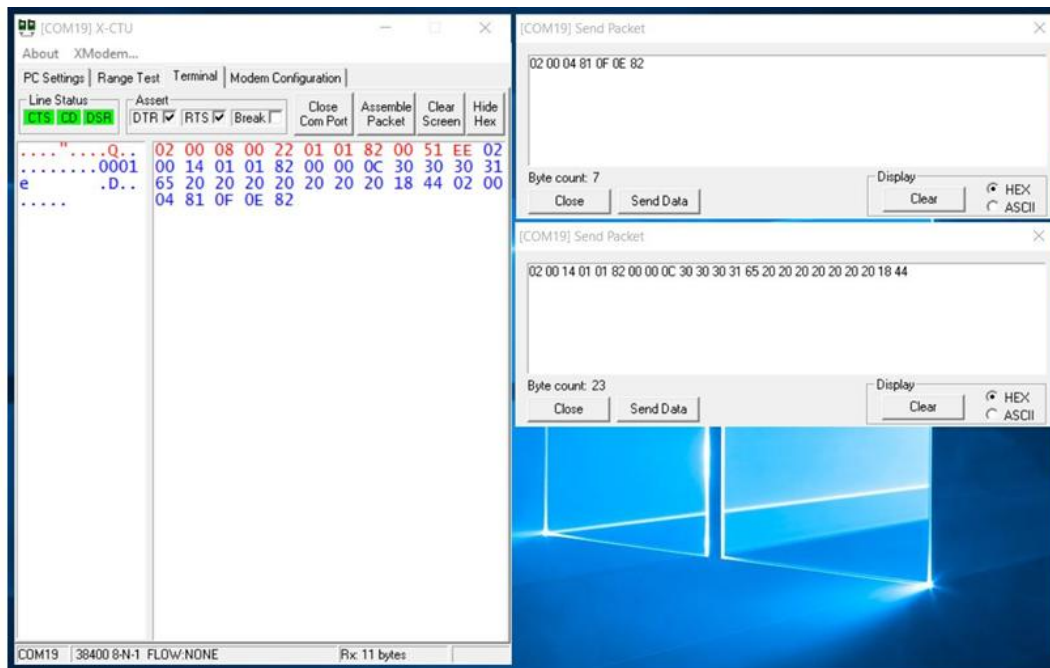


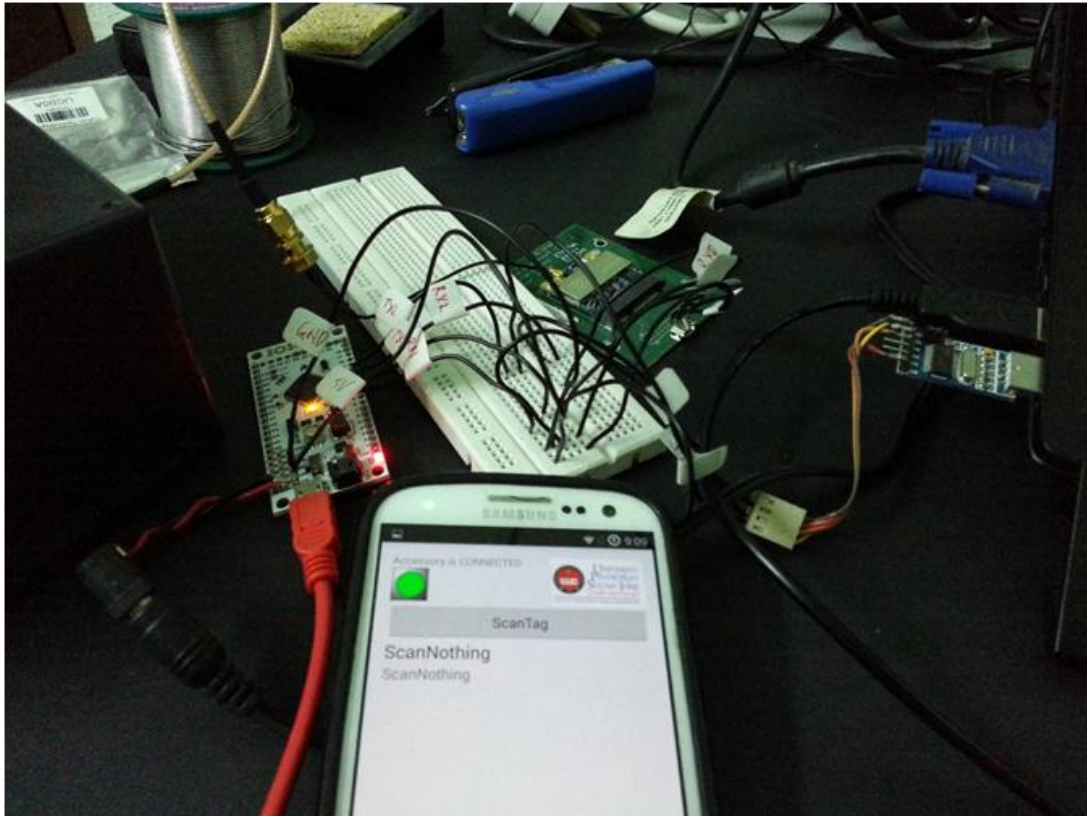
Figure 3.17. Scanning Tag with the PC using X-CTU Firmware

### 3.3.3 Phase 3

Phase 3 involved the enhancement of the proposed system developed in Phase 2 by integrating it with Zigbee wireless transceiver as depicted in Figure 3.18. In its original form, the system had limited read range communication, which depended on the antenna of the reader. The original system was clearly not practical for long range communication applications.

Furthermore, the database containing the details of tagged objects was located in the PC. Therefore, wireless communication was entailed to retrieve relevant details. Even

though all Android devices support Wi-Fi communication, they are not suitable for tracking objects because the radius of Wi-Fi communication is small, which is only applicable for home networking.



*Figure 3.18.* Phase 3 of the Hardware Development

### 3.3.3.1 Relay Controller

Relay Controller is to conserve the energy of the battery and also provides flexibility in switching between two modes. Figure 3.19 shows the schematic of the relay controller.

The transistor setup works as an inverting gate to conserve battery when the proposed system module functions in the mobile mode. In mobile mode, IOIO board does not sleep and whenever mobile phone is connected, IOIO board will function as a master and mobile phone function as a slave. Therefore it consumes high power. The relay will be switched off to help reduce power consumption in stationary mode. However, when the proposed system operates in the stationary mode, the relay will be turned on. Effectively, the proposed system will switch to mobile mode when it is connected to a mobile phone, and it will switch to stationary mode when the mobile phone is disconnected.

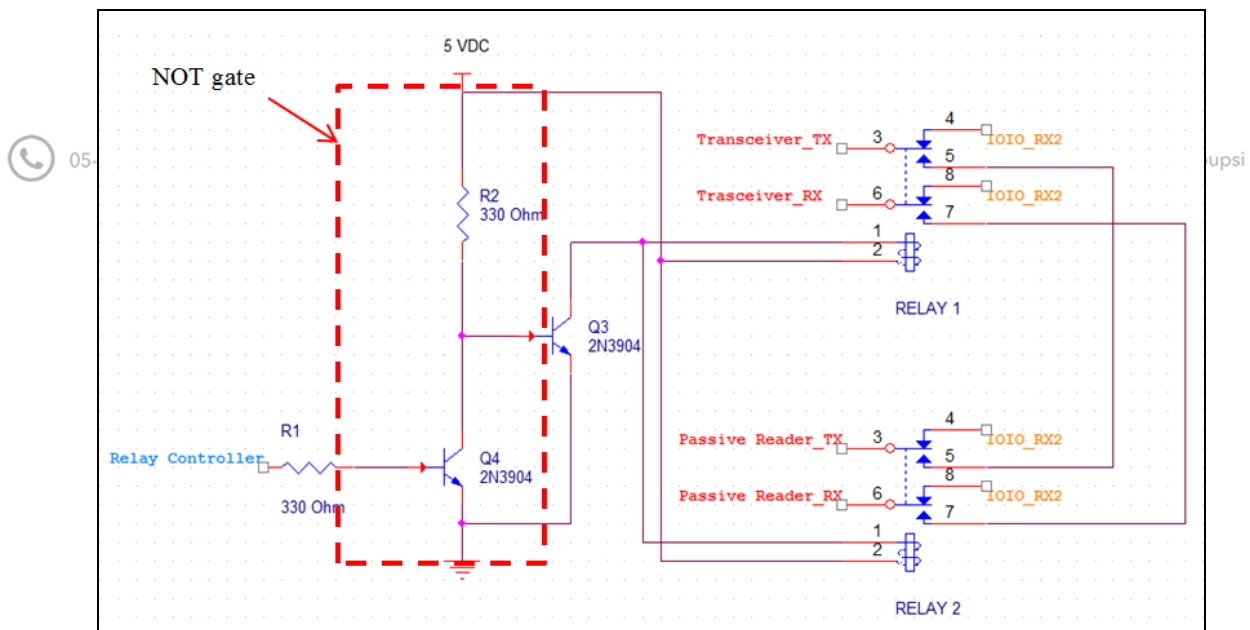


Figure 3.19. Schematic of Relay Controller

As highlighted, the mobile mode will be activated when the hardware is connected to mobile phones for tracking purposes. Figure 3.20 shows the block diagram



of the communication in mobile mode, and Figure 3.21 shows the communication in stationary mode. In mobile mode, the relay will be enabled when the wireless transceiver and passive reader are disconnected from the IOIO. The main controller in mobile mode will be the developed mobile application; whereas, in stationary mode, the main controller will be the PC application.

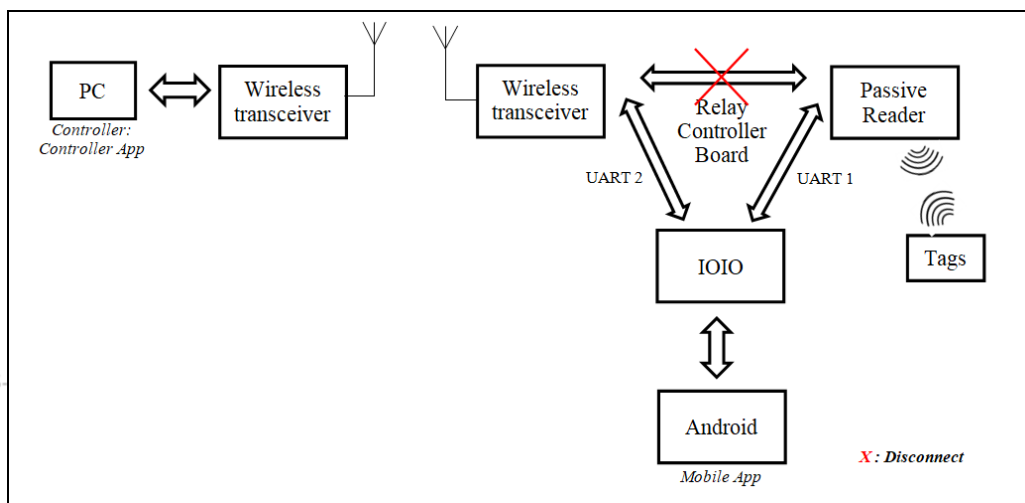


Figure 3.20. Communication in Mobile Mode

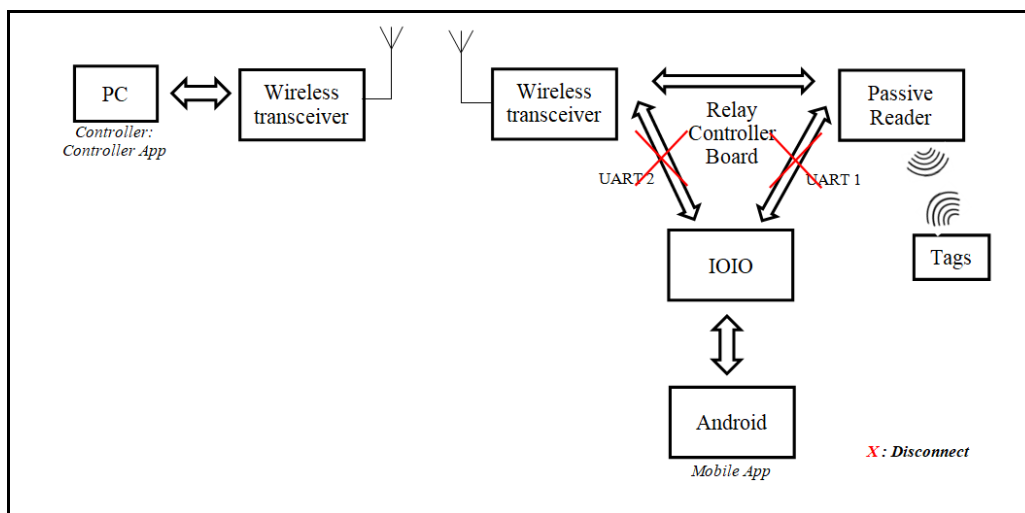


Figure 3.21. Communication in Stationary Mode

### 3.4 Software Design and Development

An android application was used as a user interface in the Android device. Pressing the application's button would allow the Android device to scan the tags and retrieve their details from the database. The development of this application was divided into several phases involving a number of processes. The complete source codes of the developed software are shown in Appendix A. Figure 3.22 shows the flow of the processes of the software development.



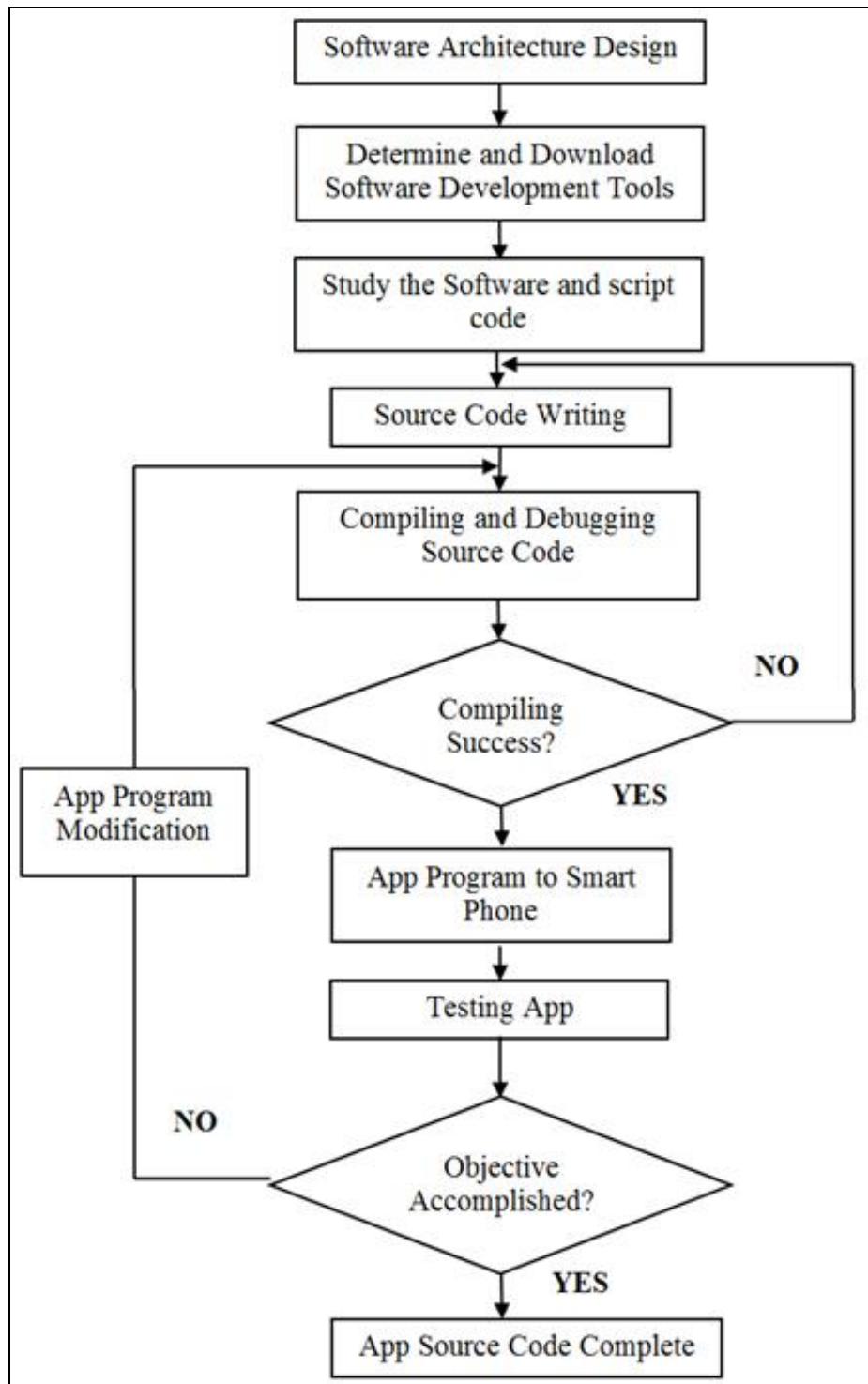


Figure 3.22. The Processes of the Software Development



Essentially, the software development consisted of three phases, which were carried out with the use of two types of software, namely Android Studio and TCL Scripting. Android Studio was used to develop the mobile application, while TCL scripting was employed for the development of the database. As stated, Android Studio was used to create the mobile application for this project research. In addition to being non-proprietary, this software has fewer issues relating to bugs and is more stable compared to its rivals, such as Eclipse. Thus, it is not surprising that Android Studio has attracted a lot of followers among android application developers since 2013.

Actually, Android Studio is the official IDE for Android application development based on IntelliJ IDEA. Android Studio functions with SDK tools, such as SDK Manager, offering several procedures, which include alternative instructions for using the SDK tools rather than using the command line. To run the software, Android Studio needs to be installed with Java SE Development Kit (Rojatkar, Jengathe, Khairnar, & Lengure, 2016; Singh, Sharma, & Singh, 2016).

On the other hand, TCL is a scripting language and an interpreter of respective language that is designed to be more efficient for embedded applications. Moreover, the interpreter has been extended from UNIX to DOS and Macintosh environments. As a scripting language, TCL is similar to other UNIX shell languages, such as the Bourne Shell, C Shell, Korn Shell, and Perl. With sound programmability, such shell programs enable smooth execution of programs (variables, control flow, procedures) to build complex scripts that assemble programs into a new tool tailored for users' needs.





Moreover, TCL has the ability to easily add TCL interpreter to applications, thus making it more robust and flexible compared to other shell programs. As such, TCL can play the role of an extension language to help configure and customize applications. Additionally, TCL has simple constructs, which are quite similar with those of C, thus easing the process of adding new TCL primitives by writing C procedures (Welch, Jones, & Hobbs, 2003).

Figure 3.23 shows the four phases involved in the software development of the proposed system. Phase 1 involved the development of a temporary application for UART communication test. In Phase 2, a mobile application for tracking and monitoring purposes was developed. In fact, Android Studio software was used in this development in the first and second phase. Phase 3 involved TCL scripting to develop a database that was located on the computer. The complete development of software to fulfil the objectives of the proposed system was realized in Phase 4.



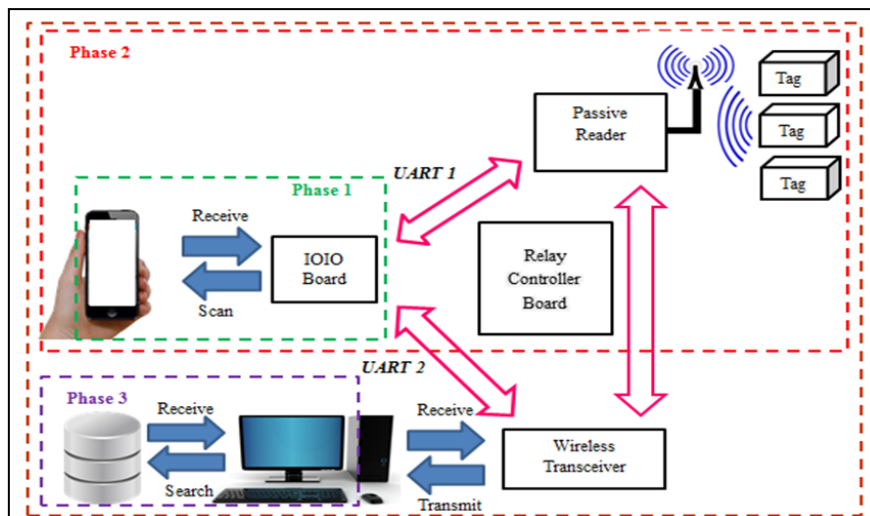


Figure 3.23. The Block Diagram of the Software Development

### 3.4.1 Phase 1

Given that an Android device is not equipped with serial communication features, an interface must therefore be developed to enable such a device to communicate with other devices through it. To support serial communication of Android devices with electronic circuits, a simple Android application was developed in this study.

Figure 3.24 shows the Android application layout of the Android device. As illustrated, the red box indicates the LED button, which when pressed would turn on the LED on the IOIO board. The green box indicates the Text Area where texts would be inserted. On pressing the Send button, a string of texts would be sent and displayed on the Receive Message section of the interface as highlighted by the blue box.

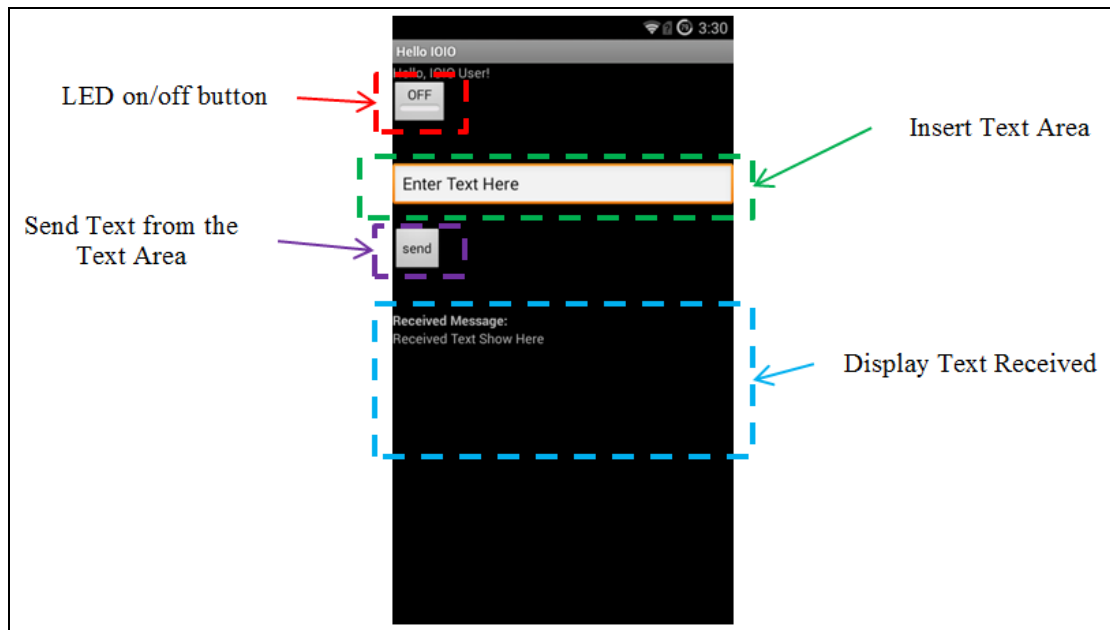


Figure 3.24. The Android Layout of Development in Phase 1

Initially, the temporary application called HelloIOIO for testing the communication between Android devices with the IOIO board was developed. Essentially, two files were used for such testing, namely *MainActivity.java* and *main.xml*. The *MainActivity.java* contains codes that controlled the activity carried out in this project. Additionally, a subclass of Abstract IOIO Activity was created to implement the IOIO framework for the specific activity of the application.

On the other hand, the *main.xml* file contains information about the arrangement of the user interface for the activity. Several command texts were created to perform a number of functions, such as *ToggleButton* to turn on the light emitting diode (LED) indicator, *TextView* to display received text and default text, *Button* to send text, and

*EditText* to insert text. Furthermore, transmit and receive pins of IOIO board was short-circuited. Figure 3.25 shows the flow of communication between the Android application and the electronic circuit.

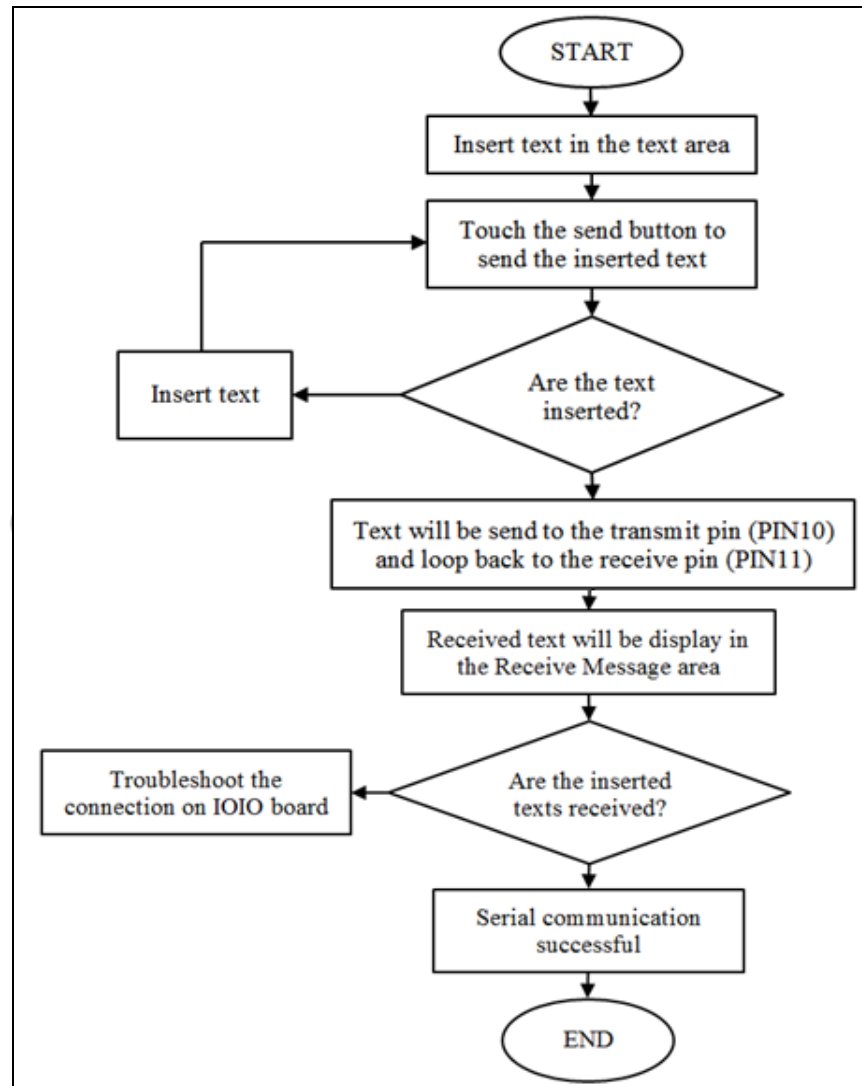


Figure 3.25. The Flow of Communication between Android Application and Electronic Circuit

Figure 3.26 shows a looping test carried out to test the transmission and reception process between the Android device and the IOIO board. To send a message to the transmit pin of IOIO board, first it must be inserted in the Text box, and followed by pressing the Send button. Subsequently, the receive pin would send the received message to the Android application and display it on the Received Message section of the application's interface.

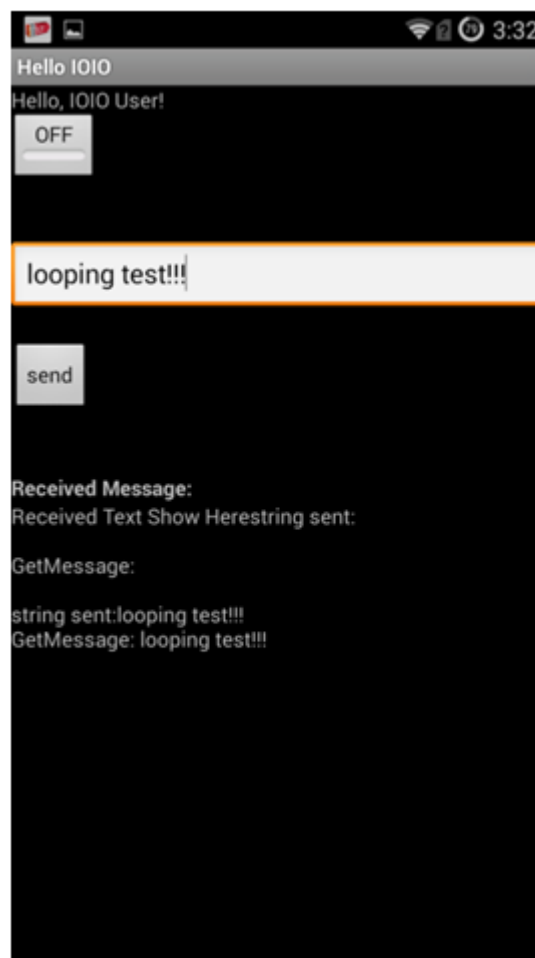


Figure 3.26. Looping Test for Transmit and Reception Process

### 3.4.2 Phase 2

The development activity in Phase 2 also used Android studio. Given that UI threads of Android are sensitive, all the IOIO processing activities were created into separate classes, namely *MainActivity* Java class and the Java classes as summarised in Table 3.2.

Table 3.2

Java classes

No	Java Class	Brief Description
1	BUF_DATA	Managing buffer
2	CircularBuf	
3	MainActivity	MAIN
4	MyDialogTAGDetails	Pop-up window for tag details
5	Uart_ContRD_RFID_Thread	Managing Thread
6	Uart_ContRD_TAGINFO_Thread	
7	Uart_ContRD_Thread	

#### 3.4.2.1 Main

*MainActivity.java* contains several codes that controlled the activities carried out in this study. Figure 3.27 shows the taxonomy of *MainActivity.java*.



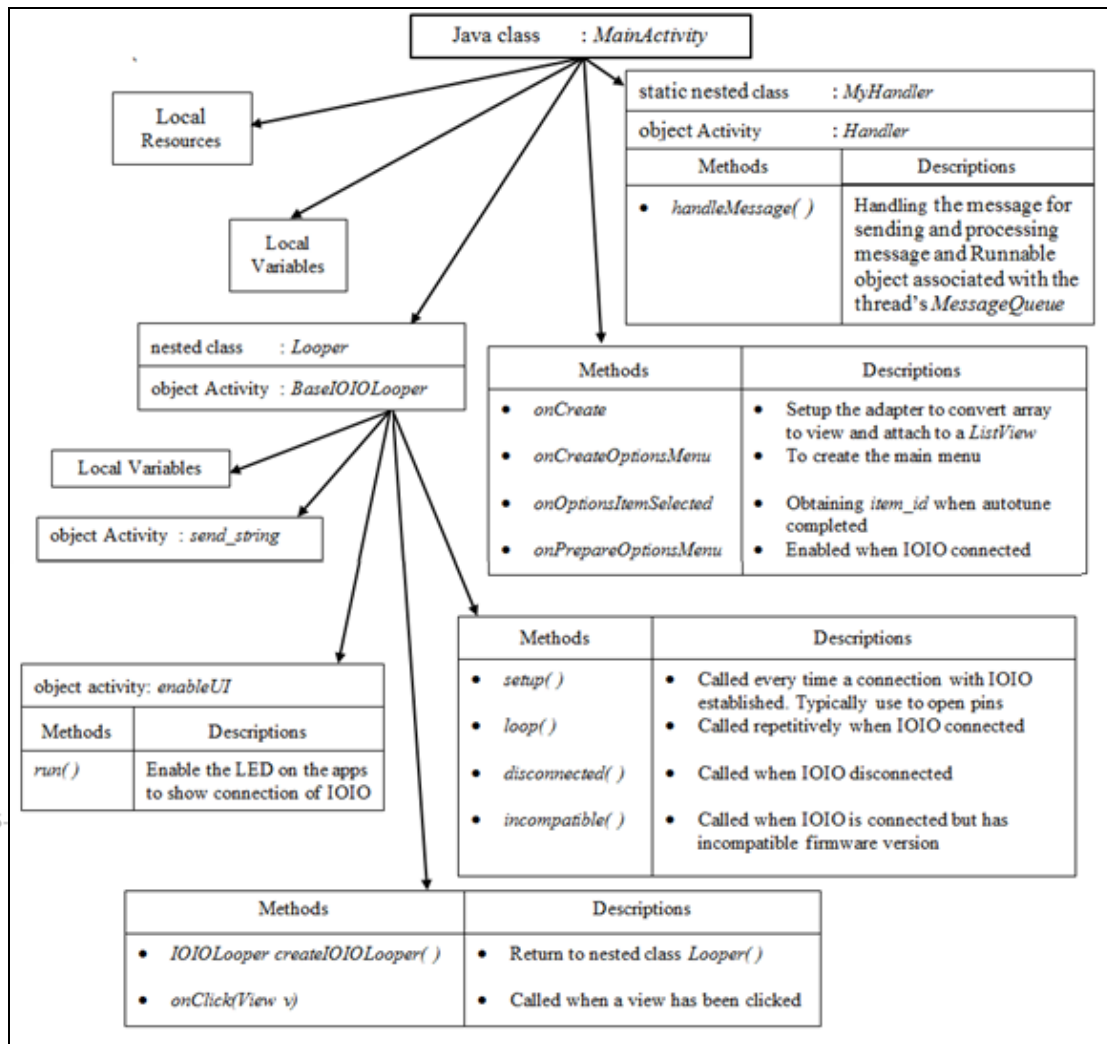


Figure 3.27. Taxonomy of MainActivity Java Class

For *MainActivity.java*, the first command line to be used is called *onCreate* after setting the local resources and variables. This command will be called when the Activity has been created. After invoking *onCreate* on the superclass, this command will associate the layout with the Activity and then create a link to a member variable that holds a reference to the *TextView*, *Button*, *ListView* and *ImageView* as shown in Figure

3.28. In addition, the *onCreate* command will create an adapter and convert an array to view and attach the adapter to the ListView.

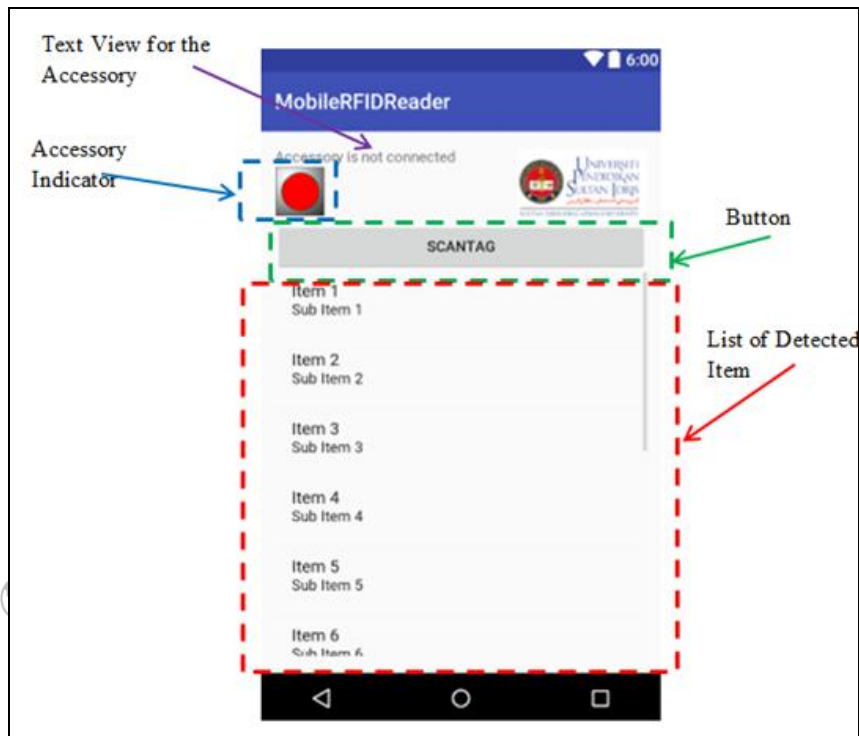


Figure 3.28. The Layout of MainActivity.java

The first class that was displayed after setting local resources and variables was the *MyHandler* static nested class, which was the same thread of posted messages. As the messages contained target *Handler*, the handler could not collect garbage-collected. Moreover, given that the *Handler* was static, the Activity could not be garbage-collected even after it was destroyed. The method for *MyHandler* nested class was *handleMessage*, which handled messages by using switch case functions. Figure 3.29 shows the flowchart of the *handleMessage* class.

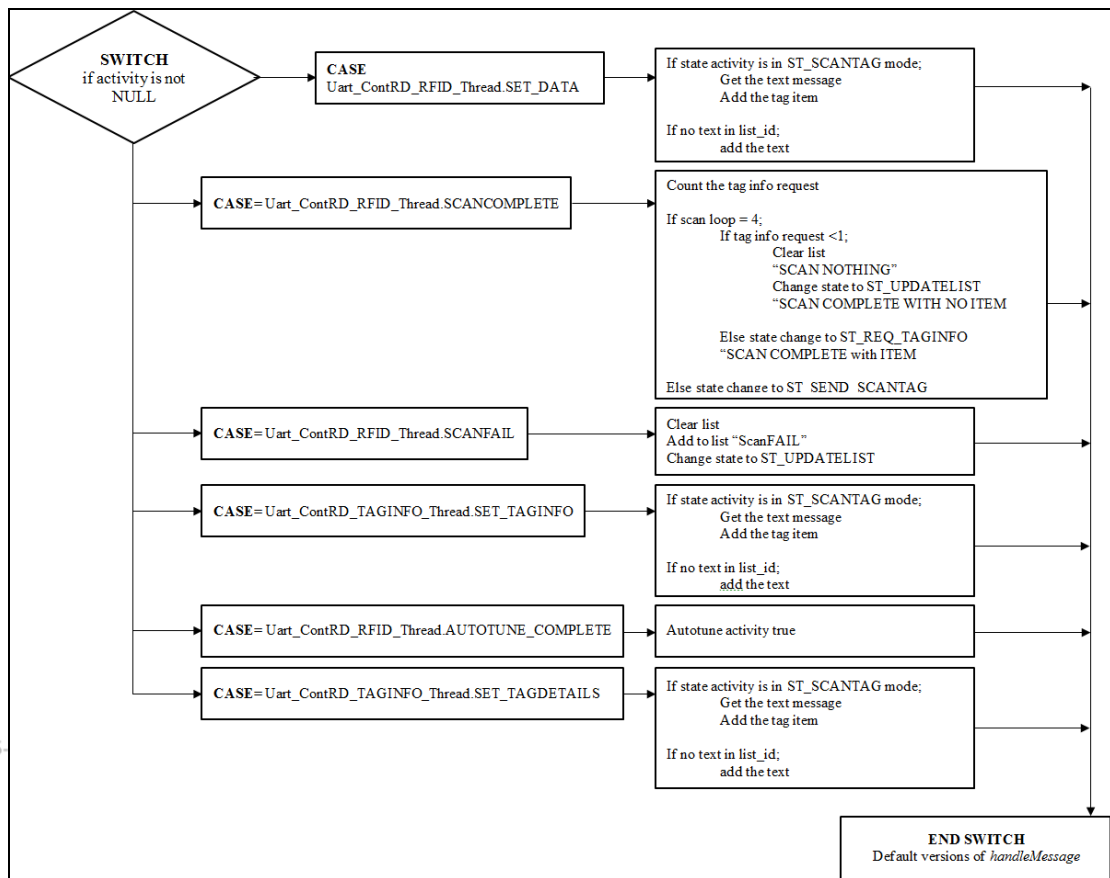


Figure 3.29. The Flowchart of handleMessage class

The next class of MainActivity was the Looper() that extended the thread activity of BaseIOIOLooper. This was the thread of all IOIO activities that were carried out that would run every time the application was resumed and aborted. The method setup() would be called right after a connection with the IOIO had been established. However, loop() would be called repetitively until the IOIO was disconnected.

### 3.4.2.2 Thread

Two types of extended thread classes for two different activities would run separately, namely the tag ID thread and the read tag information thread. Both threads were created by extending the Thread class, with which the override run() method must be used. Figure 3.30 shows the taxonomy of the Read Tag ID thread, and Figure 3.31 shows the taxonomy of the Read Tag Info Thread.

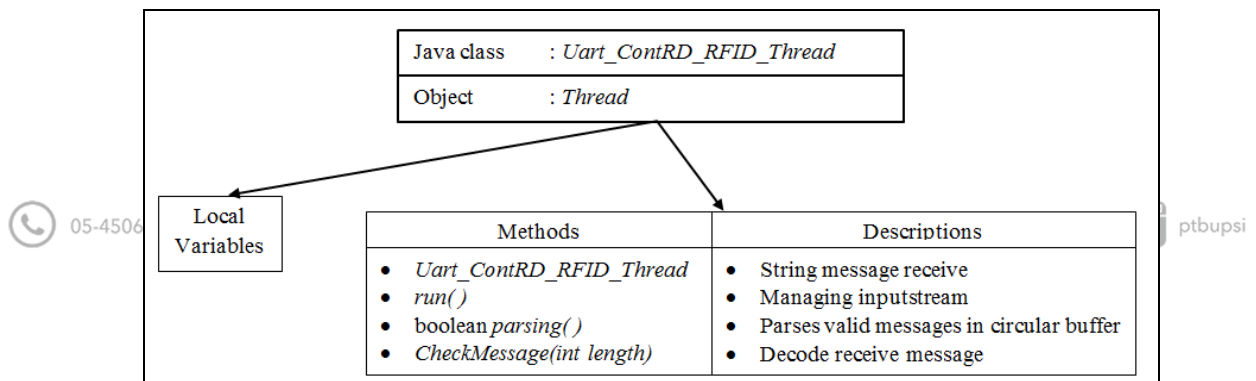


Figure 3.30. The Taxonomy of Read Tag ID Thread

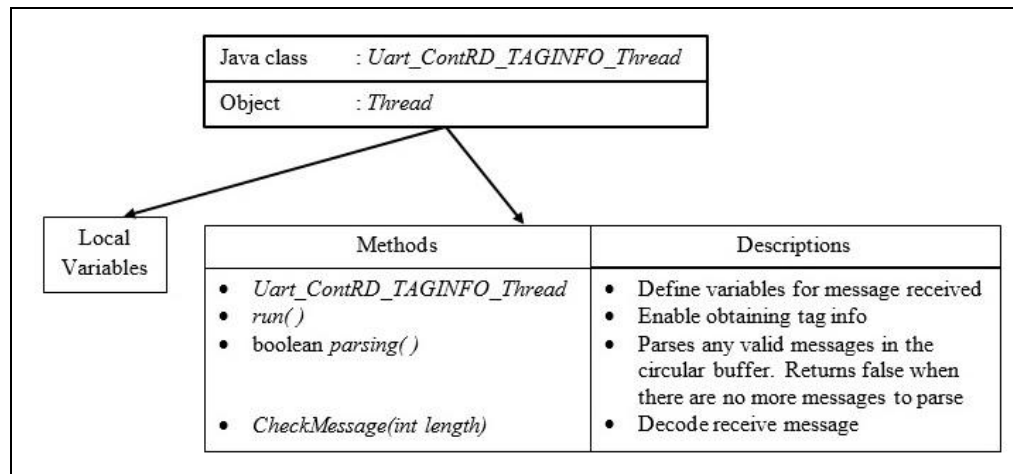


Figure 3.31. The taxonomy of Read Tag Info Thread

### 3.4.3 Phase 3

Phase 3 was the development of the database using TCL programming language. *TCL* is a high-level, general-purpose, and dynamic programming language. The development was performed by setting the serial port communication and configuring options on a channel. The details of ID channel are shown in Figure 3.32. Specifically, the blocking, buffering, mode, eofchar, and in translation of the ID channel was configured as summarized in Table 3.3.

```

# #
# # MAIN
# #
set fh [open "\\\\.\\COM3" RDWR]
#set fh [open COM3: RDWR]
fconfigure $fh -blocking 0 -buffering none \
           -mode 38400,n,8,1 -translation binary -eofchar {}

```

Figure 3.32. Details of ID channel

Table 3.3

The Details of the ID Channel

Channel ID	Descriptions
Blocking <i>Boolean</i>	determines whether I/O operations on the channel can cause the process to block indefinitely
Buffering <i>Newvalue</i>	<p><b>full</b> the I/O system will buffer output until the flush command is invoked</p> <p><b>line</b> the I/O system will automatically flush output for the channel whenever a newline character is output</p> <p><b>none</b> the I/O system will flush automatically after every output operation</p>
Mode	Format: baud rate, parity, data, stop.
Eofchar	default value for <b>-eofchar</b> is the empty string in all cases except for files under Windows. In that case the <b>-eofchar</b> is Control-z (\x1a) for reading and the empty string for writing
Translation <i>Mode</i>	<p><b>Binary</b> No end-of-line translations are performed</p> <p><b>cr</b> The end of a line in the underlying file or device is represented by a single carriage return character</p> <p><b>crlf</b> The end of a line in the underlying file or device is represented by a carriage return character followed by a linefeed character</p>

The following event handler setup involved setting the event handlers, such as *matched*, *command*, and *datato*“0”. Then, the variables *returnName* and *returnDescription* were set to “TR” and “TD” setup, respectively. Subsequently, the event handlers were set to *if else loop* by checking the existence of a read handler. If the read handler was present, the file event handler would be set to be readable to start the delimiter. Otherwise, the handler would be set to *while loop*.

In the *while loop* mode, the process would wait until the variable was matched with the message that arrived. If the matching of message and debug variable had occurred, the process would debug the message. Otherwise, the process would decode the received message and set the channel to “done 1” if no matching message had occurred and the debug variable was equal to “1”. Figure 3.33 shows the flow of the database development.

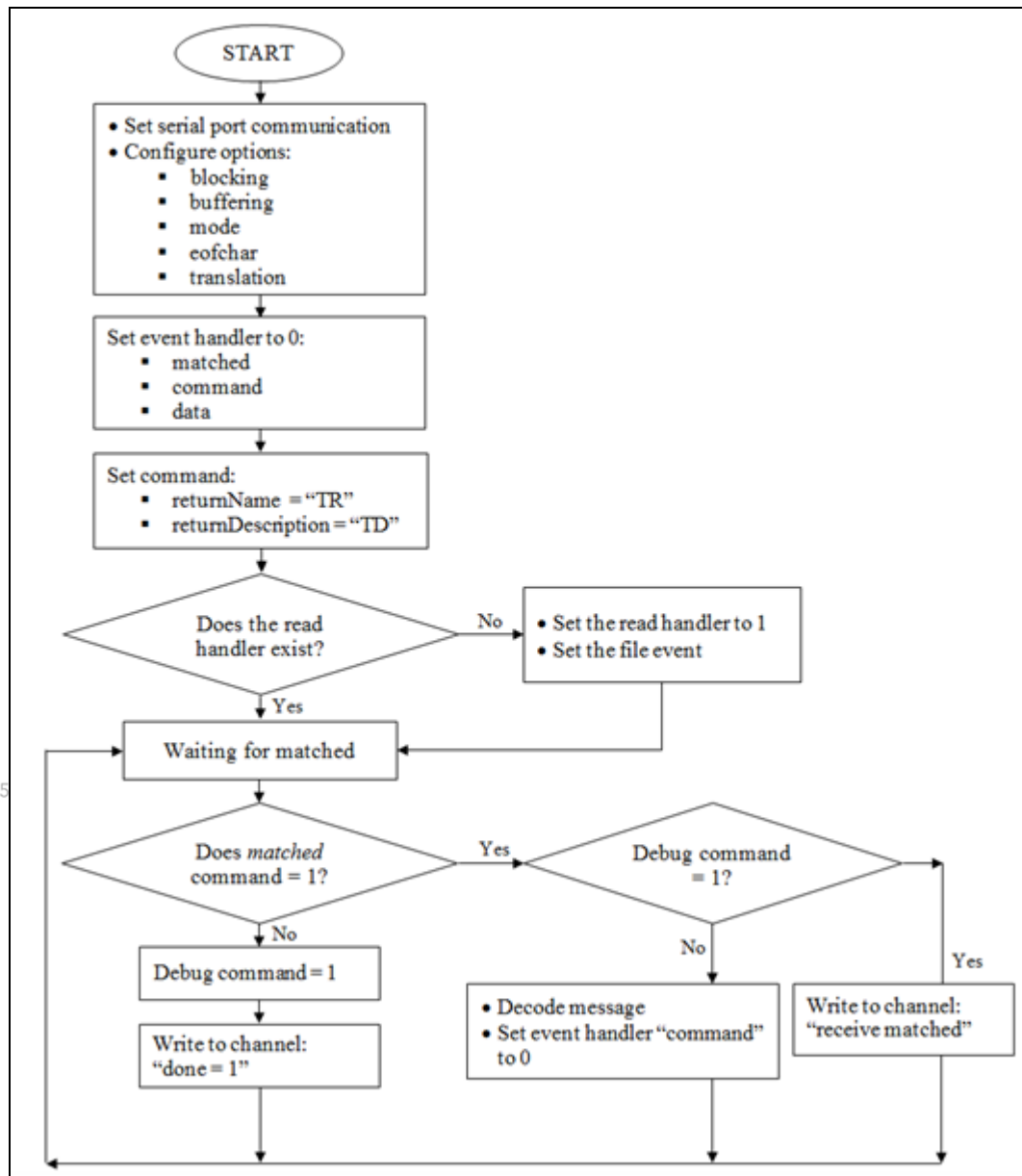


Figure 3.33. The Flow of the Database Development



### 3.4.3.1 Decoding Message

Packet message was divided into 4 different parts, namely delimiter, packet length, command and tag ID as shown in Figure 3.34. Table 3.4 summarizes the description of each part of the packet message.

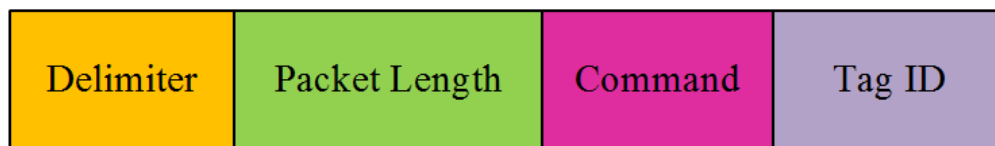


Figure 3.34. The Parts of the Packet Message

Table 3.4

The Descriptions of the Packet Message

Packet	Byte	Value	Descriptions
Delimiter	2	02	Message will only be decoded when the delimiter value equals to 02 or else it will be considered as rubbish
Packet Length	2	31	To determine the maximum message to be decoded
Command	2	TR – assemble packet TD – fetch from database	Differentiate two processes
Tag ID	25	25 digits	Can be modified according to users

Table 3.5 shows sample data stored in the database. When the system received the command TR, only the Tag ID would be displayed as highlighted in Figure 3.35.

Table 3.5

The Sample Data Stored in the Database

Tag ID	Part Number	Quantity	Description
00000000000000000000000000000001	A0010-0008	10pcs	Philips Screw Driver; Medium size;
00000000000000000000000000000002	A0010-0002	1000pcs	Philips Screw; Diameter: 5mm; Length: 30mm;
00000000000000000000000000000003	W0001-0012	1000pcs	Wire 12AWG; Wire; Length: 10meters;

```

C:\Users\user\Dropbox\Zalina\PortablePRFID\TCL>tclsh UART_DataServer_v3.tcl
fileevent set
name matched
Packet Return -> <0a TR000000000000000000000002;SCREW>
name matched
Packet Return -> <0a%TR000000000000000000000003;WIRE 12AWG>
name matched
Packet Return -> <0a%TR000000000000000000000004;WIRE 18AWG>
name matched
Packet Return -> <0a~TD000000000000000000000002;Part Number: A0010-0002; Quanlit
y: 1000pcs; Description: Philips Screw; Diameter: 5mm; Length: 30mm>
name matched
Packet Return -> <0a~TD000000000000000000000003;Part Number: W0001-0012; Quanlit
y: 100pcs; Description: Wire 12AWG; Wire Length: 10meters>
name matched
Packet Return -> <0a~TD000000000000000000000004;Part Number: W0001-0018; Quanlit
y: 100pcs; Description: Wire 18AWG; Wire Length: 10meters>

```

Figure 3.35. The Descriptions of TR and TD in the Database

### 3.4.4 Phase 4

Phase 4 is the final phase of developing Android app for the proposed system. This phase is the completed phase where all the connections and communications between the key components are involve. The proposed system is named Wireless Mobile RFID Reader (WiBRED). After the developed mobile application using Android Studio has been completed, the application is save as mobile RFID reader.APK. The application can also be shared with other smart phone via Bluetooth or USB-PC connection. Figure 3.36 show WiBRED mobile application has been installed on smart phone of Android operating system.

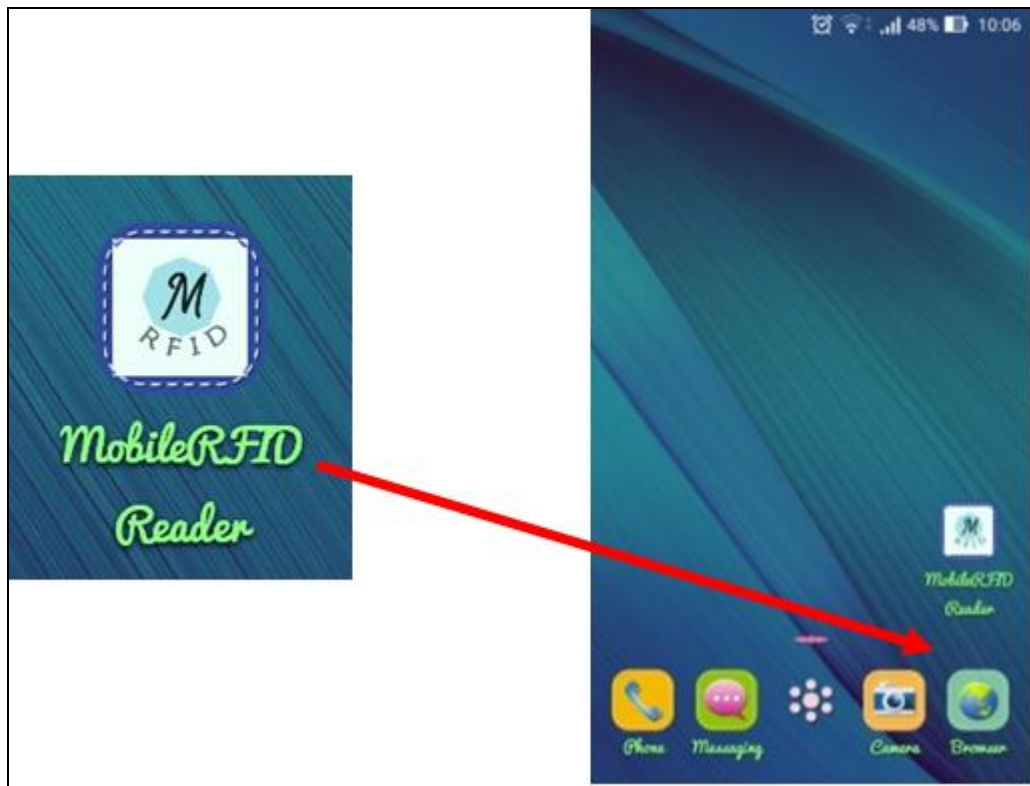


Figure 3.36. The WiBRED Icon Displayed on the Interface

### 3.4.4.1 WiBRED Application Layout

Figure 3.37 illustrates the layout of the WiBRED applications consisting of a scan button, a connector indicator, and a display area.

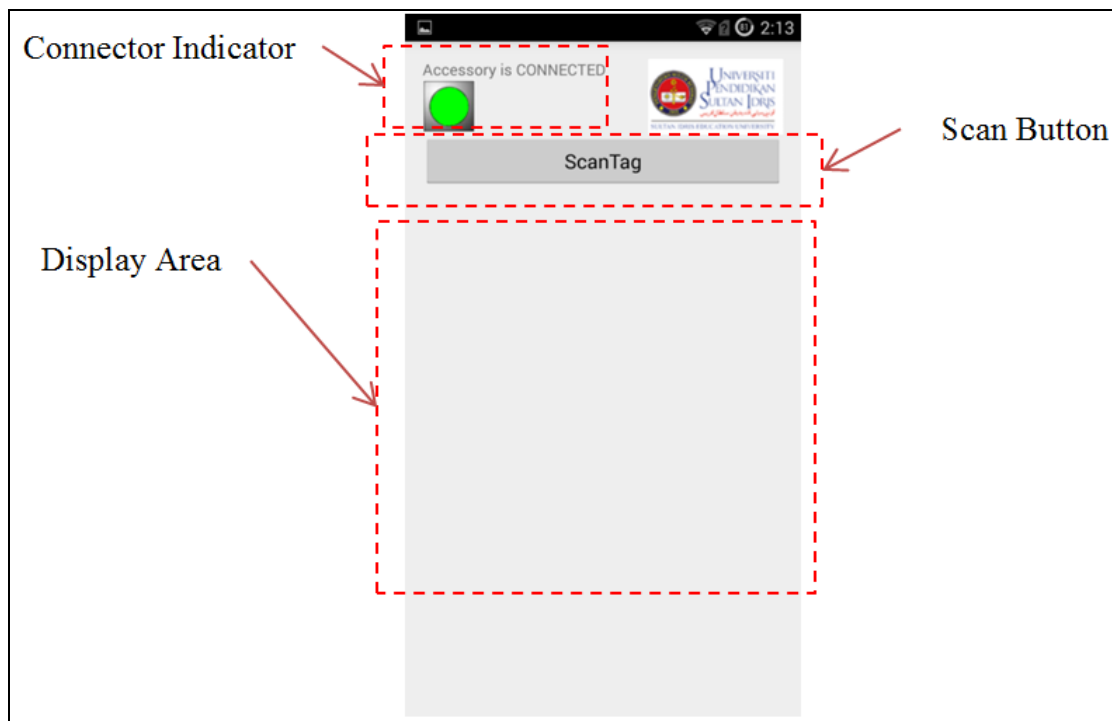
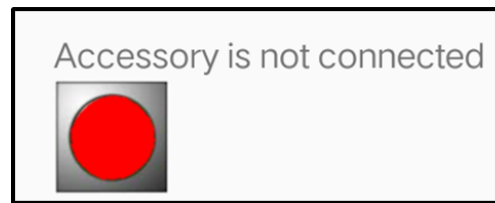


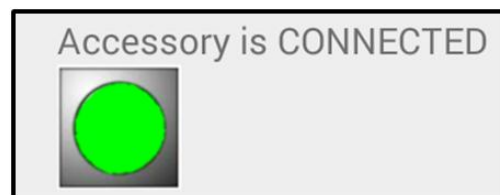
Figure 3.37. The WiBRED Application Layout

The application would notify the user that an accessory was not connected by making the indicator button red as shown in Figure 3.38. Red was chosen because it signifies the “stop/off” state or condition (Martinez-Conde & Macknik, 2014). Additionally, the text “Accessory is not connected” would appear if the hardware was not plugged in a smart phone. If the indicator button was also red even after the user had plugged in the hardware, it meant that a connection error had occurred in the accessory, not the hardware.



*Figure 3.38.* Red Indicate No Accessory Connected

On the hand, the indicator button would turn green when the hardware was connected as shown in Figure 3.39. Green was chosen because it represents the “start/on” state or condition (Martinez-Conde & Macknik, 2014). In addition, the text “Accessory is connected” would be displayed when the hardware was properly connected to the smart phone and there was no connection error in the circuit.



*Figure3.39.* Green Indicate an Accessory is Connected

Passive tags were tagged on the materials palettes, and as they passed through the portal, the passive reader would sense and record the tags’ data, which were then transferred to the wireless transceiver. Then, the wireless transceiver will transfer the data to the coordinator. Effectively, such reception and transmission of passive and active RFID data over a long distance using the mesh network enabled the active reader to



receive data in real time and send the data to a PC, on which data collected, were centralized.

In addition, the proposed system had a special designation, allowing it to communicate with the Android mobile phone. As it was attached to the active reader, data collected would appear on the phone screen during the process of searching objects, thus preventing users from going to the base station to check the number of tags that had been collected by the RFID system. As such, users (notably the manager of a company) could monitor the daily production performance through his or her smart phone, as data transmission could be performed wirelessly. Furthermore, the database would be updated in real time, making RFID an efficient method for asset tracking and inventory management. In particular, RFID smart label solutions would help manufacturing personnel to monitor and control the flow of materials in real time with ease. Figure 3.40 shows the flow of operational process of the WiBRED application.



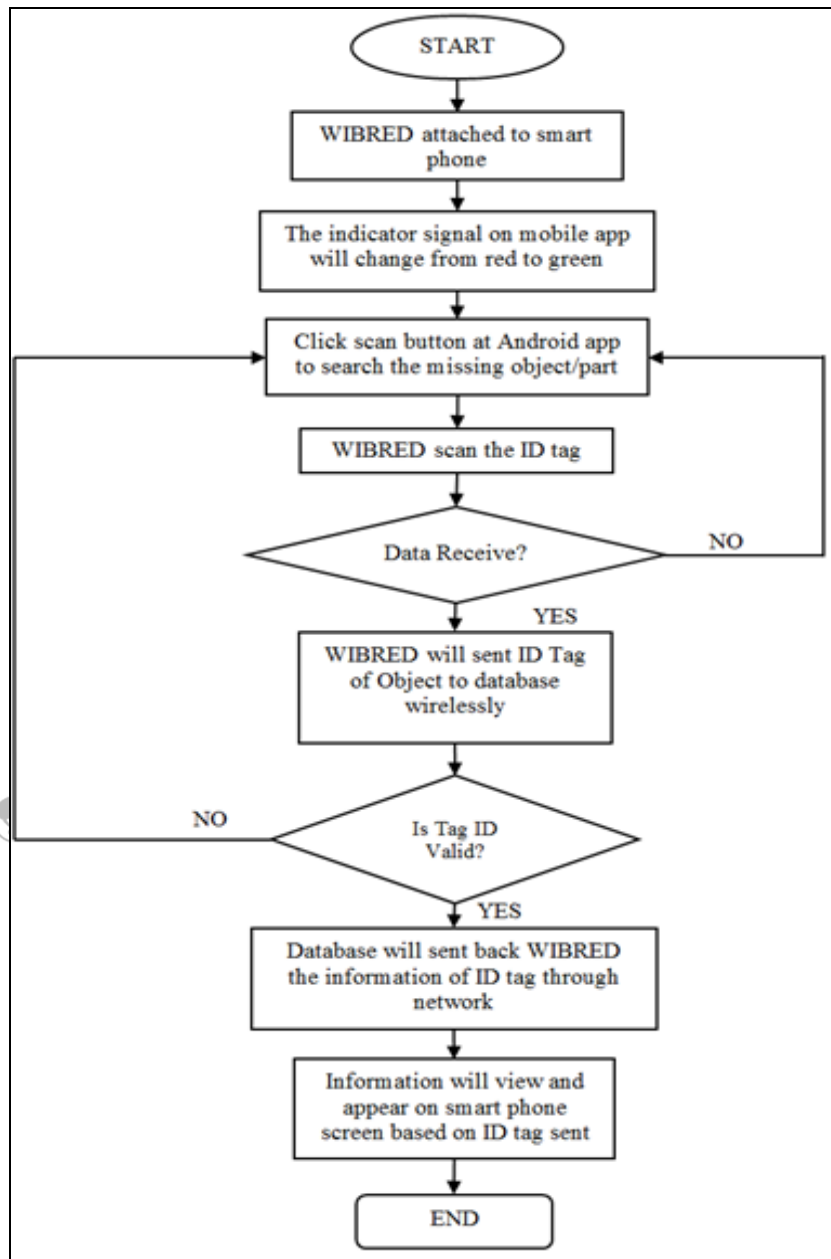


Figure 3.40. The Flow of Operational Process of the WiBRED Application

Pressing the scan tag button would display the available tagged object within a detected range on the phone screen as shown in Figure 3.41. To retrieve the details of the



tagged object, the user must click the respective tags accordingly. Figure 3.42 shows a description of a tagged object (in this case was a screwdriver).

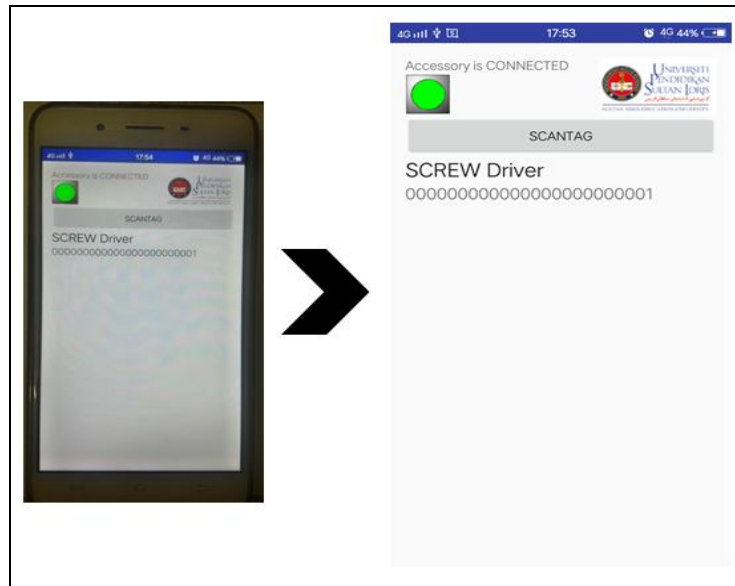


Figure 3.41. To Scan at Tagged Object



Figure 3.42. A Display of the Description of a Tagged Object

### 3.5 Summary

The research methodology used to develop the proposed system was based on ADDIE model, encompassing both relevant hardware and software. For the hardware development, three phases were carried out. First, hardware development began by integrating an IOIO board and a smart phone to ensure the passive reader would be able to communicate with the smart phone, as the latter was not equipped with UART communication. Then, a passive reader was embedded onto the integrated IOIO board. Finally, the board was embedded with an active reader, which effectively made the former as a host to control the transmission of data.

Similarly, three phases were performed in the software development. First, Android Studio software was used to develop a novel mobile application to establish serial communication between Android devices and a passive reader. Secondly, a loop back test of the mobile application was carried out by short-circuiting the reception and transmission pins of the IOIO board. The proposed system was developed after the communication between such components was tested to be successful. Thirdly, TCL programming language was used to develop the database. To differentiate between processes, two commands were set to different values, namely “TR” for assemble packet and “TD” for fetching information from the database. Upon successful communication testing of the proposed system, the mobile application was saved as an APK file, which was later installed on the Android smart phone.



## CHAPTER 4

### RESULTS AND DISCUSSIONS



#### 4.1 Overview

In this chapter, the researcher discusses the results and discussions of the WiBRED system. As discussed in Chapter 4, the testing of the system was carried out in a real world environment based on two operational modes, namely stationary and mobile mode. Testing was performed on all related components of the system, such as a mobile phone, WiBRED, tags, routers, a coordinator, and a host computer. Performance measure focused on repeatability, which is a variation in measurements taken by a single person or an instrument on a specific item under particular conditions. Therefore, taking ten measurements has been the norm for consistent and reliable calculations. Clearly, taking





50 measurements would be better than taking a smaller number of measurements, such as 20, but if the readings are between 4 and 10, such a smaller number is deemed sufficient (Salahinejad & Aflaki, 2007). Therefore in this chapter, the measurements are taking ten measurements.

## 4.2 Application Interface Setup

To conserve the energy of a rechargeable battery, WiBRED is designed to run on two modes of operation, namely mobile mode and stationary mode. On stationary mode, the system can support fixed RFID readers. On mobile mode, WiBRED can enable support higher mobility of the RFID reader via OTG-USB connection with a rechargeable battery.

A mobile application supporting Android OS was developed to scan and store details of tagged objects in a data server. Such a process was carried out in several stages. First, when a tag detected the tagged object, its identification (ID) would be compared to all information stored in the database. Then, the identity of the tagged object would be displayed on the phone screen when the ID matched with one particular information in the database. In fact, the details of the tagged object would only be displayed when specifically requested by the user. For successful communication, the COM port and baud rate of the server should match with the COM port and baud rate of the passive RFID reader. Figure 4.1 shows the configuration of the database, entailing the user to change the COM port and baud rate in order to match with those of the RFID reader.



```

184
185
186  ##
187  ## MAIN
188  ##
189  set fh [open "\\\\.\\COM3" RDWR]
190  #set fh [open COM3: RDWR]
191  fconfigure $fh -blocking 0 -buffering none \
192  -mode 38400,n,8,1 -translation binary -eofchar {}
193

```

Figure 4.1. Database Configuration

USB OTG is designed to connect a mobile phone to its accessory. The received ID from the tags would be transmitted to WiBRED and to the server terminal to find the matched tag's ID. Figure 4.2 shows the block diagram of the general hardware setup

based on the mobile mode of operation.

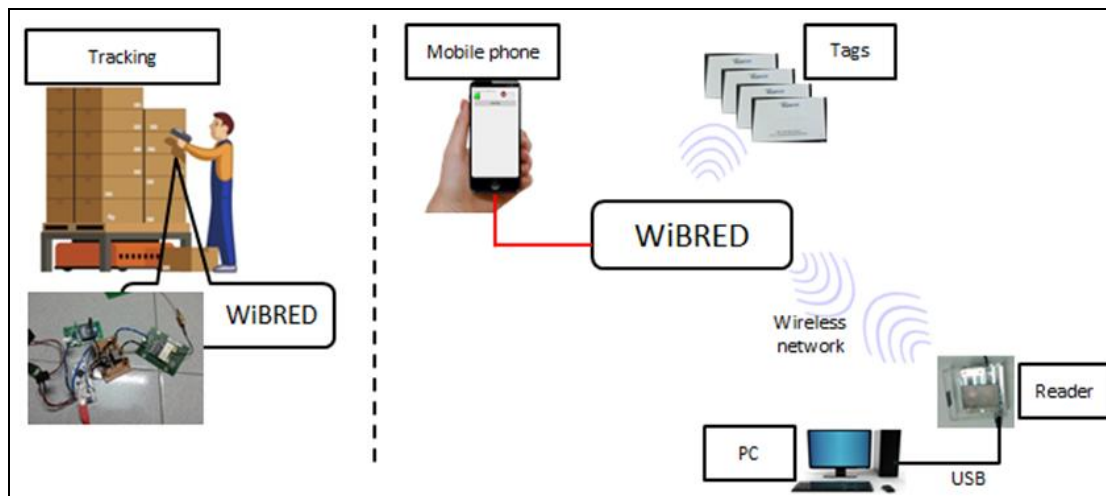


Figure 4.2. The General Hardware Setup

### 4.3 Anti-Collision Test

The main purpose of anti-collision test was to test the capability of multiple readings at one particular time. This test was conducted in a laboratory where all the relevant hardware, including a rechargeable battery, a computer, a power adapter, and WiBRED, were placed on a table. Data collection was carried out based on the two operational modes (stationary mode and mobile mode) for 10 times to allow better analysis on the measured data. On each mode, two different antennas, namely Antenna 1 and Antenna 2, were used. Figure 4.3 shows the configuration of the test setup.

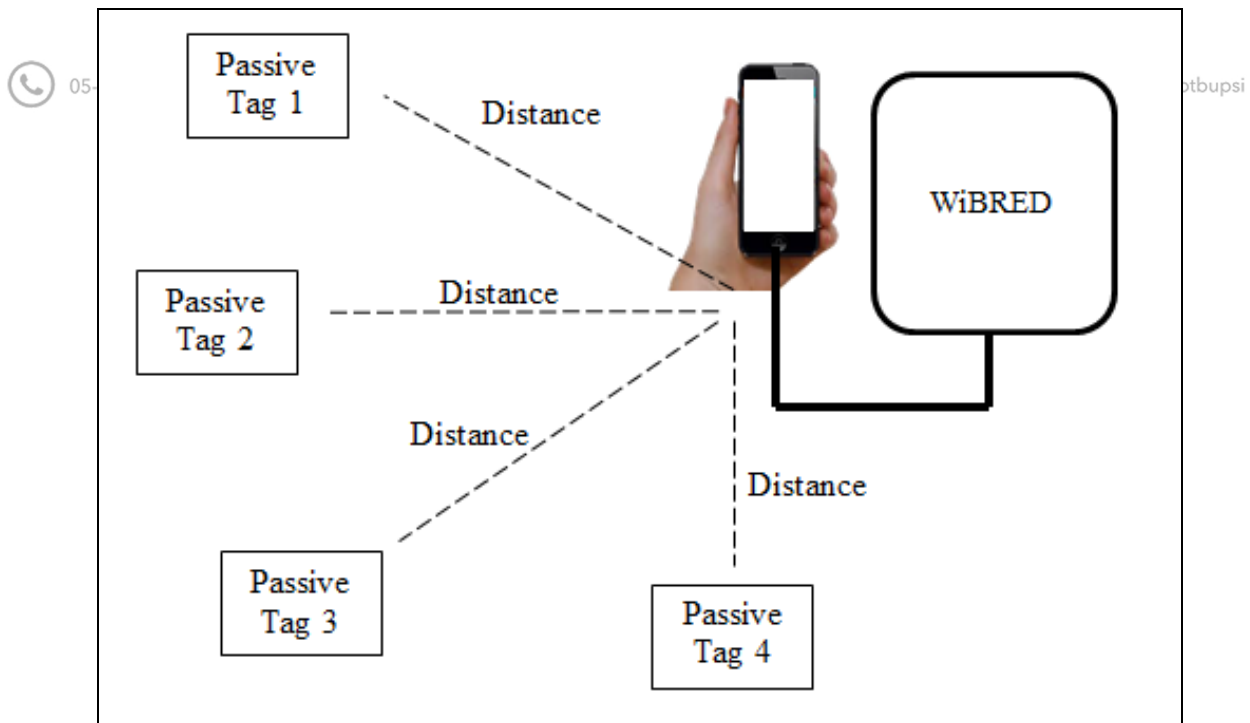


Figure 4.3. Anti-Collision Test Setup

Figure 4.4 shows two different antennas used in the test, running with the same range of frequencies from 902 to 928 MHz. However, the antennas used had different sizes, with Antenna 1 and Antenna 2 measuring 5 cm by 5 cm and 450 mm x 450mm x 75 mm, respectively. The description of antenna parameters are listed in Table 4.1

Table 4.1

Antenna Parameters

Parameters	Antenna 1	Antenna 2
Size	5 cm by 5 cm	450*450*75mm
Operating Frequency	902-928 MHZ 865MHz-868MHz	902-928 MHZ 865MHz-868MHz
Gain	5 dBi	12 dBi
Polarization	Circular	Linear

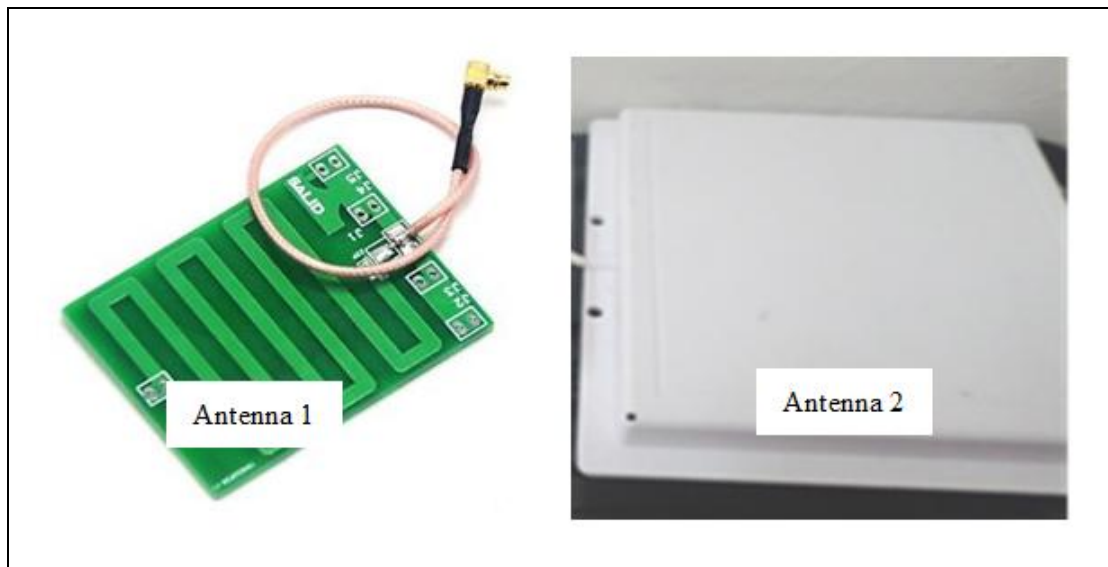


Figure 4.4. Antenna 1 And Antenna 2 Used In The Test

The percentage of the data received based on both stationary and mobile mode was calculated using Equation 4.1 as follows:

Percentage of data received:

$$= \frac{\text{number of data received}}{\text{number of tags} \times \text{number of repetitions}} \times 100\% \quad (4.1)$$

### 4.3.1 Stationary Mode

On stationary mode, the controller application on the PC was used to detect if there were any differences in the number of data received based on the distance between the tag and the reader. In this test, such distances were set to 5 cm, 10 cm, and 15 cm for Antenna 1 and to 1 m, 2 m, and 3 m for Antenna 2. The test was repeated for 10 times using 4 tags. Equation 4.1 was used to calculate the percentage of data received. Figure 4.5 shows a screenshot of the firmware that acted as a controller application, indicating that the data were randomly received.



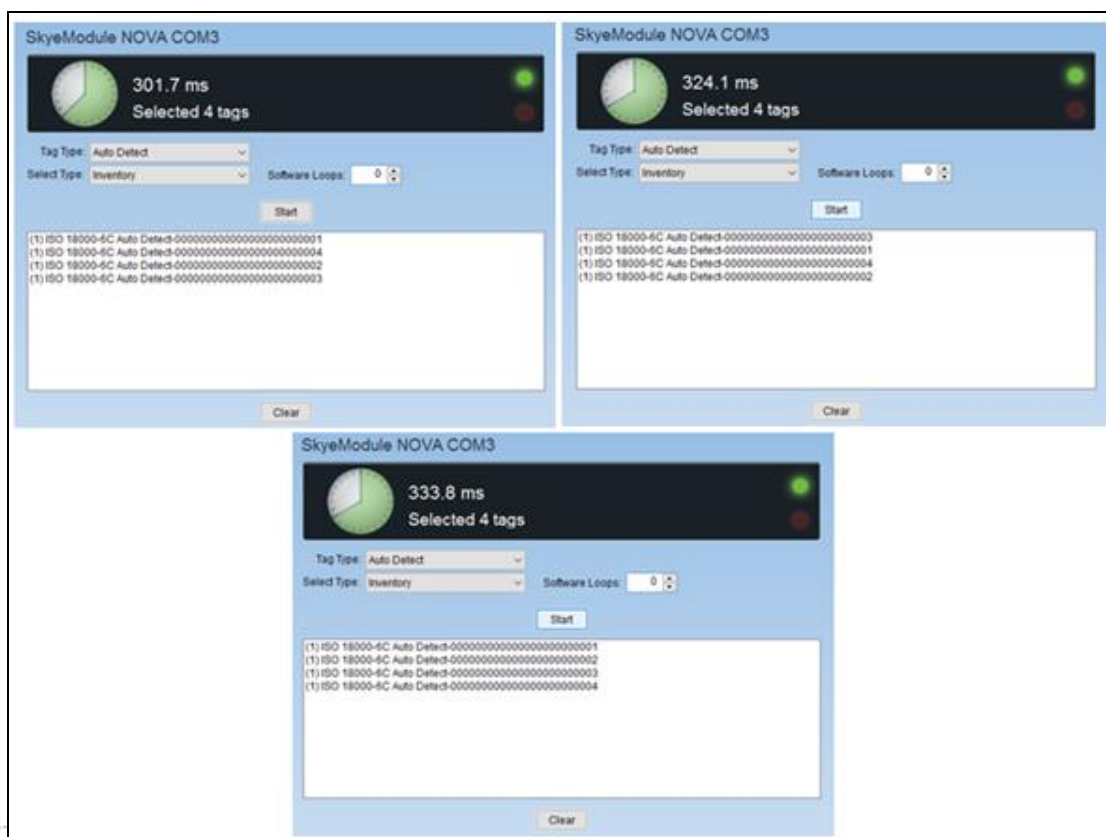


Figure 4.5. The Results Of Anti-Collision Test Based On Stationary Mode

Table 4.2.

Percentage of Data Received for Stationary Mode

Antenna	Distances	Percentage of Data Received (%)
1	5 cm	100
	10 cm	100
	15 cm	75
	1 m	100
2	2 m	95
	3 m	80

The above findings show that the efficiencies of the anti-collision test in stationary mode of Antenna 1 at 5 cm, 10 cm and 15 cm distances were 100%, 100% and 75% respectively. Antenna 1 m, 2 m, and 3 m were 100%, 95%, and 80%, respectively.

### 4.3.2 Mobile Mode

On the mobile mode, the mobile application developed and installed on the mobile phone was used. Given that no date and time were displayed when tag information was received, the time of the clock was recorded every time the test was performed, which was repeated 10 times based on a one-minute interval. Figure 4.6 shows three screenshots of the test at three different times, namely at 00:18, 00:20, and 00:21.

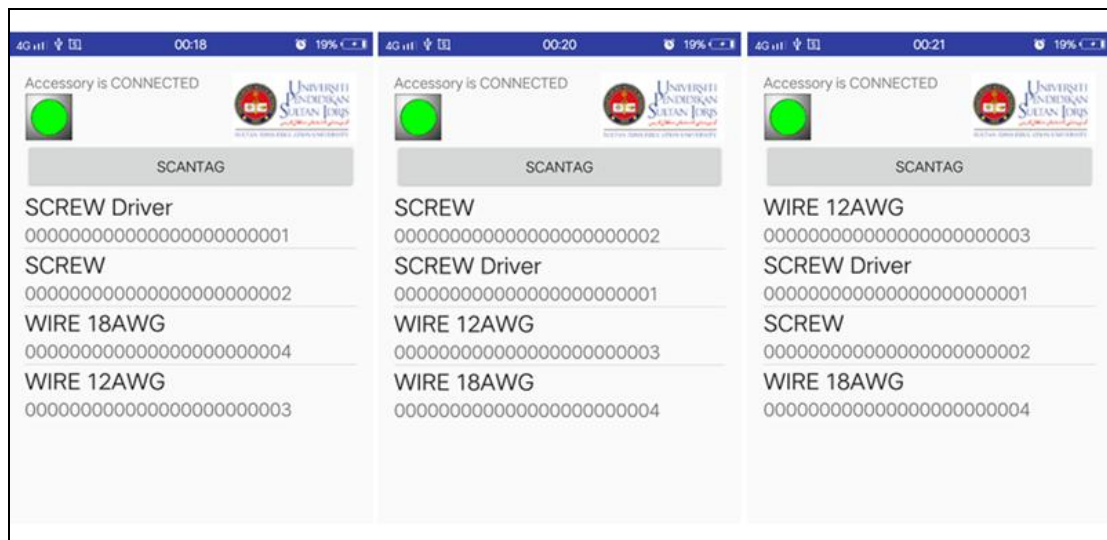


Figure 4.6. The Results Of Anti-Collision Test Based On Mobile Mode

Table 4.3

## Percentage of Data Received for Mobile Mode

<b>1</b>	<b>Distances</b>	<b>Percentage of Data Received (%)</b>
	5 cm	100
1	10 cm	100
	15 cm	77.5
	1 m	100
2	2 m	97.5
	3 m	87.5

The above findings show that the efficiencies of the anti-collision test of mobile mode of Antenna 1 at 5cm, 10cm and 15cm distances were 100%, 100% and 77.5% respectively. Antenna 1 m, 2 m, and 3 m were 100%, 97.5%, and 87.5%, respectively.

#### 4.4 Comparison of results of the Anti-Collision test

The transceiver was equipped with CSMA/CA algorithm (SKYEMODULE, 2014). Thus enabling WiBRED to perform the anti-collision test successfully. Based on Figure 4.7 and Figure 4.8, clearly there were no significant differences between the percentages of data received based on the two types of antenna used in the test.

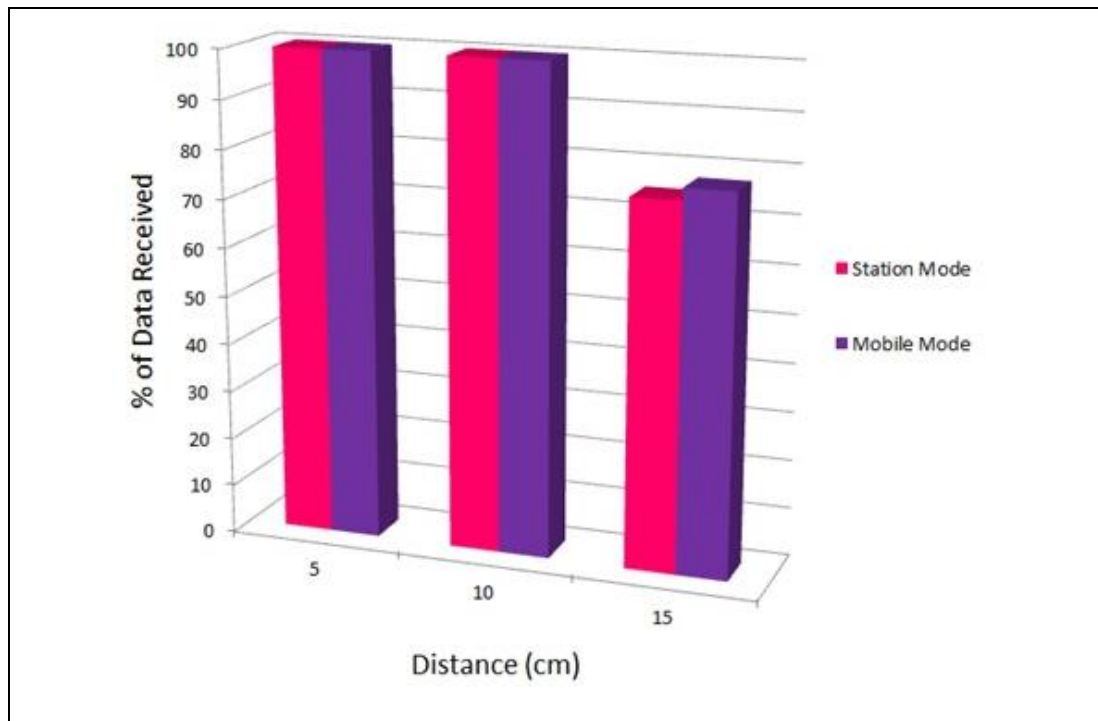


Figure 4.7. Comparison of Results Of The Anti-Collision Test Based On Antenna 1

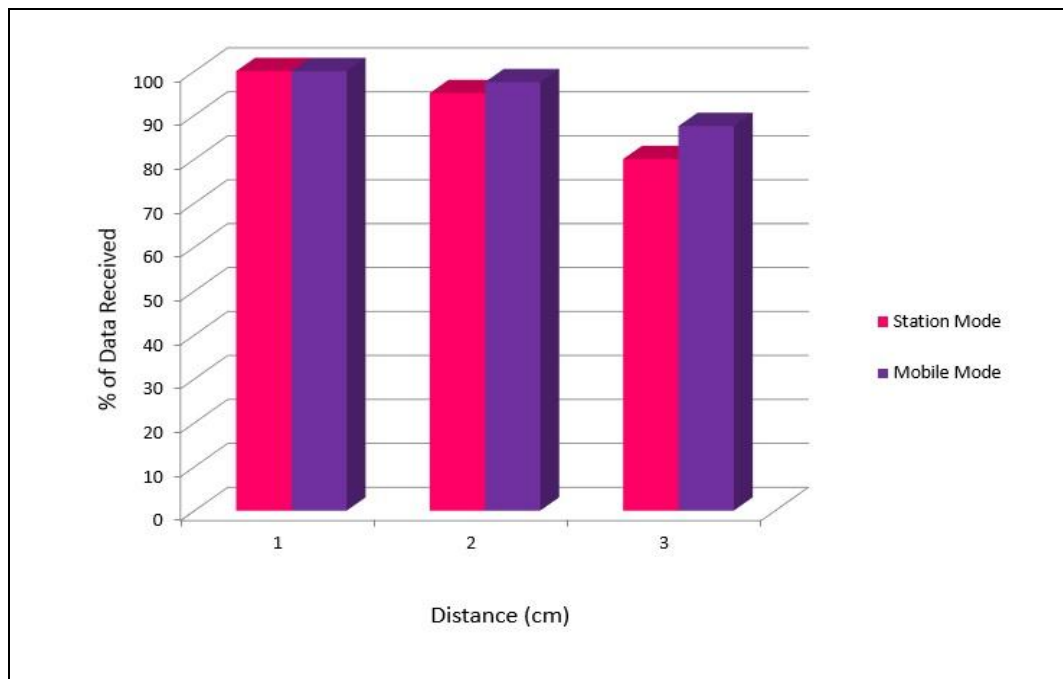


Figure 4.8. Comparison of Results Of The Anti-Collision Test Based On Antenna 2

Both graph shows that there is no significance difference between stationary and mobile mode. Hence the graphical patterns shows that as the distance increase, the efficiency of anti-collision for both modes decreases (Lindholm & Auster, 2007).

#### 4.5 Maximum Read Range Measurement

WiBRED was placed in an actual environment in which it would operate after testing. Therefore, the testing was carried out to determine the maximum readable distance over which WiBRED would operate in indoor of NLOS and line-of-sight (LOS) environments. Essentially, WiBRED supports two types of wireless communication, namely passive and active communication. As such, maximum read range was tested for both types of communication. Figure 4.9 shows the passive and active wireless communication as represented by labels “A” and “B”, respectively

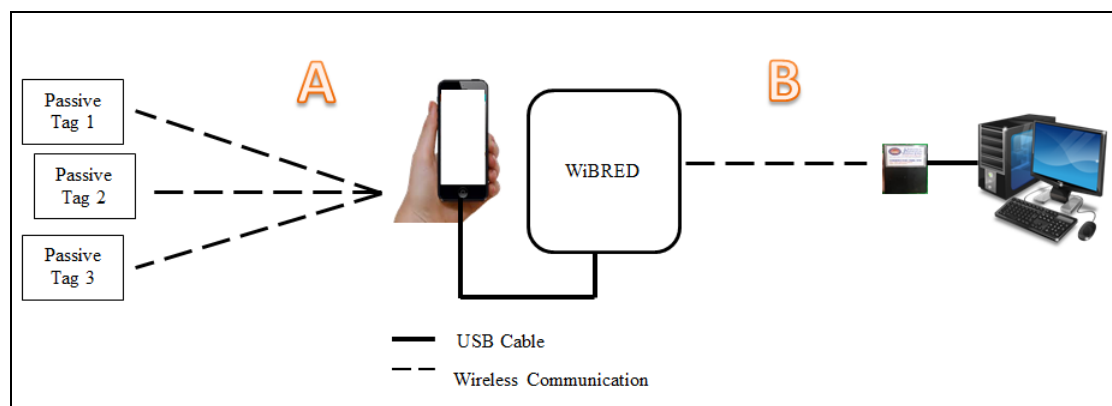


Figure 4.9. Test Setup for Read Range Measurement



The test in the indoor LOS environment was carried out inside an industrial building of a company. However, the company prohibited the shooting of photos of the production areas for security reasons. It is important to note that several factors, such as the antenna's orientation, characteristic, material, and position, would affect the accuracy of read range measurement.

Hence, the researcher adhered to the following guidelines to prevent any errors in the measurement:

- The heights of RFID tag and reader have to be the same at all time.
- The orientations of RFID tag and reader antenna must be the same at all time.
- The same antenna must be used for RFID tag and reader at all time.
- The antenna of RFID tag and reader must have the same maximum output power at all time.



#### 4.5.1 Stationary Mode

The reader was placed at a fixed location, while the tag was moved until the reader was not able to receive information from the tag. The measurement was repeated 10 times in the test. The antenna used the Ultra High Frequency RFID (UHF RFID), ranging from 902 to 928 MHz. Table 4.4 summarizes the test results of the NLOS and LOS propagations of passive and active RFID.



Table 4.4

The NLOS and LOS Propagations Based on Stationary Mode

Environment	Passive (cm)				Active (m)	
	Antenna 1 (cm)		Antenna 2 (m)		NLOS	LOS
Propagation Type	NLOS	LOS	NLOS	LOS		
Average						
Maximum Read Range	10.8	14.3	2.61	3.45	40.6	72.72

#### 4.5.2 Mobile Mode

The reader was placed at a fixed location, while the tag was moved until the reader was not able to receive information from the tag. The measurement was repeated 10 times, with the antenna operating in the UHF RFID frequencies, ranging from 902 to 928 MHz.

Table 4.5 summarizes the results of the average maximum read range for the indoor environment. Clearly, the comparison between the LOS and NLOS propagations shows that the former had longer detection range than that of the latter.

Table 4.5

The NLOS and LOS Propagations Based on Mobile Mode

Environment	Passive				Active	
	Antenna 1 (cm)		Antenna 2 (m)		NLOS	LOS
Propagation Type	NLOS	LOS	NLOS	LOS		
Average Maximum Read Range	10.7	14.6	2.60	3.50	44.80	74.54

### 4.5.3 Comparison of Station and Mobile Read Range

The following figures show the comparison between the read range measurements of NLOS and LOS based on the two modes. Figure 4.10 and Figure 4.11 show the read range measurements of the passive communication using different Antenna, while Figure 4.12 shows the comparison for read range of active, indicating that there was no significant difference in such measurements between the two modes.

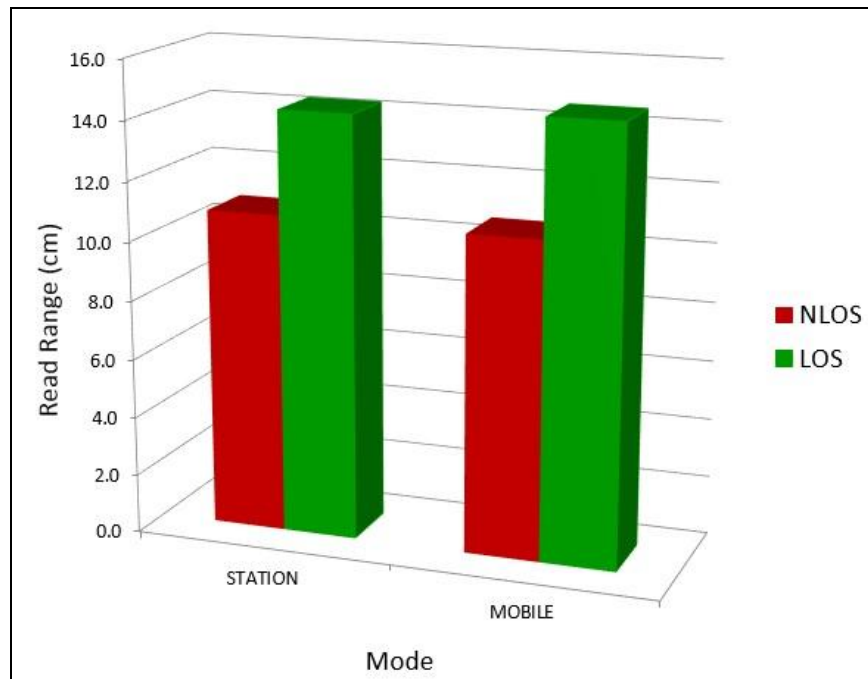


Figure 4.10 . The Read Range Result of Passive RFID using Antenna 1



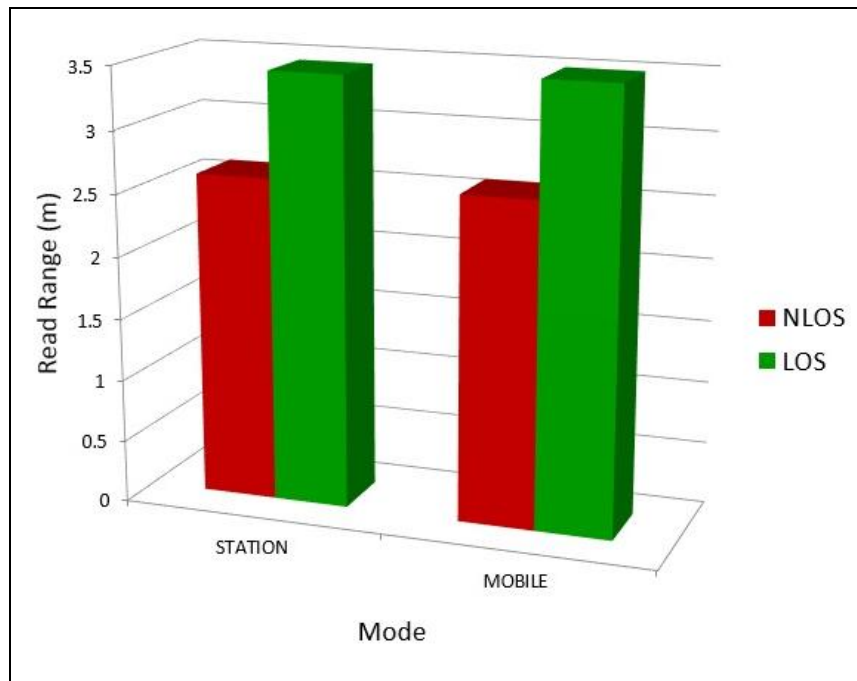


Figure 4.11. The Read Range Result of Passive RFID using Antenna 2

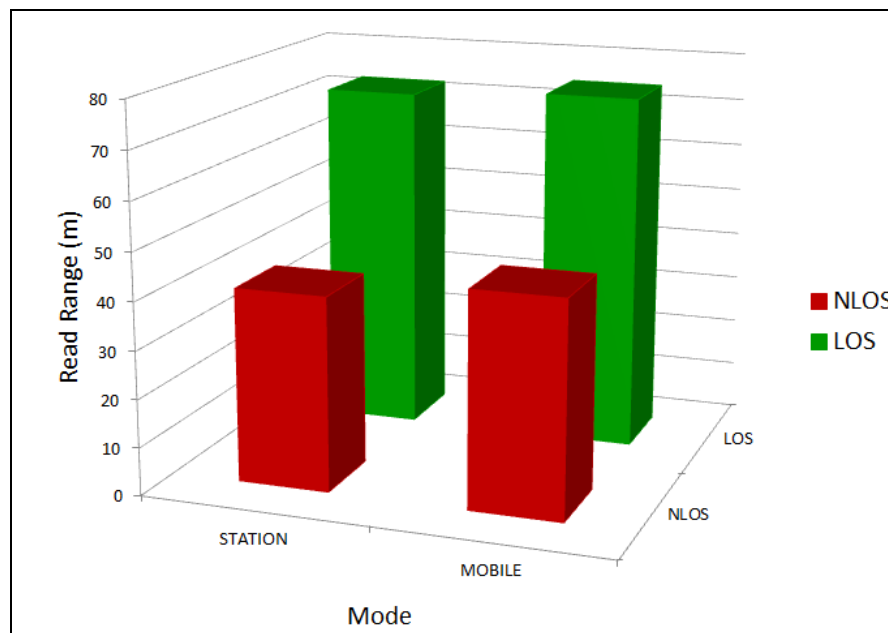


Figure 4.12. The Read Range Result of Active RFID



The findings shows that there no significance difference between stationary and mobile mode for read range measurement result. Between two propagations; NLOS and LOS shows significance difference. This is because LOS deploys when there is no obstruction between transmitter and receiver. Therefore, due to less attenuation in LOS communication, it offers good signal strength.

#### 4.6 Wireless Mesh Network Test

This section describes the test of the performance of the monitoring application in a real world environment using all the RFID components, such as WiBRED, routers, and coordinator. This test was in fact carried out in the UPSI E-learning building. The whole network of the mesh topology comprised 4-hops as illustrated in Figure 4.13. Mesh topology was employed because the router of the network would have to communicate with the farthest router, which might be located beyond the permitted distances of the router. Furthermore, such a topology only permits data packets to pass through multiple hops in a network to route data from any source in any direction in different average traffic loads.



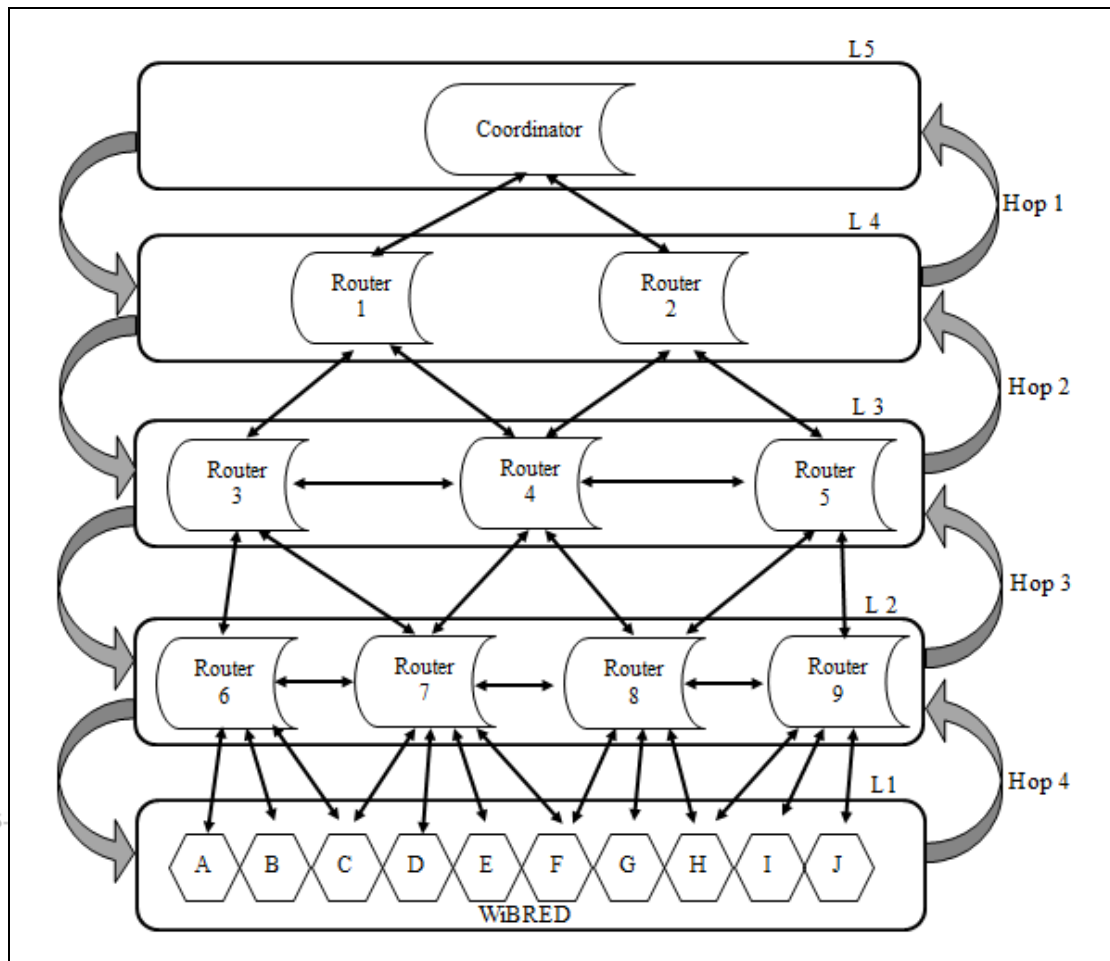


Figure 4.13. Layout of Wireless Mesh Sensor Network in Multi-Hop Environment

WMN has the capability of self-healing that guarantees network coverage at all time, despite some of its nodes may be defective or turned off. As such, the test was carried out by switching off one router at a time. WMN is similar to peer-to-peer topology where the use of an intermediate device that relays data with removed dead spots constitutes an expandable network (Eu, Tan, & Seah, 2009; Hamidian et al., 2009). The tag was passed through a series of spots, from spot A to spot J, at which 10 data were sent at a time from each spot.

Table 4.6

## Self-healing Capability of WMN

Spot	Station Mode										Mobile Mode									
	A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J
Router Closed																				
1	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
2	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
3	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
4	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
5	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
6	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
7	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
8	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
9	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

As highlighted, both modes were capable of self-healing in the wireless mesh network. Moreover, messages could be relayed across all levels (from level 1 to level 5) at every spot.

#### 4.7 Tags Collection Time

The purpose of this test was to measure the tag collection time based on stationary and mobile modes, which indicate the performance of monitoring and tracking of the mobile application. Of late, the collection of ID tags has become more imperative as it provides important information critical to the wellbeing of a manufacturing organisation. More specifically, such wireless collection of ID tags helps improve material management,

product identification, and work-in process tracking, which would be difficult to handle if manual data entry or barcode technology was used. In this study, testing was performed on several numbers of tags, namely 1, 5, 10, 15, 20, and 25 tags. Table 4.7 and Figure 4.12 show the tags collection time of several numbers of tags based on stationary and mobile modes.

Table 4.7

Tags Collection Time Based on Stationary and Mobile Modes

No. of Tags	Collection Time (ms)	
	Station	Mobile
1	136.74	104.00
5	217.96	145.90
10	261.29	154.80
15	278.48	162.80
20	321.62	170.60
25	367.47	192.50

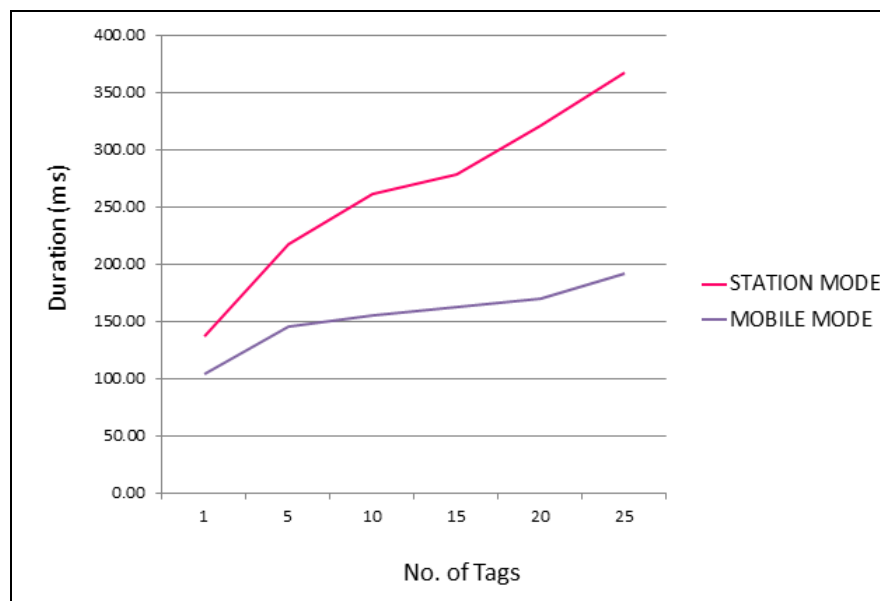


Figure 4.14. Tags Collection Time Based on Stationary and Mobile Modes



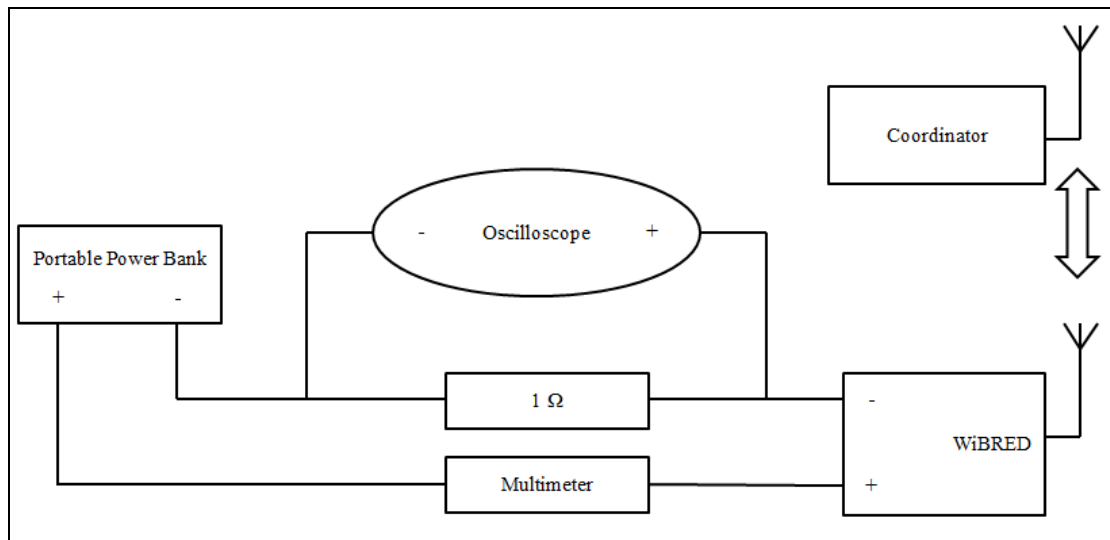
From the findings shows that stationary and mobile mode shows significant difference in tags collection time. The duration of tags collection time for mobile mode is less compared to stationary mode.

#### 4.8 Energy Analysis

This section discusses the performance of WiBRED in terms of energy consumption. One of the methods to measure current is to use a resistor across a current output and measure the voltage according to Ohm's Law (Wines & Braathen, 2008) as follows:  $V = IR$ , where  $I$  is the current flowing through a resistor of a known value. Figure 4.15 shows the setup used in the measurement of energy consumption. The current was measured using a  $1\text{-}\Omega$  resistor placed in series at the lower side of a circuit, which was from the circuit to the ground, as shown in Figure 4.15. This resistor was chosen because low resistance allows the measurement of large signal, while keeping the voltage fluctuation under control. Moreover, such small value of resistance can minimize additional voltage drop (Shinghal, Noor, Srivastava, & Singh, 2011). A digital oscilloscope was connected across the resistor to measure the current, and a multi-meter was used to measure the idle and sleep modes of the circuit. When using a multimeter to measure current of idle and sleep mode, the only way that can be used to detect the level of current flowing is to break into the circuit so that the current passes through the meter. Idle mode is measured when the system is waiting for executing task but the key components are at nominal state while



sleep mode is when the key components shuts itself down. Testing was only performed on mobile mode, because it did not have to rely on a fixed power source.



**Figure 4.15.** The Setup of the Energy Consumption Measurement (Jain & Braathen, 2011; Selvig, 2007)

Figure 4.16 shows the waveform that was captured by the oscilloscope during the scan tag process. The measurement of the waveform was repeated 10 times to obtain an average waveform of the voltage for each type of transmission during each cycling process.

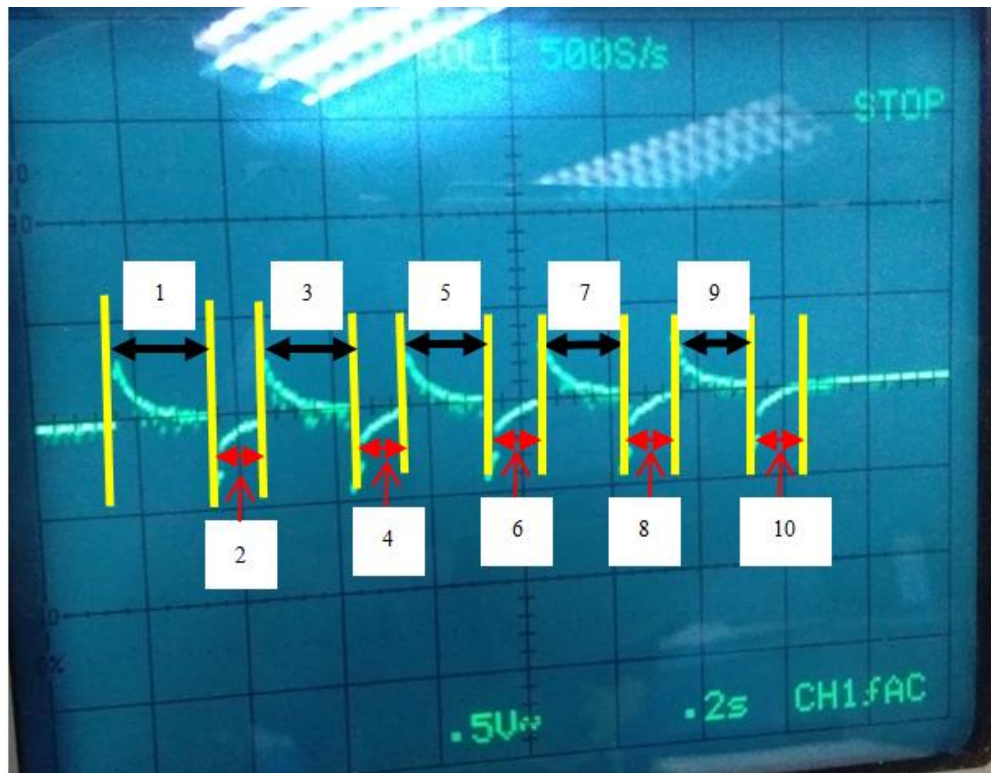


Figure 4.16. The Waveform of Scan Tag Process

Table 4.8 shows the average current consumption for the active period. The sequence were divided into different phases in which the approximate current and duration of each phase was measured. Such sequential measurements were carried out because the waveform had multiple peaks (Jain & Braathen, 2011). Table 4.8 summarizes the current consumption during the scan tag process, with each event having specific values of the current consumed and time used.



Table 4.8

## The Current Consumption during Transmission and Reception

No	Description	Current (mA)	Transmit		Receive		
			Duration (ms)	(mA)*(ms)	Current (mA)	Duration (ms)	(mA)*(ms)
1	Start Scan Tag	300	180	54000			
2	Receive ACK				400	120	48000
3	Determine Delimiter	300	180	54000			
4	Receive Delimiter				400	120	48000
5	Determine Packet Length	300	180	54000			
6	Receive Packet Length				400	120	48000
7	Determine Command	300	180	54000			
8	Receive Command				400	140	56000
9	Determine Tag ID	350	160	56000			
10	Receive Tag ID				400	120	48000
	<b>SUM</b>		<b>880</b>	<b>272000</b>		<b>620</b>	<b>248000</b>
	<b>AVERAGE CURRENT</b>	<b>309.09</b>			<b>400</b>		

## 4.8.1 Calculated Current Consumption

The current consumption was divided into four modes: sleep, transmit, receive, and measure. The sleep mode was activated when the tag was asleep and other components were in their respective save modes. This mode was not applicable to the IOIO board, because it served as a host on mobile mode. Furthermore, the IOIO board would be disconnected from the Android device if it were to be on the sleep mode. Table 4.9 shows the current consumption of WiBRED.

Table 4.9

Calculated Current Consumption per Circuit

Mode	IOIO Board (mA)	Passive reader (mA)	Wireless transceiver (mA)	Relay (mA)	Total (mA)
Sleep	-	10	0.101	40	50.101
Transmit	1000	450	40	40	1530
Receive	1000	450	50	40	1540

The theoretical total current consumption of each mode (receive, transmit, and sleep) was equal to the sum of current consumption of each device. To estimate the current consumption, it was assumed that the mobile reader sent commands every 60 seconds, while the tag listened to these commands. The reason for using a sampling period of 60 seconds is that the application of the proposed system is meant for tracking objects in manufacturing plants. Equation 4.2 was used to calculate the duration of the sleep mode as follows:

$$\begin{aligned}
 T_{sleep} &= 60 \text{ s} - T_{receive} - T_{transmit} \\
 &= (60 - 0.62 - 0.88) \text{ second} \\
 &= 58.5 \text{ second}
 \end{aligned}
 \tag{4.2}$$

The time of each mode was converted to percentage as presented in Table 4.10.

Table 4.10

Percentage of Time usage for Each Mode

<b>4356</b>	<b>Time (s)</b>	<b>Percentage of Time In 60s (%)</b>
<b>Sleep</b>	58.5	97.500
<b>Transmit</b>	0.88	1.467
<b>Receive</b>	0.62	1.033
<b>Total</b>	60	100

The average current was calculated from the usage percentage and current consumption of each mode. The time listed in Table 4.11 was multiplied with the corresponding current consumption listed in Table 4.9, and the sum of all currents gives the average current consumption (Bengtsson, 2007). Table 4.11 shows that the calculated current consumption of WiBRED in a period of 60s was 87.201 mA.

Table 4.11

The Calculated Current Consumption of the Tag in a Period of 60 S

<b>Mode</b>	<b>Percentage of Time In 60s (%)</b>	<b>Calculated Current Consumption</b>	
		<b>Total per circuit (mA)</b>	<b>% Current consumption per circuit (mA)</b>
<b>Sleep</b>	97.500	50.101	48.848
<b>Transmit</b>	1.467	1530	22.445
<b>Receive</b>	1.033	1540	15.908
<b>Total</b>	100		87.201

## 4.8.2 Measured Current Consumption

For the real-time application of the system, the measured current consumption during transmits, receive and sleeps are tabulated in Table 4.12. Clearly, the calculated and measured values would be different as all components used had some tolerances, which were more pronounced for passive components. Therefore, such differences suggest that the calculated total circuit resistance would be slightly higher than the theoretical one. The datasheet specification was based on fixed current at certain voltage range and temperature, but in real life, the voltage of a power source would decrease as current is being drawn over time.

### The Measured Current Consumption

Mode	Measured Current Consumption
	(mA)
Sleep	90.2
Transmit	309.09
Receive	400.00

Within a period of 60s, the total current consumption was calculated to be 96.611mA as shown in Table 4.13. The calculation of the current was carried out by adding the multiplied percentages of measured current consumption in the period of 60 s.

Table 4.13

The Measured Current Consumption of the Tag in a Period Of 60 S

Mode	Percentage of Time In 60s (%)	Measured Current Consumption		
		mA	[mA] * [% of Time In 60s]	% of Comparison
Sleep	97.500	90.2	87.945	91.030
Transmit	1.467	309.09	4.534	4.690
Receive	1.033	400.00	4.132	4.277
<b>Total</b>	100		96.611	100

### 4.8.3 Rechargeable Battery Lifetime Estimation

The WiBRED was equipped with a rechargeable battery with a capacity of 10,000 mAh, and the average current consumption was 96.611 mA. Therefore, the rechargeable battery lifetime estimation was calculated according to the formula as follows:

$$\frac{10000 \text{ mAh}}{96.611 \text{ mA}} = 103.5 \text{ hours} = 4.3 \text{ days}$$

### 4.9 Comparison of WiBRED with Existing Systems

Table 4.14 shows the comparison of WiBRED performance and features with those of existing systems. The comparison descriptions are the operating frequency, OS platform, power source, design flexibility, read range and functionality.

Table 4.14

## Comparison of WiBRED with Existing Systems

Hardware Name	Embedded RFID and WSN	Operating Frequency	Android Platform	Battery capacity/ supply voltage	Flexible design, adding & modification software & hardware	Operation Range	Function
MINI ME UHF RFID Reader	RFID only	902-928MHz	√	5 VDC (powered by USB)	X	5 to 50 cm	Read, Write
Audio Jack UHF RFID Portable Reader	RFID only	860-960MHz	Support iOS and Android	USB, 5V/0.5A	X	0.5 m	Read, Write
<b>WiBRED</b>	√	860-960MHz, 2.4 GHz	√	2-5.5 VDC (powered by rechargeable battery)	√	Passive: 10 cm – 3 m (flexible based on antenna used) Active: >100 m	Read, Write, Database
RFID Reader B-SL(Vhatkar & Bhole, 2010)	RFID + Wi-Fi	135 kHz, 2.4 GHz	PDA/ mobile device (J2ME emulator)	Not mention	√	10-15 cm	Read
Handheld Reader (Abdulla, 2012)	√	2.4 GHz	Android (Version 1.5 above)	Not mention	√	Indoor: 40 m Outdoor: 120 m	Read

WiBRED is embedded of RFID and WSN technology and supports Android OS platform same as handheld reader by Abdulla, 2012. However, it does not support passive RFID like other existing system. WiBRED has operating voltage range of 2-5.5 V



compared to other system with fixed voltage range. Moreover it has the flexibility of modifying the hardware and software based on application required. WiBRED also equipped with database for storing of tags descriptions.

#### 4.10 Summary

The following is the summary of the findings of the test performed in the real world environment:

- Anti-collision plays an important role in detecting multiple tags at a time to ensure packets received will not collide. As such, the transceiver module was programmed with the CSMA anti-collision algorithm. The test conducted used two types of antennas working in the frequency range of 860-960 MHz. However, they differed in size, with Antenna 1 measuring 5 cm x 5 cm and Antenna 2 measuring 25 cm x 25 cm. For Antenna 1, the anti-collision efficiencies at the distances of 5 cm, 10 cm and 15 cm were 100%, 100%, and 75% based on stationary mode. At the same distances but on the mobile mode, the anti-collision efficiencies remained the same (at 100%), except for the distance of 15 cm which saw the efficiency climb to 77.5%. The anti-collision efficiencies of Antenna 2 at the distances of 1 m, 2 m, and 3 m based on the stationary mode were 100%, 95%, and 80%. At the same distances, the anti-collision efficiencies of the same antenna were 100%, 97.5%, and 87.5%, respectively.





- The proposed system is designed to function as a tracking application for manufacturing plants. However, for security reasons, photo shooting in the production areas was prohibited. As revealed, the test findings show that the difference in the read range measurements between stationary and mobile mode was not significant. Additionally, it was observed that Antenna 1 was capable in detecting objects up to 15 cm, while Antenna 2 was capable in detecting the same object up to 3.5 m.
- The WiBRED is a flexible system capable of working effectively in any types of frequency within the range of 860-960 MHz. The working frequency of the integrated antenna depends on the specification of the tag's antenna, thus meeting the industrial needs in tracking multiple types of products.
- The proposed system was also tested on the WMN test bed consisting of several essential hardware, such as the WiBRED system, routers, and a coordinator. Given that WMN has a self-healing capability, whenever any router was switched off, the ID could still be transmitted from L1 to L5. The test was also conducted on both stationary and mobile mode, the results of which confirm that both modes can operate on the WMN platform.
- The tag collection time test was performed to measure the time of collecting tags on both stationary and mobile mode. In fact, such a test (which is important in the manufacturing industry) can help determine the performance of a monitoring and







tracking application. The analysis performed on data collected showed that the application took less time for such collection when on mobile mode compared with that of the stationary mode. Furthermore, the maximum number of tags collected was 25.

- For energy consumption test, the proposed system was only tested on mobile mode. Moreover, the sleep mode of the proposed system was set at 90.2 mA, which was quite high because the IOIO board that served as a host would remain active, not asleep. The calculation of the energy consumption was based on the assumption that the proposed system would continuously transmit data to the coordinator every 60s. The analysis performed showed that the current consumption for 60s sampling rate was 96.611 mA, effectively helping there chargeable battery to last 4.3 days. Arguably, the battery lifespan can be extended if the interval of transmission is stretched to more than an hour.





## CHAPTER 5

### CONCLUSION AND FUTURE WORK



#### 5.1 Conclusion

The proposed system, namely WiBRED, has been demonstrated to be a potent, novel system that can improve the monitoring of warehouse management and control of material flow in the manufacturing industry. What makes WiBRED innovative and effective is that this system is embedded with active and passive RFID running on the WMN platform. As such, this system can serve as a cost-effective RFID smart label solution to help manufacturing personnel to monitor and control material flow in real time. Moreover, WiBRED can be connected to an Android smart phone (running on Android OS version 1.5 and above) via IOIO-OTG, providing high mobility and flexibility for manufacturing personnel to track goods in a warehouse.





Interesting, WiBRED can operate on two modes, namely stationary and mobile mode, thus enhancing its utility in manufacturing. Equipped with this ability, this system can help conserve the power of a rechargeable battery when running on mobile mode. Equally interesting is that connecting an Android smart phone to a USB IOIO cable will switch the system from stationary mode to mobile mode – which will occur spontaneously and automatically. WiBRED will be on stationary mode when it remains fixed (or stationary) and disconnected from the Android smart phone. On this mode, a portable power bank or a fixed power source can be used to charge the system.

As revealed, the active reader of WiBRED is able to detect manufacturing objects up to 70 m, which seems adequate for tracking purposes in large manufacturing plants. More revealingly, the read range of the active reader can be extended further by adding routers, because WiBRED can perform effectively on the WMN platform and has the capability of self-healing when one router is switched off at a time from level 1 to level 5.

On the other hand, the passive reader of WiBRED has been programmed with a CSMA anti-collision algorithm, providing the system with some flexibility to work with any types of antenna within the frequency range of 860-960 MHz. For both modes of the WiBRED, the antennas will be able to detect multiple tags over some distances of up to 15 cm and to 3.5 m using Antenna 1 and Antenna 2, respectively. Obviously, the appropriate type of antenna to be used will depend on the specifications of tag antenna and the type of tagged products used in the manufacturing industry.





Equally revealing is that the tag collection time for both modes differed quite significantly, with mobile mode taking less time to detect tags compared to that of stationary mode. As such, only the mobile mode of the WiBRED was tested for power consumption, indicating that the power consumption of the whole circuit of WiBRED was 96.11 mA based on a transmission sampling rate of 60s.

## 5.2 Future Work

Arguably, in its current form, WiBRED has several drawbacks, which can be addressed in future research. As such, future studies should focus on the following recommendations:



1. Develop the cloud computing so that companies can either reduce or eliminate the size of their own data centres. With reduction of numbers of servers, can reduce the software cost and the number of staff without impacting the organization's IT capabilities.
2. Make improvements to existing smart system that will enable it to pinpoint the locations of tracked objects with higher precision.





## REFERENCES

Abdulla, R. M. T. (2012). *Design And Implementation Of Multi-Hop Active RFID System With Embedded Wireless Sensor Network*. Universiti Sains Malaysia.

Abdullah, S., Ismail, W., & Abdul, Z. (2015). Implementation of Wireless RFID for Production Line Management System in a Real Environment. *Wireless Personal Communications*, 83(4), 3119–3132.

Abinayaa, V., & Jayan, A. (2014). Case Study on Comparison of Wireless Technologies in Industrial Applications. *International Journal of Scientific and Research Publications*, 4(1), 2250–3153.

Aju, O. G. (2015). A Survey of ZigBee Wireless Sensor Network Technology: Topology, Applications and Challenges. *International Journal of Computer Applications*, 130(9), 975–8887.



Akshita Dubey, Amisha Jain, & Aditi Mantri. (2015). COMPARATIVE STUDY: WATERFALL V/S AGILE MODEL. *International Journal of Engineering Sciences & Research Technology (IJESRT)*, 4(1), 70–75.

Al-Harbawi, M., Rasid, M. F. A., & Noordin, N. K. (2009). Improved Tree Routing (ImpTR) Protocol for ZigBee Network. *IJCSNS International Journal of Computer Science and Network Security*, 9(10).

Anamika Vatsal, E., & Fatima, M. (n.d.). Impact of Frequency Offset on Interference between Zigbee and Wifi for Smart Grid Applications. *IOSR Journal of Electronics and Communication Engineering*, 7(5), 2278–8735.

Attaran, M. (2012). Critical Success Factors and Challenges of Implementing RFID in Supply Chain Management. *Supply Chain and Operations Management*, 10(1), 144–167.

Bachelor, B. (2014). *RFID news roundup : A Summary of Tags Comprise Half of Global RFID Market*.

Bal, M. (2014). An industrial Wireless Sensor Networks framework for production monitoring. In *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)* (pp. 1442–1447). IEEE.

Bala, K., Sharma, S., & Kaur, G. (2015). A Study on Smartphone based Operating System. *International Journal of Computer Applications*, 121(1), 17–22.





- Barr, M., & Massa, A. (2009). *Programming Embedded Systems* (2nd ed.). O'Reilly Media.
- Bazard, M., & Bhardwaj, S. (2011). Overview on Android – The New Mobile Operating System. *SIGI Reflections-International Journal of Science, Technology and Management.*, 2(1), 25–34.
- Ben-Tsvi, Y. (2014). Power Supply OTG.
- Bengtsson, N. (2007). *Development and Implementation of a Low Power Wireless Sensor Network*. Linköpings University.
- BlackBerry. (n.d.). BlackBerry 10 device architecture. Retrieved August 7, 2017, from <https://help.blackberry.com/en/blackberry-security-overview/latest/blackberry-security-overview/awi1402929620791.html>
- Deavours, D. D. (2005). *UHF EPC Tag Performance Evaluation A Production of the Report #2 in the Series*.
- Deloitte. (2015). *Industry 4.0: Challenges and solutions for the digital transformation and use of exponential technologies*. Zurich. Retrieved from <https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry-4-0-24102014.pdf>
- Dharmistha, M., & Vishwakarma, D. (2012). IEEE 802.15.4 and ZigBee: A Conceptual Study. *International Journal of Advanced Research in Computer and Communication Engineering*, 1(7).
- Divyap, K., & Venkata Krishnakumar, S. (2016). Comparative Analysis Of Smart Phone Operating Systems Android, Apple iOS And Windows. *International Journal of Scientific Engineering and Applied Science*, (22), 2395–3470.
- Dr. Reinhard Geissbauer, Vedso, J., & Schrauf, S. (2016). *2016 Global Industry 4.0 Survey What we mean by Industry 4.0 / Survey key findings / Blueprint for digital success*. Retrieved from [www.pwc.com/industry40](http://www.pwc.com/industry40)
- Duroc, Y., & Kaddour, D. (2012). RFID potential impacts and future evolution for green projects. *Energy Procedia*, 18, 91–98.
- Eu, Z. A., Tan, H.-P., & Seah, W. K. G. (2009). Routing and relay node placement in wireless sensor networks powered by ambient energy harvesting. In *Proceeding*





*WCNC'09 Proceedings of the 2009 IEEE conference on Wireless Communications & Networking Conference* (pp. 2003–2008).

Francisco Almada-Lobo. (2015). The Industry 4.0 revolution and the future of Manufacturing Execution Systems (MES). *Journal of Innovation Management*, 4(3), 16–21.

Gaikwad, P. P., Gabhane, J. P., & Professor, A. (2015). International Journal on Recent and Innovation Trends in Computing and Communication A Review on IOT Techniques for Automating Devices. *International Journal on Recent and Innovation Trends in Computing and Communication*, 3(4), 1773–1777.

Gandhewar, N., & Sheikh, R. (2010). Google Android: An Emerging Software Platform For Mobile Devices. *International Journal on Computer Science and Engineering (IJCSE)*, 1(1), 12–17.

Hakala, M. (2013). *White Paper : A comparison of Real-Time Location Systems (RTLS) and Technologies*. Ekahau.Inc (Vol. 1).

Hall, S. P., & Anderson, E. (2009). Operating systems for mobile computing. *Journal of Computing Sciences in Colleges*, 25(2), 64–71.



Hamidian, A., Palazzi, C. E., Chong, T. Y., Navarro, J. M., Körner, U., & Gerla, M. (2009). Deployment and Evaluation of a Wireless Mesh Network. In *Advances in Mesh Networks, 2009. MESH 2009. Second International Conference on* (pp. 66–72). Athens, Glyfada, Greece: IEEE.

Harmon, D. (2007). Enabling high-speed USB OTG functionality on TI DSPs.

Huang, G. Q., Saygin, C., & Dai, Q. Y. (2012). Special issue on “RFID-Enabled Manufacturing: Insights and Lessons from Industrial Cases.” *International Journal of Computer Integrated Manufacturing*, 25(1), 1–2.

Idris, Y., & Muhammad, N. A. (2016). A Comparative Study of Wireless Communication Protocols: Zigbee vs Bluetooth. *International Journal of Engineering Science and Computing*, 6(4), 3741–3744.

Jain, S., & Braathen, M. (2011). Measuring the Power Consumption on CC2530ZNP Using CC2530 ZNP Mini Kit.

James, C. C., Cheng, C.-H., & Huang, P. B. (2012). Supply chain management with lean production and RFID application: A case study. *ELSEVIER*, 40(9), 3389–3397.

Jia, X., Feng, Q., Fan, T., & Lei, Q. (2012). RFID technology and its applications in Internet of Things (IoT). In *2012 2nd International Conference on Consumer*





*Electronics, Communications and Networks (CECNet)* (pp. 1282–1285). IEEE.

Jindal, G., & Munjal, S. (2012). The Wane of Dominant (Symbian Operating System). *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(9), 2277–128.

Joseph, J., Professor, A., Kurian, S. K., Manager, P., & Mahindra, T. (2013). Mobile OS – Comparative Study. *Journal of Engineering, Computers & Applied Sciences (JEC&AS) ISSN*, 2(10), 2319–5606.

Jou, Y., Wee, H., & Chen, H. (2009). A neural network forecasting model for consumable parts in semiconductor manufacturing. *Emerald Insight*, 99(3), 296–227.

Khanna, G., & Gupta, G. (2013). A Review on Techniques for Establishing Coexistence between WiFi and Zigbee. *International Journal of Science and Research (IJSR) ISSN (Online Index Copernicus Value Impact Factor*, 14(7), 2319–7064.

Kim, H., & Ayurzana, O. (2009). Improvement of data receive ratio in remote water meter system by upgrading sensor. *International Journal of Control, Automation and Systems*, 7(1), 145–150.



Kumar Maji, A., Hao, K., Sultana, S., & Bagchi, S. (2010). Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian. In *Proceeding ISSRE '10 Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering* (pp. 249–258).

Lee, J.-S., Su, Y.-W., & Shen, C.-C. (2007). A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society* (pp. 46–51). IEEE.

Li, C., Wang, Y., & Guo, X. (2010). The Application Research of Wireless Sensor Network Based on ZigBee. In *2010 Second International Conference on Multimedia and Information Technology* (Vol. 2, pp. 89–92). IEEE.

Li, L. (2007). *Symbian OS Architecture*.

Lindholm, J., & Auster, P. J. (2007). Site fidelity and movement of adult Atlantic cod *Gadus morhua* at deep boulder reefs in the western Gulf of Maine, USA. *Marine Ecology Progress Series*, 342, 239–247. Retrieved from <http://www.int-res.com/abstracts/meps/v342/p239-247/>

Liukkonen, M., Havia, E., & Hiltunen, Y. (2012). Computational intelligence in mass







soldering of electronics – A survey. *Expert Systems with Applications*, 39(10), 9928–9937.

Lokhande, S. E., Bamnote, G. R., Ingle, P. S., & Dharkar, S. D. (2014). SUPER SPEED DATA TRAVELLER USB 3.0. *International Journal of Electronics, Communication & Soft Computing Science and Engineering*, 2(6), 2277–9477.

Martinez-Conde, S., & Macknik, S. L. (2014, October 16). How the Color Red Influences Our Behavior. *Scientific American Mind*, 25(6), 21–23.

Matsubayashi, M., & Hiroshi, W. (2003). *Mechanism of a Factory with Illustrations*. (W. Waku, Ed.). Waku Consulting Co. Ltd.

Matumo, M., & Kisangiri, M. (2014). Design of a Power Quality Monitoring System Based on IOIO and Android Application. *Network and Complex Systems*, 4(4), 36–46.

Medavarapu, C. (2004). *An Architecture For Embedded System Communication*. Louisiana State University.

Mika, L. (2014). RFID technology in manufacturing and supply chain. *International Journal of Computer Integrated Manufacturing*, 28(8), 861–880.

Nagarajan, R., & Dhanasekaran, R. (2015). Analysing the Effect of Interference in Wireless Industrial Automation System (WIAS), 10(6).

Narayanan, L., Muthumanickam, D. ., & Nagappan, D. . (2015). Animal Health Monitoring System Using Raspberry Pi and Wireless Sensor. *International Journal of Scientific Research and Education*, 3(5), 3386–3392.

Netalkar, P. P., Kaushal, Y., Shekar, N., & Shet, V. (2014). Zigbee Based Wireless Sensor Networks for Smart Campus. *International Journal of Modern Engineering Research*, 4(7), 55–62.

North, P. (2013). *How to Better Monitor Work Floor Performance, Streamline Tracking Processes and Obtain Accurate, Up-to-the-minute Data (Increasing Efficiency and Accuracy of Production Labor Costs) The Challenges of Tracking Shop Floor Labor*. Retrieved from [www.points-north.com](http://www.points-north.com)

Okediran O, O., Arulogun O, T., & Ganiyu R, A. (2014). Mobile Operating Systems and Application Development Platforms: A Survey. *J. of Advancement in Engineering and Technology*, 1(4).





- Padhya, B., Desai, P., Pawade, D., & Student, B. (2007). Comparison of Mobile Operating Systems. *International Journal of Innovative Research in Computer and Communication Engineering (An ISO Certified Organization)*, 3297(8), 15281–15286.
- Patel, S. K., Patel, H. R., & Patel, R. S. (2011). Bluetooth usage with Architecture view & security measures. *International Journal of P2P Network Trends and Technology*, 1(3), 30–35.
- Porkodi, R., & Bhuvanewari, V. (2014). The Internet of Things (IoT) Applications and Communication Enabling Technology Standards: An Overview. In *2014 International Conference on Intelligent Computing Applications* (pp. 324–329). IEEE.
- Remple, T. B. (2003). USB on-the-go interface for portable devices. In *2003 IEEE International Conference on Consumer Electronics, 2003. ICCE*. (pp. 8–9). IEEE.
- Richards, G. (2014). *Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse*. Kogan Page.
- Rojatkar, D. V., Jengathe, G. M., Khairnar, A. B., & Lengure, S. A. (2016). Android Application Development Software – Android Studio And Eclipse. *International Journal For Engineering Applications and Technology*, 9–12.
- Ruankaew, T., & Williams, P. (2013). The Impact of Inventory Inaccuracy in the Food Manufacturing Industry: A Case Study. *Business Management Dynamics*, 2(10), 28–34.
- Saad, C., Cheikh, E. A., Mostafa, B., & Abderrahmane, H. (2014). Comparative Performance Analysis of Wireless Communication Protocols for Intelligent Sensors and Their Applications. *IJACSA) International Journal of Advanced Computer Science and Applications*, 5(4).
- Salahinejad, M., & Aflaki, F. (2007). Uncertainty Measurement of Weighing Results from an Electronic Analytical Balance. *Measurement Science Review*, 7(6).
- Sangiovanni-Vincentelli, A., Zeng, H., Natale, M. Di, & Marwedel, P. (2013). *Embedded Systems Development: From Functional Models to Implementations*. Springer Science & Business Media.
- Saxena, A., & Upadhyay, P. (2016). Waterfall vs. Prototype: Comparative Study of SDLC. *Imperial Journal of Interdisciplinary Research*, 2(6), 2454–1362.
- Selvig, B. (2007). Measuring power consumption with CC2430 & Z-Stack.





- Shinghal, K., Noor, A., Srivastava, N., & Singh, R. (2011). Power Measurements Of Wireless Sensor Networks Node. *International Journal of Computer Engineering & Science* ©IJCES ISSN, 1(1), 2231–6590.
- Singh, A., Sharma, S., & Singh, S. (2016). Android Application Development using Android Studio and PHP Framework. *International Journal of Computer Applications Recent Trends in Future Prospective in Engineering & Management Technology*, 975–8887.
- SKYEMODULE. (2014). SKYEMODULE NOVA DATASHEET, 48.
- Srivastava, N. (2006). RFID Introduction, Present and Future applications and Security Implications. *Electrical Engineering Department, George Mason University, 4400 University Drive, Fairfax, Virginia 22030, (C)*, 1–10.
- Statista. (n.d.). • Malaysia - types of smartphone OS used for personal purposes | Survey 2016. Retrieved April 18, 2018, from <https://www.statista.com/statistics/563660/malaysia-types-of-smartphone-operating-systems-used-for-personal-purposes/>
- Sulaiman, S., Umar, U. A., Tang, S. H., & Fatchurrohman, N. (2012). Application of Radio Frequency Identification (RFID) in Manufacturing in Malaysia. In *International Conference on Advances Sciences and Contemporary Engineering 2012 (ICASCE 2012)* (Vol. 50, pp. 697–706).
- Syafeeq, M., Shazli, A., & Daud, S. M. (2015). Vehicle Mobile Detection in a Building Parking Area using Smart- Phone Bluetooth Signal Technology. *Open International Journal of Informatics (OIJ) Iss*, 3(2).
- Tomai, N., & Toma, C. (2009). Issues in WiFi Networks. *Journal of Mobile, Embedded and Distributed Systems*, 1(1), 3–12.
- Turcu, C. (2011). *Deploying RFID - Challenges, Solutions, and Open Issues*. InTech Open (2nd ed.). Rijeka, Croatia: InTech.
- Turcu, C., Turcu, C., & Graur, A. (2008). Improvement of Supply Chain Performances Using RFID Technology. *Stefan Cel Mare University of Sueava, Romania*, (December).
- Umare, M. K. B., & Padole, D. D. V. (2015). The Design and Implementation of Handheld Multipurpose Scope Using Bluetooth IOIO Board. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 4(4).



Verma, M., Singh, S., & Kaur, B. (2015). An Overview of Bluetooth Technology and its Communication Applications. *International Journal of Current Engineering and Technology E International Journal of Current Engineering and Technology*, 55(33), 2277–4106.

Vhatkar, K. N., & Bhole, G. . (2010). Internal Location Based System for Mobile Devices Using Passive RFID and Wireless Technology. In *Computer Technology Department, Veermata Jijabai Technology Institute, Mumbai, India*.

W.Fisher, M. (2013). *Characteristics Of Simulated Zigbee Networks*. Southern Connecticut State University.

Wang, C., Duan, W., Ma, J., & Wang, C. (2011). The research of Android System architecture and application programming. In *Proceedings of 2011 International Conference on Computer Science and Network Technology* (Vol. 2, pp. 785–790). IEEE.

Welch, B., Jones, K., & Hobbs, J. (2003). *Practical Programming in Tcl and Tk* (4th Editio). Prentice all.

Wines, M., & Braathen, M. (2008). Measuring the Power Consumption on eZ430-RF2480.

Zare Mehrjerdi, Y. (2011). RFID and its benefits: a multiple case analysis. *Assembly Automation*, 31(3), 251–262.

## APPENDICES

### A Source Code of *MainActivity.java*

```
import android.app.DialogFragment;
import android.app.Fragment;
import android.app.FragmentTransaction;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.ref.WeakReference;
import java.util.ArrayList;
import java.util.Objects;

import ioio.lib.api.DigitalOutput;
import ioio.lib.api.IOIO;
import ioio.lib.api.Uart;
import ioio.lib.api.exception.ConnectionLostException;
import ioio.lib.util.BaseIOIOLooper;
import ioio.lib.util.IOIOLooper;
import ioio.lib.util.android.IOIOActivity;

public class MainActivity extends IOIOActivity {
    //Layout resource
    TextView tv_connected_;
    Button bt_scanTag_;
    ListView lv_scannedItem_;
    ImageView iv_led_;

    static final char[] scantag =
    {0x02,0x00,0x08,0x00,0x22,0x01,0x01,0x82,0x00,0x51,0xEE}; //
    static final char[] autotune_packet =
    {0x02,0x00,0x06,0x00,0x20,0x15,0x03,0xE1,0x11};
    int scan_loop;

    ArrayList<User>list = new ArrayList<>();
    ArrayList<String>list_id = new ArrayList<>();
    UserAdapter adapter = null;

    private static MyHandler mRFIDProgressHandler;

    //Local variable
    byte state;
    static final byte ST_SCANTAG = 1;
    static final byte ST_REQ_TAGINFO = 2;
    static final byte ST_WAIT_TAGINFO = 3;
    static final byte ST_UPDATELIST = 4;
    static final byte ST_SEND_SCANTAG = 5;
```

bupsi



```

static final byte ST_DONE = 6;

byte state_descript;
static final byte ST_GETTAGDETAILS = 1;
static final byte ST_WAITTAGDETAILS = 2;
static final byte ST_UPDATETAGDETAILS = 3;
static final byte ST_DONE_DESCR = 4;

    User details_tag_item;
int count_taginfo_request;
long task_endTime;

boolean send_auto_tune;
boolean autoTuneDone = false;
boolean ioioConnected = false;
    DialogFragment autoTuneFragment;
    DialogFragment autoTuneFragmentComplete;
    DialogFragment tagdetailsFragment;

    FragmentTransaction ft;

    String tag_description;

public static final String TAG = "MobileRFIDReaderTASK";

private static class MyHandler extends Handler {           //MyHandler :
static class .... Handler : object which either can be service or activity
private final WeakReference<MainActivity>mActivity;

public MyHandler (MainActivity activity) {
mActivity = new WeakReference<>(activity);
    }

```



```

@Override
public void handleMessage(Message msg) {
    MainActivity activity = mActivity.get();
    if (activity != null) {
        // ...
        switch (msg.what) {
            case Uart_ContrRD_RFID_Thread.SET_DATA:
                if (activity.state == ST_SCANTAG) {
                    String text = (String) msg.obj;
                    if (!activity.list_id.contains(text)) {
                        activity.list_id.add(text);
                    }
                    //activity.list.add(new User(text,"NoText"));
                    Log.d(TAG,"ADD ITEM");
                }
                break;
            case Uart_ContrRD_RFID_Thread.SCANCOMPLETE:
                activity.count_taginfo_request =
                    activity.list_id.size();
                Log.d(TAG,"LIST ID COUNT -> " +
                    activity.count_taginfo_request);
                //activity.count_taginfo_request = activity.list.size();
                if (activity.scan_loop == 4) {
                    if (activity.count_taginfo_request < 1){
                        activity.list.clear();
                        activity.list.add(new User("ScanNothing",
                            "ScanNothing"));

                        activity.state = ST_UPDATELIST;
                        Log.d(TAG,"SCAN COMPLETE with NO ITEM");
                    }else {
                        activity.state = ST_REQ_TAGINFO;
                    }
                }
                //activity.state = ST_UPDATELIST;
                Log.d(TAG,"SCAN COMPLETE with ITEM");
            }
}

```





```

        }else {
            activity.state = ST_SEND_SCANTAG;
        }
    }
    break;
    case Uart_ContrD_RFID_Thread.SCANFAIL:
        activity.list.clear();
        activity.list.add(new User("ScanFAIL", "ScanFAIL"));
        activity.state = ST_UPDATELIST;
    break;

    case Uart_ContrD_TAGINFO_Thread.SET_TAGINFO:
    if (activity.state == ST_WAIT_TAGINFO) {
        User tagData = (User) msg.obj;
        if (activity.list_id.contains(tagData.id)) {
            activity.list.add(tagData);
            activity.state = ST_REQ_TAGINFO;
        }
        --activity.count_taginfo_request;
    if (activity.count_taginfo_request == 0)
        activity.state = ST_UPDATELIST;
    }
    break;
    case Uart_ContrD_RFID_Thread.AUTOTUNE_COMPLETE:
        activity.autoTuneDone = true;
    break;
    case Uart_ContrD_TAGINFO_Thread.SET_TAGDETAILS:
    if (activity.state_descript == ST_WAITTAGDETAILS) {
        User tagDetails = (User) msg.obj;
        if (Objects.equals(activity.details_tag_item.id, tagDetails.id)) {
            activity.tag_description = tagDetails.name;
        } else {
            activity.tag_description = "Unkown Replied
TAG ID";
        }
        activity.state_descript = ST_UPDATETAGDETAILS;
    }
    break;
}
super.handleMessage(msg); //change to default handleMessage
}
}

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

// Setup layout resource
tv_connected_ = (TextView) findViewById(R.id.tv_connected);
bt_scanTag_ = (Button) findViewById(R.id.bt_scanTag);
lv_scannedItem_ = (ListView) findViewById(R.id.lv_scannedItem);
iv_led_ = (ImageView) findViewById(R.id.iv_led);

// Create the adapter to convert the array to views
adapter = new UserAdapter(this, list);
// Attach the adapter to a ListView
lv_scannedItem_.setAdapter(adapter);

lv_scannedItem_.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
@Override
public void onItemClick(AdapterView<?> parent, final View view,
int position, long id) {
if (state_descript == ST_DONE_DESCR) {
state_descript = ST_GETTAGDETAILS;
details_tag_item = (User) parent.getItemAtPosition(position);
}
}
}
}

```







```

    task_endTime = System.currentTimeMillis() + 3000;
        Log.d(TAG, "CurrentTime = " + System.currentTimeMillis()
+ "; EndTime = " + task_endTime);
    } else {
        toast("Preparing in progress...");
    }
    });

mRFIDProgressHandler = new MyHandler(this);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    //handle item selection
    switch (item.getItemId()){
    case R.id.auto_tune:
        send_auto_tune = true;
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    MenuItem item = menu.findItem(R.id.auto_tune);
    if (ioioConnected)
        item.setEnabled(true);
    else
        item.setEnabled(false);
    return true;
}

/**
 * This is the thread on which all the IOIO activity happens. It will be
run
 * every time the application is resumed and aborted when it is paused.
The
 * method setup() will be called right after a connection with the IOIO
has
 * been established (which might happen several times!). Then, loop()
will
 * be called repetitively until the IOIO gets disconnected.
 */
class Looper extends BaseIOIOLooper implements View.OnClickListener { //
    /** The on-board LED. */
    private DigitalOutput led_;
    private DigitalOutput relay_;
    private Uart uart;
    private Uart uart2;
    private InputStream in_; //the input stream defined for the uart
connection
    private OutputStream out_;
    private InputStream in2_; //the input stream defined for the uart
connection
    private OutputStream out2_;

    private Uart_ContRD_RFID_Thread workers_;
    private Uart_ContRD_TAGINFO_Thread workers2_;

```







```

/**
 * Called every time a connection with IOIO has been established.
 * Typically used to open pins.
 *
 * @throws ConnectionLostException
 *         When IOIO connection is lost.
 *
 * // @see ioio.lib.util.IOIOLooper# setup()
 */
@Override
protected void setup() throws ConnectionLostException {
    int pin_RX1_ = 10;
    int pin_TX1_ = 11;
    int pin_RX2_ = 13;
    int pin_TX2_ = 14;

    showVersions(ioio_, "IOIO connected!");
    try {
        bt_scanTag_.setOnClickListener(this);

        led_ = ioio_.openDigitalOutput(0, true);
        led_.write(Boolean.FALSE);
        relay_ = ioio_.openDigitalOutput(1, true);
        relay_.write(Boolean.TRUE);
        uart = ioio_.openUart(pin_RX1_, pin_TX1_, 38400, Uart.Parity.NONE,
            Uart.StopBits.ONE);
        uart2 = ioio_.openUart(pin_RX2_, pin_TX2_, 38400, Uart.Parity.NONE,
            Uart.StopBits.ONE);
        in_ = uart.getInputStream();
        out_ = uart.getOutputStream();
        in2_ = uart2.getInputStream();
        out2_ = uart2.getOutputStream();
        enableUi(true);
    }
    catch (ConnectionLostException e) {
        e.printStackTrace();
        // can display some error when connection is lost, but not used at this
        // stage. Does nothing
    }

    state = ST_DONE;
    state_descript = ST_DONE_DESCR;

    //RFID
    workers_ = new
    Uart_ContrRD_RFID_Thread(in_, mRFIDProgressHandler, "worker_SCANTAG");
    workers_.start();
    //XBEE
    workers2_ = new
    Uart_ContrRD_TAGINFO_Thread(in2_, mRFIDProgressHandler, "worker_TAGINFO");
    workers2_.start();
}

/**
 * Called repetitively while the IOIO is connected.
 *
 * @throws ConnectionLostException
 *         When IOIO connection is lost.
 * @throws InterruptedException
 *         When the IOIO thread has been interrupted.
 *
 * @see ioio.lib.util.IOIOLooper#loop()
 */
@Override
public void loop() throws ConnectionLostException, InterruptedException {
    if (state == ST_REQ_TAGINFO) {

```





```

        String scanned_ID = list_id.get(count_taginfo_request-1);
char[] scanned_ID_char = scanned_ID.toCharArray();
int message_length = scanned_ID_char.length + 2;
char [] req_tag_cmd = new char[message_length+3];
        req_tag_cmd[0] = 0x02;
        req_tag_cmd[1] = 0x00;
        req_tag_cmd[2] = (char) message_length;
        req_tag_cmd[3] = 'T';
        req_tag_cmd[4] = 'R';
        System.arraycopy(scanned_ID_char, 0, req_tag_cmd, 5,
scanned_ID_char.length);
        send_string(req_tag_cmd, "UART2");
        String req_tag_cmd_str = new String(req_tag_cmd);
        Log.d(TAG, "SendNameMSG => " + req_tag_cmd_str);
state = ST_WAIT_TAGINFO;
        }else {
            Thread.sleep(100);
        }

if (state == ST_SEND_SCANTAG) {
scan_loop = scan_loop + 1;
        send_string(scantag, "UART1");    //send the command
state = ST_SCANTAG;
        }
if (state == ST_UPDATELIST) {
        runOnUiThread(new Runnable() {
@Override
public void run() {
adapter.notifyDataSetChanged();
bt_scanTag.setEnabled(true);
        }
});
        Log.d(TAG, "ST_UPDATELIST");
state = ST_DONE;
        }else if ((state != ST_DONE) && (System.currentTimeMillis()
>task_endTime)) {
if (state == ST_SCANTAG) {
        Log.d(TAG, "TimeOUT at State ST_SCANTAG, TIME = " +
System.currentTimeMillis());
list.add(new User("TAGScanTimeOut", "TAGScanTimeOut"));
state = ST_UPDATELIST;
        }
else if (state == ST_WAIT_TAGINFO) {
        Log.d(TAG, "TimeOUT at State ST_WAIT_TAGINFO, TIME = " +
System.currentTimeMillis());
list.add(new User("TAGInfoRequestTIMEOUT", "TAGInfoRequestTIMEOUT"));
state = ST_UPDATELIST;
        }
else {
        Log.d(TAG, "TimeOUT at TIME = " + System.currentTimeMillis());
list.add(new User("SCANTimeOut", "SCANTimeOut"));
state = ST_UPDATELIST;
        }
        }

if (send_auto_tune ) {
send_auto_tune = false;
        send_string(autotune_packet, "UART1");
ft = getFragmentManager().beginTransaction();

        Fragment prev_autotuneFragment =
getFragmentManager().findFragmentByTag("dialog_autotune");
if (prev_autotuneFragment != null) {
ft.remove(prev_autotuneFragment);
        }
ft.addToBackStack(null);
// Create and show the dialog.

```





```

autoTuneFragment = MyDialogAutoTune.newInstance();
autoTuneFragment.show(ft, "dialog_autotune");
    }

    if (state_descript == ST_GETTAGDETAILS) {
    char[] scanned_ID_char = details_tag_item.id.toCharArray();
    int message_length = scanned_ID_char.length + 2;
    char [] req_tag_cmd = new char[message_length+3];
        req_tag_cmd[0] = 0x02;
        req_tag_cmd[1] = 0x00;
        req_tag_cmd[2] = (char) message_length;
        req_tag_cmd[3] = 'T';
        req_tag_cmd[4] = 'D';
        System.arraycopy(scanned_ID_char, 0, req_tag_cmd, 5,
scanned_ID_char.length);
        send_string(req_tag_cmd, "UART2");
        String req_tag_cmd_str = new String(req_tag_cmd);
        Log.d(TAG, "SendNameMSG => " + req_tag_cmd_str);
    state_descript = ST_WAITTAGDETAILS;
        } else if (state_descript == ST_UPDATETAGDETAILS) {
    state_descript = ST_DONE_DESCR;
    // DialogFragment.show() will take care of adding the fragment
    // in a transaction. We also want to remove any currently
    showing
        // dialog, so make our own transaction and take care of that
    here.
    ft= getFragmentManager().beginTransaction();
        Fragment prev =
getFragmentManager().findFragmentByTag("dialogTAGDETAILS");
    if (prev != null) {
    ft.remove(prev);
        }
    ft.addToBackStack(null);

    // Create and show the dialog.
    tagdetailsFragment =
MyDialogTAGDetails.newInstance(details_tag_item.name,details_tag_item.id,tag
_description);
    tagdetailsFragment.show(ft, "dialogTAGDETAILS");
        } else if ((state_descript == ST_WAITTAGDETAILS) &&
(System.currentTimeMillis() >task_endTime)) {
        Log.d(TAG, "TimeOUT TIME = " + System.currentTimeMillis());
    //list.add(new User("ScanTIMEOUT", "ScanTIMEOUT"));
    tag_description = "Unable to retrieve tag description from server. Please
try it again...";
    state_descript = ST_UPDATETAGDETAILS;
        }

    if (autoTuneDone) {
    autoTuneDone = false;
    autoTuneFragment.dismiss();
        FragmentTransaction ft2 =
getFragmentManager().beginTransaction();
        Fragment prev_autotuneFragment =
getFragmentManager().findFragmentByTag("dialog_autotune");
    if (prev_autotuneFragment != null) {
        ft2.remove(prev_autotuneFragment);
        Log.d(TAG, "remove prev autotune dialog");
        }
    }

    // Create and show the dialog.
    autoTuneFragmentComplete = MyDialogAutoTuneComplete.newInstance();
    autoTuneFragmentComplete.show(ft2, "dialog_autotune");

    }
    }
}

```





```

/**
 * Called when the IOIO is disconnected.
 *
 * @see ioio.lib.util.IOIOLooper#disconnected()
 */
@Override
public void disconnected() {
    enableUi(false);
    toast("IOIO disconnected");
}

/**
 * Called when the IOIO is connected, but has an incompatible
 firmware version.
 *
 * @see ioio.lib.util.IOIOLooper#incompatible(IOIO)
 */
@Override
public void incompatible() {
    showVersions(ioio_, "Incompatible firmware version!");
}

//function to send string
private void send_string(final char[] string,final String choice)//final
String string)
{
byte[] bytes = new byte[string.length];

for (int i = 0; i < string.length; i++) //the char array is converted
into a byte array to send
bytes[i] = (byte) string[i];
if (choice.equals("UART1")) {
try {
out_.flush();
out_.write(bytes); //write byte array on outputstream
out_.flush();
} catch (IOException e) {
e.printStackTrace();
toast("IO failed");
}
} else{
try {
out2_.flush();
out2_.write(bytes); //write byte array on outputstream
out2_.flush();
} catch (IOException e) {
e.printStackTrace();
toast("IO failed");
}
}
}

private void enableUi(final boolean enable) {
// This is slightly trickier than expected to support a multi-IOIO use-case.
runOnUiThread(new Runnable() {
@Override
public void run() {
if (enable) {
if (numConnected_++ == 0) {
//iv_led_.setVisibility(View.VISIBLE);
iv_led_.setBackgroundColor(getResources().getColor(R.color.green));
String temp_str = "Accessory is CONNECTED";
tv_connected_.setText(temp_str);
bt_scanTag_.setEnabled(true);
ioioConnected = true;
}
} else {
if (--numConnected_ == 0) {

```





```

//iv_led_.setVisibility(View.INVISIBLE);
iv_led_.setBackgroundColor(getResources().getColor(R.color.red));
        String temp_str = "Accessory is not connected";
tv_connected_.setText(temp_str);
bt_scanTag_.setEnabled(false);
ioioConnected = false;
    }
    }
    });
}

@Override
public void onClick(View v) {
    switch(v.getId()) {
        case R.id.bt_scanTag:
            if (state == ST_DONE) {
                scan_loop = 0;
                bt_scanTag_.setEnabled(false);
                send_string(scantag, "UART1"); //send the command
                state = ST_SCANTAG;
                task_endTime = System.currentTimeMillis() + 5000;
                Log.d(TAG, "CurrentTime = " +
                    System.currentTimeMillis() + "; EndTime = " + task_endTime);
                list.clear();
                list_id.clear();
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        adapter.notifyDataSetChanged();
                    }
                });
                toast("SCANTAG START");
            } else {
                toast("SCANTAG pending to complete.");
            }
    }

    break;
}

/**
 * A method to create our IOIO thread.
 *
 * //@see ioio.lib.util.AbstractIOIOActivity#createIOIOThread()
 */
@Override
protected IOIOLooper createIOIOLooper() {
    return new Looper();
}

private void showVersions(IOIO ioio, String title) {
    toast(String.format("%s\n" +
        "IOIOLib: %s\n" +
        "Application firmware: %s\n" +
        "Bootloader firmware: %s\n" +
        "Hardware: %s",
        title,
        ioio.getImplVersion(IOIO.VersionType.IOIOLIB_VER),
        ioio.getImplVersion(IOIO.VersionType.APP_FIRMWARE_VER),
        ioio.getImplVersion(IOIO.VersionType.BOOTLOADER_VER),
        ioio.getImplVersion(IOIO.VersionType.HARDWARE_VER)));
}

private void toast(final String message) {
    final Context context = this;

```





05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi

```
        runOnUiThread(new Runnable() {
@Override
public void run() {
    Toast.makeText(context, message, Toast.LENGTH_LONG).show();
    }
});
}

private int numConnected_ = 0;
}
```



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



## B Source Code of Getting Tag ID

```

import android.os.Handler;
import android.os.Message;
import android.util.Log;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

class Uart_ContrRD_RFID_Thread extends Thread {
private Handler handler_;
private boolean run_ = true;
private final InputStream in_;
//String display_str;
String name_;

    CircularBuf buf_data = new CircularBuf(1000);

byte packet_SOF;
byte packet_complete;
int packet_length;

    Message msg = null;

static final int SET_DATA = 2;
static final int SCANCOMPLETE = 3;
static final int AUTOTUNE_COMPLETE = 4;
static final int SCANFAIL = -1;
static final byte [] SCANTAGPASS = { (byte) 0x81,0x0F};
static final byte [] TAG_INFO = {0x01,0x01,(byte) 0x82,0x00};
static final byte [] AUTOTUNE = {0x15,0x03};

//String messageRcv;
public Uart_ContrRD_RFID_Thread(InputStream in, Handler handler,String name)
{
super(name);
in_ = in;
name_ = name;
handler_ = handler;
}

@Override
public void run() {
    Log.d("UartThread", "Uart Thread [" + getName() + "] is running.");
try {
packet_complete = 1;
packet_SOF = 0;
packet_length = 12;
while (run_) {
try {
int availableBytes = in_.available();
if (availableBytes >0) {
byte[] readBuffer = new byte[availableBytes];
int result = in_.read(readBuffer, 0, availableBytes);

for (int i = 0; i < availableBytes; i++){
buf_data.add(readBuffer[i]);
}
while(Parsing());
}
} catch (IOException e) {
e.printStackTrace();
}
Uart_ContrRD_RFID_Thread.sleep(50);
}
} catch (InterruptedException e) {

```

bupsi





```

        e.printStackTrace();
    } finally {
        Log.d("UartThread", "Uart Thread [" + getName() + "] is exiting.");
    }
}

public void abort() {
    run_ = false;
    interrupt();
}

/*
 * Parses any valid messages in the circular buffer. Returns false when
 there are no more messages to parse
 */
boolean Parsing()
{
    /*
     * (packet.complete = TRUE) indicates that all previous UART messages
 have completed processing
     * and so the buffer is searched for a new SOF (start of frame) delimiter
     */
    if (packet_complete != 0)
    {
        /*
         * Find next SOF in buffer
         */
        while((packet_SOF != 0x02) && (buf_data.count >0))
        {
            packet_SOF = buf_data.get();
        }

        /*
         * Get the length of the UART message
         */
        if(buf_data.count >1) {
            byte temp_msb = buf_data.get();
            packet_length = ((int) buf_data.get()) + ((int) temp_msb)*256;
            /*
             * packet.complete = FALSE indicates that we are waiting for the
 rest of the packet to be processed
             */
            packet_complete = 0;
        }

        /*
         * If the entire message is in the buffer, check the message and parse
 it
         */
        if(buf_data.count >= packet_length )
        {
            CheckMessage(packet_length);
            packet_complete = 1;
            packet_SOF = 0;
            /*
             * Could be more messages to parse, so return TRUE
             */
            return Boolean.TRUE;
        }

        /*
         * There are not enough bytes left in the buffer to parse any more
 messages, so return FALSE
         */
        return Boolean.FALSE;
    }
    else

```

bupsi







```

{
/*
    * If the entire message is in the buffer, check the message and parse
it
    */
if(buf_data.count >= packet_length)
    {
        CheckMessage(packet_length);
packet_complete = 1;
packet_SOF = 0;
/*
        * Could be more messages to parse, so return TRUE
        */
return Boolean.TRUE;
    }

/*
    * There are not enough bytes left in the buffer to parse any more
messages, so return FALSE
    */
return Boolean.FALSE;
}

// decode Receive Message
void CheckMessage(int length){
byte [] packet_data = new byte[length-4];

byte [] packetID = new byte[2];
    packetID[0] = buf_data.get();
    packetID[1] = buf_data.get();
for (int i = 0; i < length-4; i++){
        packet_data[i] = buf_data.get();
    }

    Log.d("UartThread", "String Length = [" + length + "]);
    StringBuilder log_str = new StringBuilder();
    log_str.append(String.format("%02X", packetID[0]));
    log_str.append(String.format("%02X", packetID[1]));
for (byte b : packet_data) {
        log_str.append(String.format("%02X", b));
    }
    Log.d("UartThread", "String Received = [" + log_str + "]);
byte crc_msb = buf_data.get();
byte crc_lsb = buf_data.get();

if (packetID[0] == TAG_INFO[0] && packetID[1] == TAG_INFO[1]
&& packet_data[0] == TAG_INFO[2] && packet_data[1] == TAG_INFO[3]){

        Log.d("UartThread", "GET DATA");
int tagID_length = ((int) packet_data[2])*256+((int) packet_data[3]);
        StringBuilder sb = new StringBuilder();

byte[] temp_data_array = Arrays.copyOfRange(packet_data,4,4+tagID_length);
// end byte exclusive
for (byte b : temp_data_array) {
        sb.append(String.format("%02X", b));
    }

msg = Message.obtain(handler_, SET_DATA, sb.toString());

handler_.sendMessage(msg);
    }else if (packetID[0] == SCANTAGPASS[0] && packetID[1] ==
SCANTAGPASS[1]) {
        Log.d("UartThread", "SCANCOMPLETE");
//msg = Message.obtain(handler_, SCANCOMPLETE, null);
handler_.sendEmptyMessage(SCANCOMPLETE);

```





05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi

```
    }else if (packetID[0] == AUTOTUNE[0] && packetID[1] == AUTOTUNE[1]) {  
        Log.d("UartThread", "AUTOTUNE Complete");  
        //msg = Message.obtain(handler_, AUTOTUNE_COMPLETE, null);  
        handler_.sendMessage(AUTOTUNE_COMPLETE);  
    }else{  
        Log.d("UartThread", "SCANFAIL");  
        //msg = Message.obtain(handler_, SCANFAIL, null);  
        handler_.sendMessage(SCANFAIL);  
    }  
}  
}
```



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



## C Source Code of Tag Info Thread

```

import android.os.Handler;
import android.os.Message;
import android.util.Log;

import java.io.IOException;
import java.io.InputStream;

class Uart_ContrRD_TAGINFO_Thread extends Thread {
private Handler handler_;
private boolean run_ = true;
private final InputStream in_;
//String disp_str;
String name_;

    CircularBuf buf_data = new CircularBuf(400);

byte packet_SOF;
byte packet_complete;
int packet_length;

    Message msg = null;

static final int SET_TAGINFO = 8;
static final int SET_TAGDETAILS = 9;
static final char [] TAGINFO_CMD = {'T','R'};
static final char [] TAGDETAILS_CMD = {'T','D'};

public Uart_ContrRD_TAGINFO_Thread(InputStream in, Handler handler,String
name) {
super(name);
in_ = in;
name_ = name;
handler_ = handler;
}

@Override
public void run() {
    Log.d("UartThread", "Uart Thread [" + getName() + "] is running.");
    try {
        packet_complete = 1;
        packet_SOF = 0;
        packet_length = 12;
        while (run_) {
            try {
                int availableBytes = in_.available();
                if (availableBytes >0) {
                    byte[] readBuffer = new byte[availableBytes];
                    int result = in_.read(readBuffer, 0, availableBytes);

                    for (int i = 0; i < availableBytes; i++){
                        buf_data.add(readBuffer[i]);
                    }
                    while(Parsing());
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
                Uart_ContrRD_TAGINFO_Thread.sleep(50);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
            finally {
                Log.d("UartThread", "Uart Thread [" + getName() + "] is exiting.");
            }
        }
    }
}

```





```

    }

    public void abort() {
        run_ = false;
        interrupt();
    }

    /*
     * Parses any valid messages in the circular buffer. Returns false when
     there are no more messages to parse
     */
    boolean Parsing()
    {
        /*
         * (packet.complete = TRUE) indicates that all previous UART messages
         have completed processing
         * and so the buffer is searched for a new SOF (start of frame) delimiter
         */
        if (packet_complete != 0)
        {
            /*
             * Find next SOF in buffer
             */
            while((packet_SOF != 0x02) && (buf_data.count >0))
            {
                packet_SOF = buf_data.get();
            }

            /*
             * Get the length of the UART message
             */
            if(buf_data.count >1) {
                byte temp_msb = buf_data.get();
                packet_length = ((int) buf_data.get()) + ((int) temp_msb)*256;
            }
            /*
             * packet.complete = FALSE indicates that we are waiting for the
             rest of the packet to be processed
             */
            packet_complete = 0;
        }

        /*
         * If the entire message is in the buffer, check the message and parse
         it
         */
        if(buf_data.count >= packet_length )
        {
            CheckMessage(packet_length);
            packet_complete = 1;
            packet_SOF = 0;
        }
        /*
         * Could be more messages to parse, so return TRUE
         */
        return Boolean.TRUE;
    }

    /*
     * There are not enough bytes left in the buffer to parse any more
     messages, so return FALSE
     */
    return Boolean.FALSE;
}
else
{
    /*
     * If the entire message is in the buffer, check the message and parse
     it

```





```

        */
    if(buf_data.count >= packet_length)
    {
        CheckMessage(packet_length);
        packet_complete = 1;
        packet_SOF = 0;
        /*
            * Could be more messages to parse, so return TRUE
        */
        return Boolean.TRUE;
    }

    /*
        * There are not enough bytes left in the buffer to parse any more
        messages, so return FALSE
    */
    return Boolean.FALSE;
}

```

```

// decode Receive Message
void CheckMessage(int length){
    byte [] packet_data = new byte[length-2];

    byte [] packetID = new byte[2];
    packetID[0] = buf_data.get();
    packetID[1] = buf_data.get();
    for (int i = 0; i < length-2; i++){
        packet_data[i] = buf_data.get();
    }
}

```

```

char [] packet_data_char = (new String(packet_data).toCharArray());
String packet_data_str = new String(packet_data_char);
Log.d("TAGINFOthread", "String Length = [" + length + "]");
Log.d("TAGINFOthread", "String Received = [" + packet_data_str + "]");

```

```

if (packetID[0] == (byte) TAGINFO_CMD[0] && packetID[1] == (byte)
TAGINFO_CMD[1]){
    int delimiter_indx = 0;
    for (int i = 0; i<packet_data_char.length;i++) {
        if (packet_data_char[i] == ';') {
            delimiter_indx = i;
        }
        break;
    }
    char [] id_char = (new String(packet_data,0,delimiter_indx).toCharArray());
    char [] info_char = (new String(packet_data,delimiter_indx+1,length-3-
delimiter_indx).toCharArray());
    String id_str = new String(id_char);
    String info_str = new String(info_char);
    User data = new User(id_str,info_str);
    msg = Message.obtain(handler_, SET_TAGINFO, data);
    handler_.sendMessage(msg);
} else if (packetID[0] == (byte) TAGDETAILS_CMD[0] && packetID[1] ==
(byte) TAGDETAILS_CMD[1]) {
    int delimiter_indx = 0;
    for (int i = 0; i<packet_data_char.length;i++) {
        if (packet_data_char[i] == ';') {
            delimiter_indx = i;
        }
        break;
    }
    char [] id_char = (new String(packet_data,0,delimiter_indx).toCharArray());
    char [] details_char = (new String(packet_data,delimiter_indx+1,length-3-
delimiter_indx).toCharArray());
    String id_str = new String(id_char);
    String details_str = new String(details_char);
}

```





05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi

```
        User data = new User(id_str,details_str);  
msg = Message.obtain(handler_, SET_TAGDETAILS, data);  
handler_.sendMessage(msg);  
    }  
}
```



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



## D Source Code of Tag Info Thread

```

import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.widget.TextView;

import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

class Uart_ContrRD_Thread extends Thread {
int BUFFER_SIZE = 100;
private final TextView rcvTxtView_;
private Handler handler_;
private boolean run_ = true;
private final InputStream in_;
    String disp_str;
    String name_;

    CircularBuf buf_data = new CircularBuf(50);

byte packet_SOF;
byte packet_complete;
int packet_length;

    Message msg = null;

static final int SET_DATA = 123;

    String messageRcv;
public Uart ContrD Thread(InputStream in, TextView rcvTxtView, Handler
handler, String name) {
super(name);
in_ = in;
name_ = name;
rcvTxtView_ = rcvTxtView;
handler_ = handler;
}

void CheckMessage(int length){
byte [] packet_data = new byte[length];
for (int i = 0; i < length-2; i++){
    packet_data[i] = buf_data.get();
}

byte crc_msb = buf_data.get();
byte crc_lsb = buf_data.get();

    StringBuilder sb = new StringBuilder();

byte[] temp_data_array = Arrays.copyOf(packet_data,length-2);
for (byte b : temp_data_array) {
    sb.append(String.format("%02X ", b));
}

//char[] Temp = (new String(packet_data, 0,length-2)).toCharArray();
//messageRcv = new String(sb.toString());
//disp_str = rcvTxtView_.getText() + name_+": "+ messageRcv + "\n";
disp_str = rcvTxtView_.getText() + name_+": "+ sb.toString() + "\n";
rcvTxtView_.post(new Runnable() {
@Override
public void run() {
rcvTxtView_.setText(disp_str);
}
});
}

```





```

msg = Message.obtain(handler_, SET_DATA, sb.toString());

handler_.sendMessage(msg);
}

/*
 * Parses any valid messages in the circular buffer. Returns false when
 there are no more messages to parse
 */
boolean Parsing()
{
/*
 * (packet.complete = TRUE) indicates that all previous UART messages
 have completed processing
 * and so the buffer is searched for a new SOF (start of frame) delimiter
 */
if (packet_complete != 0)
{
/*
 * Find next SOF in buffer
 */
while((packet_SOF != 0x20) && (buf_data.count >0))
{
packet_SOF = buf_data.get();
}

/*
 * Get the length of the UART message
 */
if(buf_data.count >1) {
byte temp_msb = buf_data.get();
packet_length = ((int) buf_data.get()) + (int) temp_msb*256;
/*
 * packet.complete = FALSE indicates that we are waiting for the
 rest of the packet to be processed
 */
packet_complete = 0;
}

/*
 * If the entire message is in the buffer, check the message and parse
 it
 */
if(buf_data.count >= packet_length )
{
    CheckMessage(packet_length);
packet_complete = 1;
packet_SOF = 0;
/*
 * Could be more messages to parse, so return TRUE
 */
return Boolean.TRUE;
}

/*
 * There are not enough bytes left in the buffer to parse any more
 messages, so return FALSE
 */
return Boolean.FALSE;
}
else
{
/*
 * If the entire message is in the buffer, check the message and parse
 it
 */
}
}

```







```

if(buf_data.count >= packet_length)
{
    CheckMessage(packet_length);
    packet_complete = 1;
    packet_SOF = 0;
    /*
        * Could be more messages to parse, so return TRUE
        */
    return Boolean.TRUE;
}

/*
    * There are not enough bytes left in the buffer to parse any more
    messages, so return FALSE
    */
return Boolean.FALSE;
}

@Override
public void run() {
    Log.d("UartThread", "Uart Thread [" + getName() + "] is running.");
    try {
        packet_complete = 1;
        packet_SOF = 0;
        packet_length = 12;
        while (run_) {
            try {
                int availableBytes = in_.available();
                if (availableBytes >0) {
                    byte[] readBuffer = new byte[availableBytes];
                    int result = in_.read(readBuffer, 0, availableBytes);
                    for (int i=0; i<availableBytes; i++){
                        buf_data.add(readBuffer[i]);
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
            Uart_ContrD_Thread.sleep(50);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            Log.d("UartThread", "Uart Thread [" + getName() + "] is exiting.");
        }
    }
}

public void abort() {
    run_ = false;
    interrupt();
}
}

```





## E Source Code of Tag Description Dialog Box

```

import android.app.DialogFragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

/**
 * Created by user on 25/5/2016.
 */
public class MyDialogTAGDetails extends DialogFragment implements
View.OnClickListener {
    String name_;
    String id_;
    String description_;
    TextView tv_name_;
    TextView tv_id_;
    TextView tv_description_;
    Button bt_back_;

    /**
     * Create a new instance of MyDialogFragment, providing "num"
     * as an argument.
     */
    static MyDialogTAGDetails newInstance(String name, String ID, String
description) {
        MyDialogTAGDetails f = new MyDialogTAGDetails();
        // Supply num input as an argument
        Bundle args = new Bundle();
        args.putString("name", name);
        args.putString("ID", ID);
        args.putString("description", description);
        f.setArguments(args);

        return f;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        name_ = getArguments().getString("name");
        id_ = getArguments().getString("ID");
        description_ = getArguments().getString("description");
        int style = DialogFragment.STYLE_NORMAL;
        setStyle(style, 0);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.popup_tagdetail, container,
false);
        //View v = inflater.inflate(R.layout.popupwin,null);
        View tv = v.findViewById(R.id.title);
        String title_str = "Item Description";
        Log.d("MyDialogTAGDETAILS", title_str);
        getDialog().setTitle(title_str);

        tv_name_ = (TextView) v.findViewById(R.id.tv_name);
        tv_id_ = (TextView) v.findViewById(R.id.tv_id);
        tv_description_ = (TextView) v.findViewById(R.id.tv_description);

```





05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi

```
tv_name_.setText(name_);  
tv_id_.setText(id_);  
tv_description_.setText(description_);  
  
bt_back_ = (Button) v.findViewById(R.id.bt_back);  
bt_back_.setOnClickListener(this);  
return v;  
}  
  
@Override  
public void onClick(View v) {  
    dismiss();  
}  
}
```



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



05-4506832



pustaka.upsi.edu.my



Perpustakaan Tuanku Bainun  
Kampus Sultan Abdul Jalil Shah



PustakaTBainun



ptbupsi



## F Source Code of Database

```

set start_delimiter [binary format H* 02]
global debug
set debug 1

global database
array set database {}
set database(0) "000000000000000000000001 {SCREW Driver} {Part Number:
A0010-0008; Quanlity: 10pcs; Description: Philips Screw Driver. Medium size}"
set database(1) "000000000000000000000002 {SCREW} {Part Number: A0010-
0002; Quanlity: 1000pcs; Description: Philips Screw; Diameter: 5mm; Length:
30mm}"
set database(2) "000000000000000000000003 {WIRE 12AWG} {Part Number:
W0001-0012; Quanlity: 100pcs; Description: Wire 12AWG; Wire Length:
10meters}"
set database(3) "000000000000000000000004 {WIRE 18AWG} {Part Number:
W0001-0018; Quanlity: 100pcs; Description: Wire 18AWG; Wire Length:
10meters}"
set databse_size [array size database]
#
# PRIVATE channel read event handler for send_expect. Should
# not be called by user.
#
proc private_send_exp_reader {fh start_delimiter} {
global send_exp
    global debug
    global smatched

if {[eof $fh]} {
close $fh
set send_exp($fh.matched) -1
puts "close"
return
}

append send_exp($fh.buffer) [read $fh]
set delimiter_detect [string first $start_delimiter $send_exp($fh.buffer) 0]

if {$delimiter_detect > 0} {
set last [expr [string length $send_exp($fh.buffer)] - 1]
set $send_exp($fh.buffer) [string range $send_exp($fh.buffer)
$delimiter_detect $last]
}

# ----- debug MSG -----
if {$debug == 1} {
puts "delimiter detect -> $delimiter_detect"
puts "Buffer -><[binary encode hex $send_exp($fh.buffer)]>"

```



```
        puts "Buffer Length -> [string length $send_exp($fh.buffer)]"
    }
    # -----

    if {$delimiter_detect >=0} {
        set packet_length [binary encode hex [string index
$send_exp($fh.buffer) 2]]
        scan $packet_length %x packet_length
        set current_len [string length $send_exp($fh.buffer)]
        if {$current_len >= [expr $packet_length + 3]} {
            set send_exp($fh.matched) 1
            set send_exp($fh.command) [string range
$send_exp($fh.buffer) 3 4]
            set send_exp($fh.data) [string range $send_exp($fh.buffer) 5
[expr 4 + $packet_length]]

            set smatched 1

            if {$current_len > [expr $packet_length + 3]} {
                set send_exp($fh.buffer) [string range
$send_exp($fh.buffer) [expr 4 + $packet_length] end]
            } else {
                set send_exp($fh.buffer) {}
            }
        }
        # ----- debug MSG -----
        if {$debug == 1} {
            puts "packet length -> $packet_length"
            puts "Matched => $send_exp($fh.matched)"
            puts "command => $send_exp($fh.command)"
            puts "data => $send_exp($fh.data)"
            puts "BufferRemain -><$send_exp($fh.buffer)>"
        }
        # -----
    }

}

#
# Return the current contents of the send_expect buffer
#
# @param fh
#     channel identifier that was used with send_expect
#
# @returns string
#     the current contents of the buffer for the channel
#
proc send_exp_getbuf {fh} {
```



```

global send_exp
return $send_exp($fh.buffer)
}

#
# Reset the send_expect buffer, returning its contents
#
# @param fh
#   channel identifier that was used with send_expect
#
# @returns string
#   the current contents of the buffer for the channel
#
proc send_exp_resetbuf {fh} {
global send_exp

set buf $send_exp($fh.buffer)
set send_exp($fh.buffer) {}
return $buf
}

#
# Close out a send_expect session, closing I/O event handler
#
# @param fh
#   channel identifier that was used with send_expect
#
# @returns
#   the channel identifier passed as the fh parameter
#
proc send_exp_end {fh} {
global send_exp

fileevent $fh readable {}
foreach v [array names send_exp $fh.*] {
catch [list unset send_exp($v)]
}
return $fh
}

# assemble packet for request
proc assemble_packet {CMD_R ID DATA start_delimiter} {
set temp $CMD_R
append temp $ID ";" $DATA
set length [string length $temp]
set length_ascii [binary format H* [format %04X $length]]
set packet $start_delimiter
append packet $length_ascii $temp

```



```
        return $packet
    }

# decode message
proc decode_message {fh databse_size start_delimiter debug} {
    global send_exp
    global database
    global cmd

    set packet [assemble_packet $send_exp($fh.command) $send_exp($fh.data)
"ITEM IS NOT FOUND IN DATABASE" $start_delimiter]

    if {$send_exp($fh.command) == $cmd($fh.cmd_returnName)} {
        # ----- debug MSG -----
        if {$debug == 1} { puts "cmd return name correct" }
        # -----
        for {set k 0} {$k < $databse_size} {incr k 1} {
            if {$send_exp($fh.data) == [lindex $database($k) 0]} {
                set packet [assemble_packet $send_exp($fh.command)
$send_exp($fh.data) [lindex $database($k) 1] $start_delimiter]

                #puts -nonewline $fh $packet
            }
        }
    }
}
```

```
        # ----- debug MSG -----
        if {$debug == 1} {
            puts "name matched"
            puts "Packet Return -><$packet>"
        }
        # -----
    }
}

} elseif {$send_exp($fh.command) == $cmd($fh.cmd_returnDescription)} {
    # ----- debug MSG -----
    if {$debug == 1} { puts "cmd return description correct" }
    # -----
    for {set k 0} {$k < $databse_size} {incr k 1} {
        if {$send_exp($fh.data) == [lindex $database($k) 0]} {
            set packet [assemble_packet $send_exp($fh.command)
$send_exp($fh.data) [lindex $database($k) 2] $start_delimiter]
            #puts -nonewline $fh $packet

            # ----- debug MSG -----
            if {$debug == 1} {
                puts "name matched"
                puts "Packet Return -><$packet>"
            }
            # -----
        }
    }
}
```



```

    }
  }
  puts -nonewline $fh $packet
}

```

```

##
## MAIN
##
set fh [open "\\\\.\\COM3" RDWR]
#set fh [open COM3: RDWR]
configure $fh -blocking 0 -buffering none \
  -mode 38400,n,8,1 -translation binary -eofchar {}

```

```

global send_exp
global smatched
set smatched 0
set send_exp($fh.matched) 0
set send_exp($fh.command) 0
set send_exp($fh.data) 0

```

```

global cmd ##global variables
set cmd($fh.cmd_returnName) "TR"
set cmd($fh.cmd_returnDescription) "TD"

```

```

# set up our Read handler before outputting the string.
if {![info exists send_exp($fh.setReader)]} {
fileevent $fh readable {private_send_exp_reader $fh $start_delimiter}
set send_exp($fh.setReader) 1

```

```

    puts "fileevent set"
}

```

```

while 1 {
  vwait smatched
  if {$smatched == 1} {
    # ----- debug MSG -----
    if {$debug == 1} {
      puts "receive matched"
    }
    # -----
    decode_message $fh $databse_size $start_delimiter $debug
    set $send_exp($fh.matched) 0
    set smatched 0
  }
  # ----- debug MSG -----
  if {$debug == 1} {
    puts "done 1"
  }
}

```





}  
# -----

}

## G Measurement Results

Table 0.1 *Wireless transceiver RFID for Station Mode*

No. of Repeat	1	2	3	4	5	6	7	8	9	10	Average
NLOS (m)	40.16	41.59	40.67	40.06	43.13	38.55	38.67	40.45	41.02	41.67	40.60
LOS (m)	71.17	72.45	71.34	70.63	74.76	75.24	70.34	73.23	73.18	74.84	72.72

Table 0.2 *Wireless transceiver RFID for Mobile Mode*

No. of Repeat	1	2	3	4	5	6	7	8	9	10	Average
NLOS (m)	45.66	44.29	43.19	45.32	45.77	44.37	46.05	45.45	43.59	44.28	44.80
LOS (m)	72.34	74.56	77.21	75.49	74.83	73.56	72.65	74.81	75.61	74.37	74.54

Table 0.3 *Passive reader RFID for Mobile Mode for Antenna 1*

No. of Repeat	1	2	3	4	5	6	7	8	9	10	Average
NLOS (cm)	11.3	10.6	10.7	10.5	10.8	10.6	10.7	10.8	10.6	10.4	10.7
LOS (cm)	14.7	14.3	14.3	14.5	14.8	14.4	14.6	15.5	14.6	14.4	14.6

Table 0.4 *Passive reader RFID for Mobile Mode for Antenna 2*

No. of Repeat	1	2	3	4	5	6	7	8	9	10	Average
NLOS (m)	2.14	2.32	2.61	2.45	2.63	2.75	2.81	2.83	2.99	2.49	2.60
LOS (m)	3.33	3.82	3.17	3.29	3.45	3.76	3.92	3.23	3.27	3.77	3.50

Table 0.5 *Passive reader RFID for Station Mode for Antenna 1*

No. of Repetitions	1	2	3	4	5	6	7	8	9	10	Average
NLOS (cm)	10.2	10.5	11.2	10.6	10.8	10.5	11.2	11.6	11.3	10.3	10.8
LOS (cm)	14.3	14.9	15.0	14.1	13.6	13.6	13.7	14.5	14.4	14.7	14.3

Table 0.6 *Passive reader RFID for Station Mode Antenna 2*

No. of Repetitions	1	2	3	4	5	6	7	8	9	10	Average
NLOS (m)	2.53	2.67	2.71	2.83	2.45	2.72	2.76	2.99	2.16	2.23	2.61
LOS (m)	3.45	3.23	3.29	3.57	3.67	3.12	3.89	3.29	3.52	3.47	3.45

Table 0.7 *Tags Collection Time for Station Mode*

No. of Tags	1	2	3	4	5	6	7	8	9	10	AVG
1	127.8	144.3	148.2	131.4	139.3	128.4	149.8	138.9	139.9	119.4	136.74
5	263.4	235	223.3	240.6	237.2	239.1	150.3	154.6	222.3	213.8	217.96
10	252.9	276.7	266	252.8	268.7	292.6	256.1	268.6	235.5	243	261.29
15	290	276.1	264.8	256.2	289.3	274.8	262.1	278.4	293.6	299.5	278.48
20	369.4	323.4	312.6	308	256.9	300.1	356.4	324.7	352	312.7	321.62
25	388.3	376.21	323.76	376.26	352	366.45	351.01	372.39	380.23	388.04	367.47

Table 0.8 *Tags Collection Time for Mobile Mode*

No. of Tags	1	2	3	4	5	6	7	8	9	10	AVG
1	108	88	104	120	108	92	108	88	104	120	104.00
5	144	152	136	140	150	152	144	150	136	155	145.90
10	164	148	148	156	168	148	148	156	164	148	154.80
15	160	166	156	172	160	160	166	160	156	172	162.80
20	180	158	172	156	176	176	176	156	176	180	170.60
25	183	180	189	200	210	183	200	180	200	200	192.50



### Anti-Collision Antenna 1 for Stationary Mode

For the distance of 5 cm, the percentage of data received was calculated as follows:

$$\text{The percentage of data received} = \frac{40}{4 \times 10} \times 100\% = 100\%$$

For the distance of 10 cm, the percentage of data received was calculated as follows:

$$\text{The percentage of data received} = \frac{40}{4 \times 10} \times 100\% = 100\%$$

The distance of the anti-collision test was extended to 15 cm. At this distance, the percentage of data received was calculated as follows:

$$\text{The percentage of data received} = \frac{30}{4 \times 10} \times 100\% = 75\%$$

The above findings show that the efficiencies of the anti-collision test at the distances of 5 cm and 10 cm were 100%, and such efficiency dropped to 75% when the distance was extended to 15 cm.





## Anti-Collision Antenna 2 for Stationary Mode

For the distance of 1 m, the percentage of data received was calculated as follows:

$$\text{The percentage of data received} = \frac{40}{4 \times 10} \times 100\% = 100\%$$

For the distance of 2 m, the percentage of data received was calculated as follows:

$$\text{Percentage of data received at 2 m distance} = \frac{38}{4 \times 10} \times 100\% = 95\%$$

The distance of the anti-collision test was extended to 3 m. At this distance, the percentage of data received was calculated as follows:

$$\text{Percentage of data received at 3 m distance} = \frac{32}{4 \times 10} \times 100\% = 80\%$$





## Anti-Collision Antenna 1 for Mobile Mode

For the distance of 5 m, the percentage of data received was calculated as follows:

$$\text{Percentage of data received} = \frac{40}{4 \times 10} \times 100\% = 100\%$$

For the distance of 10 m, the percentage of data received was calculated as follows:

$$\text{Percentage of data received} = \frac{40}{4 \times 10} \times 100\% = 100\%$$



The distance of the anti-collision test was extended to 15 cm. At this distance, the percentage of data received was calculated as follows:

$$\text{Percentage of data received} = \frac{31}{4 \times 10} \times 100\% = 77.5\%$$

The above findings show that the efficiencies of the anti-collision test at the distances of 5 cm and 10 cm were 100%, and such efficiency dropped to 77.5% when the distance was extended to 15 cm.

Therefore anti-collision efficiency is 100% at 5 cm and 10 cm distance while at 15 cm the efficiency is 77.5%.





## Anti-Collision Antenna 2 for Mobile Mode

For the distance of 1 m, the percentage of data received was calculated as follows:

$$\text{Percentage of data received} = \frac{40}{4 \times 10} \times 100\% = 100\%$$

For the distance of 2 m, the percentage of data received was calculated as follows:

$$\text{Percentage of data received at 2 m distance} = \frac{39}{4 \times 10} \times 100\% = 97.5\%$$



The distance of the anti-collision test was extended to 3 m. At this distance, the percentage of data received was calculated as follows:

$$\text{Percentage of data received} = \frac{35}{4 \times 10} \times 100\% = 87.5\%$$

The above findings show that the efficiencies of the anti-collision test at the distances of 1 m, 2 m, and 3 m were 100%, 97.5%, and 87.5%, respectively.

