Towards a Cross Platform Solution to Share Small-Scale Data Across Smart Devices using Visible Light

A Prototype Developed in Android Environment

A Thesis
Submitted to the Department of Electrical and Electronic Engineering,

BRAC UNIVERSITY

By

FAIZAH FAIROOZ (13110019)

PROTHOMA CHOWDHURY (13221056)

REZWANA AHMED (13110004)

Supervised by
RACHAEN MAHFUZ HUQ
Lecturer
Department of Electrical and Electronic Engineering
BRAC UNIVERSITY, Dhaka

FALL 2016
BRAC UNIVERSITY, Dhaka

DECLARATION

We, hereby declare that the entire thesis work on *"Towards a Cross Platform Solution to Share Small-Scale Data Across Smart Devices using Visible Light -A Prototype Developed in Android Environment"* was carried out by us as part of our Bachelor of Science in Electrical and Electronic Engineering Program. The content of the thesis paper was not published anywhere beforehand. This thesis paper is submitted to the department under the supervision of Mr. Rachaen Mahfuz Huq, Lecturer, EEE, BRAC University, in accordance with the rules and regulations of BRAC UNIVERSITY.

……

Rachaen Mahfuz Huq

Thesis Supervisor

Thesis Members:

….……

- Faizah Fairooz

…….

- Prothoma Chowdhury

..……..

- Rezwana Ahmed

ACKNOWLEDGEMENT

We would like to thank our supervisor for his constant guidance throughout the project, for his patience and for imparting his valuable knowledge onto us. His guidance has helped us immensely in the completion of this thesis. Without his efforts, this thesis would not have taken the shape that it has taken. It was an honor for us to work under his supervision and we are grateful that we could achieve copious amounts of knowledge from him in the process.

We would also like to thank Mr. Mohammad Istiak Hossain, Doctoral Researcher, KTH Royal Institute of Technology, Sweden, for his feedback on improving the prototype of the project.

Table of Contents

List of Figures

ABSTRACT

There is a significant growth of cloud based applications in the Smartphone app-markets which leads to the necessity of finding a way where two Smartphone can easily exchange short streams of data. These small scale data usually act as a “key” to the original content that the users wish to share. Possible use cases could be – identity information exchange, file sharing links, etc. The main challenge with the existing communication protocols e.g. Bluetooth, NFC is that they do not work cross-platform, for example Android-to-IOS or vice versa. This project explores the possibility of using the integrated Smartphone sensors (e.g. ambient light sensor) for these kinds of short scale data exchange. The successful completion of this project would have a deliverable of a Smartphone app which can share the “links” to particular contents stored in the application server, using the Smartphone sensors.

1. INTRODUCTION

1.1 Background and Motivation

Machine-to-machine (M2M) communications emerge to autonomously operate to link interactions between Internet cyber world and physical systems. It is an inevitable part of the Internet Of Things which is crucial for developing connection across all aspects of the physical world around us. With modern machines increasingly containing computers as well as other electronic devices along with other billions of connected devices, it is evident that these machines need to have better connectivity, sharing capacity and interoperability with all physical devices and objects around them. Even though M2M is about communication among machines which involves no or limited human intervention, there is still a biggest challenge of interoperability. Interoperability is widely agreed to be the biggest challenge in M2M/IoT, this is the reason for which IoT/M2M still could not take off, even though significant infrastructural progress were made by tech-giants like: Google, Amazon, Samsung, IBM.

Moreover, when it comes to instant and wireless file sharing across devices, smart phones can be taken as a good example why cross-platform data sharing is necessary. With growing number of applications in both of the dominant smart phone OS markets e.g. Android and iOS, short range communication protocols still exist only inside an operating system, for example Bluetooth and NFC. For example Android users have an extensive pool of tools to choose from when it comes to sharing data. From relatively new data sharing techniques like: WI-FI Direct and NFC Beams to various file sharing between apps and services out there, we have a number of means to share files from one device with another. However, we are yet to come across a cross-platform tool that allows quick and convenient file sharing across different platforms and Operating Systems. There exists not a single application which exists in both the platform and enables us to share data between an Android Phone and an iOS phone without the intervention of the internet. The main reason behind this is that both these smart phone market leaders have their own proprietary means to serve this purpose and they made their products incompatible with the others. Having this sort of short range communications exclusively being dependent on customized transmission and receiving antenna, the hardware ecosystem is also finding itself difficult to integrate among

all devices. This is the reason why we had shifted our thoughts to solve this problem, not looking forward in the future, rather to look back in the past and use an old technique to solve this new problem.

One of the very useful ways through which data is thought of being sent across cross-platform is through visible light communications, VLC. VLC dates back to the 1880s in Washington DC when the Scottish born scientist Alexander Graham Bell invented the prototype whose work was to transmit speech on modulated sunlight.[1] However, more recent work has been done in the year 2003 at Nakagawa Laboratory in Keio University, Japan where LEDs were used to transmit data by visible light.[1] The data transmission speeds of VLC systems are seen to be rapidly improving, with a frequency-modulated white LED being shown by Siemens researchers and the Heinrich Hertz Institute in Berlin to be capable of transmitting information over 5 meters at a rate of 500 Mbps, significantly faster than present Wi-Fi technologies (that can operate at rates of up to 150 Mbps) [2]

This motivated us to come across such a way through which we will enable short stream data communication for M2M communication. Here to send or receive data we use Android-iOS smart phones where we do not need to build any separate antenna to receive/transmit data as we need for NFC/Bluetooth/WifiDirect. We demonstrate this using a smart-phone app which will be able to send receive data using its integrated flashlight (source) and ambient light sensor (receiver sensor). This triviality of light makes it Cross Platform. It is believed that the project would motivate other individuals to come up with the further improvement regarding this issue

1.2 Problem Area

1.2.1 Major Definitions

The project has certain applications areas for one to look into. Global information sharing has become very simple due to the advent of whole distributed networks, intranets, cross-platform and standardization of IP protocols. Few applications areas have been mentioned:

(a) **Internet of things IoT**: The term IoT focuses attention on the connection between things-that is sensors and devices exchanging data. To put it simply, it is basically connecting any

device to the Internet (and/or) to each other. This includes everything from cell phones, coffee-makers, headphones, lamps, etc.[3] IoT allows for virtually endless opportunities and connections to take place and for this massive amount of data needs to be produced and shared. The IoT is not just about connecting things, it is also about the connections that it creates between an organization and its customers, suppliers and competitors. Creating business value from the IoT is strongly associated with sharing data with other organizations.

Most wireless communication, including Bluetooth and Wi-Fi, occurs in the radio frequency (RF) spectrum. Unfortunately transmission in the RF spectrum is limited due to regulation and interference. Since more and more devices are having a need for wireless communication due to the rise of the internet of things, RF spectrum is becoming crowded so VLC can be a way to solve the problem. VLC is appealing because it is in an unregulated bandwidth, unlike RF, which makes adoption easier.VLC also has a potential 300 THz of bandwidth, allowing for multi-gigabit per second communications. [4] Thus, this would result in less RF interference and more dedicated communications.

M2M Visible light communications provide a lot of solutions in enhancing industrial instrumentation, coordinating the movements and timing of robots in manufacturing settings as well as enable Point of Sale (POS) communications between devices without using wires or RF spectrum. [5]

Thus, VLC wireless networks that are interconnected can improve production, safety, reliability, and efficiency in a multitude of settings.

(b) **Smart Cities**: On a broader scale, the IoT can be applied to things like transportation networks, “smart cities” which can help to reduce waste and improve efficiency for things such as energy use, thus helping us understand and improve how we work and live. Sometimes there is a technology platform that makes it easier to share data in a responsible way. Thus IoT has an immense influence in developing cities by improving infrastructure, creating more efficient and cost effective municipal services enhancing public transportation, reducing congestion and keeping citizens safe and more engaged in the community. [3]

The development of visible light communication based on the use of LED technology in an area has allowed the existence of lighting devices with better performance in terms of both energy efficiency and light expectancy. It has high available bandwidth, power efficiency, no need of

high deployment costs and most importantly the lighting functionality, which eliminates the need for new infrastructure deployment. [6] Thus VLC is mostly likely to become one of the leading factors for the forthcoming digital ecosystem in the smart area.

Data sharing thus is a very useful way through which we are getting effective communication, having improved overall accuracy of public data and an easy access to information as well as an entertainment that we want to experience.

1.3 Previous Works

There have been many researches related to possibility of potential m2m communications with the help of the built-in flashlight of a Smartphone. Hesselmann et al. proposes a method of building a bidirectional communication channel between smart phones and interactive tabletops with the help of the built-in flashlight and camera of smart phones, and the camera and screen of the tabletops that is both secure and fast. A Google Nexus One phone and an interactive tabletop were used to establish the channel. To begin the communication, the phone is positioned on the tabletop directly and the entire area below the camera of the phone is illuminated by the tabletop with constant colors in order for the transmission of data to the Smartphone from the tabletop using color based encoding. For the transfer of data from the mobile phone to the tabletop a Non-return-to-zero (NRZ) line code is used where the high and low bits were characterized by switching the flash on or off. The tabletop senses the change in light levels as the flashlight is turned on or off and decodes the transmitted message accordingly [9].

In the paper by Park et al, visible light communications was established between mobile devices using an Android smart phone and an Android Trailing kit. The data was transmitted from the Smartphone with the help of the built-in flashlight. Binary data was input into the transmitter application which was saved in an array. On-Off-Keying (OOK) modulation scheme was used to transmit binary data ‘1’ or ‘0’ by turning the flashlight on or off respectively. Trailing kit with a Samsung S5PV210 Processor is used by the receiver. The developed Android-NDK for the receiver captured the transmitted data with the help of the camera by detecting the color changes of the transmitted signal. The detected color change values are compared with a certain value and

set as ‘1’ if it is larger than that, and if not, it is recognized as ‘0’. The decoded data is then displayed on the Android application in the receiver[10].

In a research by Shirazi et al, interactions with large displays are created by means of the mobile phone’s flashlight. Here the built-in flashlight of the smart phone is not modulated and used as is. The user holds the phone with the flashlight turned on and facing the large display, and a webcam installed below the large display can capture the motion of the phone by tracking the light. The phone can then be used to interact with the large display by moving the phone which would move a mouse pointer in the large screen in turn. A Java application is used to capture the webcam’s video feed, which is then processed to calculate the radius and center of the flashlight for each frame. The calculated light position from the video feed is then mapped onto the large screen. This setup allows the phone to be used in making selections by quickly turning the flashlight off and on, moving the mouse pointer, scrolling, and zooming in or out by moving the phone closer or further away from the screen [11].

1.4 Research Gap and Contribution of the Thesis

While a considerable amount of studies have been conducted on visible light communication between smart phones and other various devices, for example, by using the smart phone’s built-in flashlight and the camera of the devices (Hesselmann, Henze, & Boll, 2010; Shirazi, Winkler, & Schmidt, 2009) [9,11], or by using the camera of the smart phone and LED light sources (Danakis, Afgani, Povey, Underwood, & Haas, 2012; Corbellini, Aksit, Schmid, Mangold, & Gross, 2014)[13,14],we have opted to use the flashlight and the ambient light sensor of the smart phone in our thesis to come up with a method to communicate between two smart phones without the involvement of the camera features reducing the total duration of human interactions. The receiver uses the On Off Keying detection method taking into account the light level of the flashlight shone on its light sensor and decodes the received message accordingly.

Additionally, since the present mobile market consists of a number of different platforms, a cross platform solution is highly desirable in order to connect and communicate with more devices, and thus potentially widen the user base. Current radio-based technologies such as Bluetooth and NFC used in communications do not work cross platform, for example in case of Android-to-iOS

or vice versa due to incompatibility. For them to work cross platform, we would need to build separate antennas in order to transmit and receive data across different platforms. The method developed in our thesis uses the integrated ambient light sensor and the built-in flashlight, both of which are present in all smart phones of recent times, and no additional hardware. This inconsequential nature of light enables us to implement the proposed technology on any smart phone, ensuring that our application works cross platform.

We also decided to use Morse code as the coding scheme when encoding data prior to transmission. Morse code is an example of variable-length encoding which sends information by transmitting a sequence of long or short pulses of light or sound. Each letter or character is assigned a unique code, the more common the letter or character, the shorter the code [15]. Since we are using light as our medium for transmission, Morse code is a reasonable choice. The use of Morse code allows us to encode a variety of letters and words which could be combined to form a vast assortment of unique messages for transmission. Moreover, the coding scheme is relatively easy to employ, and the internationally approved predefined guidelines of Morse code would mean our encoded data would be widely accepted, and so, help in attaining a more widespread implementation.

Furthermore, our thesis prioritizes accuracy during the transmission and reception of data over speed. We have carefully selected the conditions within the application developed for this thesis in order to ensure that there are no errors during the transfer of data. The transmitting light pulse width, ambient light reference level in receiver, range of pulse width used in detection in receiver were all tested and modified to increase accuracy in transmission and reception.

Since the application uses visible light communication and works across platforms, it can easily be modified and used in VLC-based indoor navigation, which use light and sensory data collected from smart phones as a means to locate people and objects within a building where other navigation techniques such as GPS fall short due to attenuation of the signal because of the building construction materials. It can also be used as a communication medium for “ubiquitous computing”, a concept that uses underlying technologies such as internet, sensors, location and positioning, and mobile protocols to make computing available anytime and anyplace[12].

1.5 Research Question

"How to develop a cross platform solution for accurate transmission and reception of short scale stream of data between smart-phones using the integrated ambient light sensors and flash lights?"

1.6 Methods Adopted

For the completion of the project, certain methods have been adopted for a successful transmission and reception data. For the transmitter side, we have used Morse code technique. Morse code is used since it's an effective way of transmitting text information as a sequence of on-off-tones, lights or clicks which can be easily understood by any listener or observer. In Morse code there is a standardized sequence of short and long signals called "dots" and "dashes". In our project, the light pulse is acting as the convention unit. As we are entering any letter or word, it is being in turn converted into its corresponding Morse code signals of either dot or dash and is transmitted as light pulse.

According to International Morse Code, the length of a dot is one unit and that of a dash is three units.Thus in the transmitting side the length of the pulse width is considered to be ON for 200ms and this is considered to be as the "dot". Hence 200ms is considered as one unit. Thus when light is being ON for 600ms it will be considered as dash since a dash is 3 units long. Furthermore, the gap between each letter is 200 ms because it is found that the space between each dot or dash is equal to the dot duration. So, the dot duration is considered as the basic unit for the measurement of time in code transmission.

For the receiving end, we use the nN-OFF-KEYING(OOK) technique for an efficient reception of data. In the receiving end, in order to avoid the effect of backlight,a reference backlight level which is about 500 lux is set. If the light level is 500 lux or above the light will be considered ON and below it OFF. As soon as the backlight level is 500 lux or above , a timer which is dependant on the light pulse gets turned ON and by measuring the time from the timer a "dot" or "dash" is being recognized If the timer is ON for '200 ms' it corresponds for a "dot" and

similarly for ‘600ms’ it refers for a “dash”. As soon as the light level goes below 500 lux the light dependant timer switches off.

As soon as the light dependant timer gets switched off, another timer is switched ON which measures the Gap Time. This Gap Time tells the receiver whether a ‘letter’ or a ‘word’ is arriving. If the Gap Time is for ‘200 ms’, it signifies that a letter has been transmitted and for ‘600 ms’ it considers a letter to have already passed. If the timer reads for more than ‘600 ms’, it is then listening for another letter. At a time when we see that a gap time is about ‘1400 ms’, it is understood that a whole word is being sent and it is waiting for the next letter to come.

Thus in this manner, seeing the duration of time, we calculate the ‘dots’ and ‘dashes’ and hence accordingly understand whether a ‘letter’ or a ‘word’ has been transmitted and also need to make sure that whether it has been received successfully as per the transmission.

1.7 Outcome of the Thesis

For our thesis, a transceiver application was prepared which is capable of transmitting and receiving small scale data messages using visible light communications. In the application, the transmitter takes input from the user and uses the inbuilt flashlight of the Android device to send out a sequence of light pulses of variable length that corresponds to the data being transmitted. The application uses the Morse code coding scheme to encode the data before transmission. The dots and dashes are signified by the light being ON with the length of the light pulse for a dash being thrice as that for a dot. The gaps between the dots and dashes are represented with the light staying OFF. Although the amount of time the light stays OFF depends on whether the gap is within a letter, between two letters, or between words. The receiver part of the thesis uses the light sensor of the device to detect the transmitted light pulses and decodes them to acquire the original message that was transmitted.

1.8 Thesis Overview

We mostly divide our whole thesis work in five basic chapters where each chapter has subsections. Chapter 1 contains the introduction where we mention about the background and motivation of the project, the certain scopes and application areas, previous works, methods

adopted and outcome of the thesis. Chapter 2 mainly deals with the theoretical background where we have vividly provided the major definitions like: android environmental sensor, encoding, ambient light sensor, etc. We have also shown a complete scenario of the established coding schemes like (Morse code, OOK) and the reasons behind choosing these so that e reader gets a clear idea regarding the working of our project. Furthermore, in the subsections of chapter 2: we describe the background on Android sensor programming and Android environment and also put some applications components, activity, the life cycle of an activity and various other important phenomena that helped in the completion of the project. The chapter 3 comprises of the design method where we have discussed in fine points the complete coding techniques. Here, we made use of some graphical representations like block diagrams, charts, etc to show clearly "what technologies" were used and in "what parts" of the project. Additionally, it contains certain particular design specifications and assumptions. The chapter 4 is dedicated to the software that is developed where we talk about the performance of the software and about some tests which we performed to see whether the app is accurately receiving the data that is being transmitted; thus mainly focused on the end product with a no of screenshots and real-pictures. The chapter 5 is the concluding section where we discuss about future works, provide the summary of findings, and talk about few of the drawbacks of our project and the development frontiers that we are leaving open for further development.

2. THEORITICAL BACKGROUND

2.1 Introduction to Sensors and Encoding

Most Android devices have inbuilt sensors that are adept at providing accurate and precise raw data. These sensors are used by the devices to record ambient room temperature, ambient geomagnetic field, ambient humidity, the orientation of a device and much more. The raw data collected by the sensors are used to implement certain features on the devices. For example, the display screen brightness level on your phone may need to be adjusted to suit your needs when you are in a dark room by accessing ambient light information from the light sensor. Moreover, some sensors track the physical position of a device to provide GPS with the exact location of the device which helps in device tracking and also enhances navigation capabilities of the device. Android sensors are assorted into three major categories:

- **Motion sensors**

  It measures acceleration forces and rotational forces along three axes to detect physical movement on a device or within an environment. It encompasses accelerometers, gravity sensors, gyroscopes, and rotational vector sensors and uses each of them individually or in combination to obtain desired information.

- **Environmental sensors**

  They measure environmental parameters, detect changes in its environment, and then provide a corresponding output. It encompasses thermometer, barometer, and photometer.

- **Position sensors**

  This is used for position measurement along linear, angular, or multi-axis. It encompasses orientation sensors and magnetometers.

2.1.1 Environmental Sensors

Environmental sensors comprise of four sensors, namely ambient humidity sensor, ambient temperature sensor, ambient light sensor and ambient pressure sensor, that work independently or together to monitor the different environmental properties and parameters. These sensors are

hardware-based, hence they are only accessible if the hardware requirements pre-exist on the device. The light sensor is present in most android devices since it is used to auto-adjust screen brightness. However, it is good practice to ensure that an environment sensor is available on the device before any attempts to access data from them. The environmental sensor is the most relevant to our project, specifically the ambient light sensor since we have used the ambient light sensor to retrieve data from incoming light signals.

2.1.2 Ambient Light Sensor

An ambient light sensor is present in most mobile phones and it automatically measures the amount of light in the environment. It measures the light intensity in **LUX (lx),** the SI unit of luminosity. Ambient light sensor is an extremely common environmental sensor that is prominently used in devices to adjust the screen brightness with respect to the surrounding light conditions. For example in dim light, with the help of ambient light sensors, the device screen gets dimmer and in bright environments, the screen gets brighter. Ambient light sensor's brightness adjustment feature is used to extend a phone’s battery life and it also contributes to making the viewing experience more comfortable for the user.

2.1.3 Encoding

Computer systems store all types of data in the form of binary digits. They convert the data to a binary form before it is stored in the device. The process of converting data into the binary form is known encoding. Different coding schemes are used to convert data into its binary form. The specific coding scheme used in this project for the purpose of transmission and reception of data is Morse code.

2.2 Coding Scheme

2.2.1 Overview

In character encoding, a collection of characters is ciphered using a coding scheme. Character encoding is used in computation, data storage, and transmission of textual data. It converts the data or message into a coded language that can be easily transmitted or stored. The predominant spoken and written language can be complex for devices to transmit and occupy copious amounts of storage space. Conversion of such languages to a simpler form enables the devices to send data, store data and retrieve them with ease. Morse code, the Baudot code, the American Standard Code for Information Interchange (ASCII) and Unicode are a few easily recognizable coding schemes used in character encoding systems.

2.2.2 Morse code

Morse code is a type of encoding scheme where letters are represented by a mixture of long and short signals of light or sound. These sequences of short and long signals are called "dots" and "dashes" and sometimes they are also referred to as "dits" and "dahs" This method of encoding data or messages is named after Samuel F. B. Morse, an inventor of the Telegraph. Primitive versions of the Telegraph sent pulses of electric current along wires. These wires controlled an electromagnet positioned at the receiving end of the Telegraph system. These Telegraph Systems prompted the search for a new encoding technique that would transmit written or spoken language combining only these pulses, and the gaps between them. Hence, Telegraph Systems were the first to implement the Morse code as a coding scheme and paved the way for the modern International Morse Code that we use today.

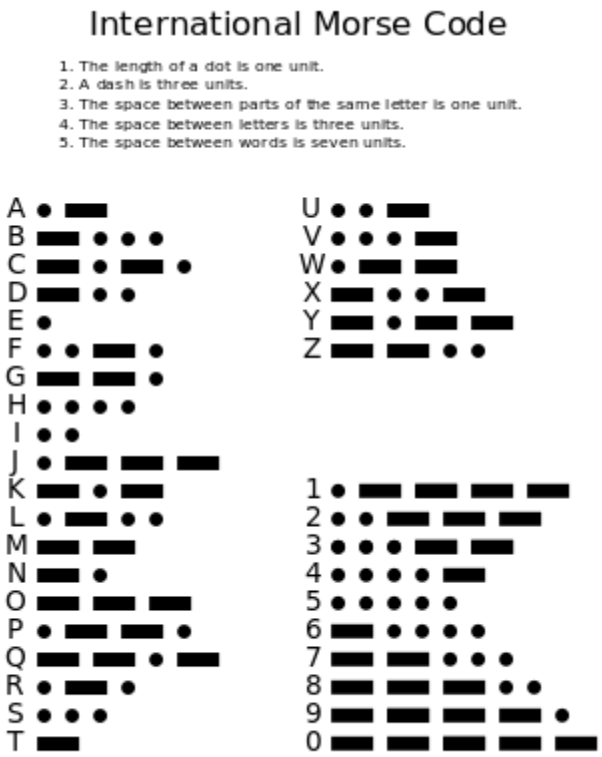

Fig: 2.2.2.1: International Morse Code, Source: [7]

2.2.3 Choice of Coding Scheme

Morse code is a universal method of transmitting data or message through combinations of short and long sound or light signals. It incorporates these sequences of "dots" and "dashes" (or "dits" and "dahs") to encode pre-existing language into a form that is easily transmissible. Since the medium of data transmission is light, Morse code is a feasible choice for the coding scheme. Using Morse code gives us the opportunity to encode a variety of letters and different lengths of words. This provides us with an arsenal of possible combinations of letters and characters to form unique messages that can be transmitted. The pre-existing set of code is internationally acknowledged, hence, the data would not be difficult to encode and decode. The application in question transmits encoded data in the form of light signals and is received on the opposite end by another device. This form of data transmission is somewhat similar to the Telegraph System,

ergo, making Morse code an appropriate choice for the coding scheme. This coding scheme is fairly easy to apply and already has certain predefined guidelines that would make our encoded data universally acceptable, which would increase the scope of implementation of our application.

2.2.4 Morse code Guidelines

All forms of Morse code whether in written form, using light signals or sound signals, follow a set of guidelines that make it universal. Each and every unique, Morse code dot/dash sequence symbols represents a text character (number or letter). The formation of these sequences of dot/dash is based on the following rules that apply to all Morse coded data:

- The length of a dot is one unit.
- The length of a dash is three units.
- The space between parts of the same letter is one unit.
- The space between two letters is three units.
- The space between two words is seven units.

Each unit depends on the pulse width. For example, if each light signal pulse width is 200ms, then one unit is 200ms. As a result, for a dot to be transmitted the light has to be ON for 200ms and for a dash, the light has to be ON for 600ms. Similarly, the gap within the letters requires the light to be OFF for 200ms. Furthermore, the gap that distinguishes between the space between separate letters and words requires the light to be OFF for 600 ms and 1400 ms respectively.

In order to understand the implementation of Morse code to encode and decode data in the application, we must first discuss the Android program environment and the Android sensor programming basics.

2.3 Android Development Environment

2.3.1 Overview

Java programming language is predominantly used to program android apps. The Android SDK manager compiles the code written by the app developer with all the relevant data and resource files into an APK (i.e. application package). All the pertinent files that are needed to install an Android app are found within the APK file. Android-powered devices use this specific APK file to install the application on the devices. Android has an application framework that assists in creating new and interactive apps. It is essential to understand the underlying concepts related to the Android app framework to make sense of the inner workings of an application.

2.3.2 Android Fundamentals

Android is a world renowned operating system. Google developed this mobile operating system, based on the Linux kernel. It was created for touch screen portable devices, for e.g. Smartphone and tablets. The Android OS is a multi-user Linux system. Separate security sandboxes are available for each android application, where they are provided protection in numerous ways:

- Each app acts like an individual user.
- The system provides each app with a distinct Linux user ID. This ID is used by the app to access the files on the system. The ID is used so that the files can only be accessed by the app that matches the given ID exclusively.
- They use different Virtual Machines for different processes. This ensures that each app code runs separately from other apps.
- Android starts the process when the app is running and shuts down the process when it's no longer needed.

The "principle of least privilege" is used by the Android operating system. This means that the app will just access those components that are needed to do the job at hand and cannot retrieve information from other files that it does not have permission to access.

2.3.3 Application Components

Application components are the foundational blocks that build up an Android application. These components are sometimes dependent on one another. There are primarily four types of app components:

- Activities.

  An activity represents a single screen on the application. It houses the visible UI through which the user interacts with the app.

- Services.

  Services are concerned with the management of the background processes of an application. They are needed to keep applications running in the background such as the music app playing songs in the background while the user is surfing the web using some other application.

- Content providers.

  Content providers handle shared sets of data that are stored in a variety of storage locations where the apps can access them from. Other applications can retrieve or alter these data sets if they are given permission to do so by the content provider. In short, the content provider manages data and data management issues.

- Broadcast receivers.

  Broadcast receivers are used to communicate between the application and the operating system or other corresponding applications. They respond to any incoming query or message accordingly. For example, if the keys in a game are restored after 3 hours, the app will send

message after 3 hours to the device's system. The operating system then notifies the user about the restoration of these keys in the form of a notification.

Each of these components is required because they serve different purposes. They have individual life cycles that dictate how each component is created and destroyed during the execution of the application. The most important component of an application is the activity because it creates a channel between the user and the application. It is through this activity that users interact with the application.

2.3.4 Activity

An activity represents a single screen on the application. It houses the visible UI through which the user interacts with the app. Each activity displays a new screen or window. There can be more than one activity in an apk file. In which case, the default launcher must be specified. This is the window that is viewed when the app is initially launched.

Whenever, a new activity is started, it opens a new window. The previous activity is stopped; however, the application retains this activity in the back stack. The back stack follows the "last in first out" stack mechanism. Therefore, when a user wishes to end an activity by pressing the back button on the device, the current activity is popped from the stack and the activity that was in use prior to it resumes.

The activity's lifecycle callback methods are used to manage this change in state of each activity. There are a variety of callback methods that the application uses during these changes. They can be programmed to execute different tasks based on the change of state of the activities. For example, when an activity is stopped, it can be programmed to stop the sensor from reading data by unregistering the sensors. This is done within the OnStop() callback method. These callback methods dictate the life cycle of an activity. It is important to understand this concept to get a clear picture of how an application works.

2.3.5: The Life Cycle of an Activity

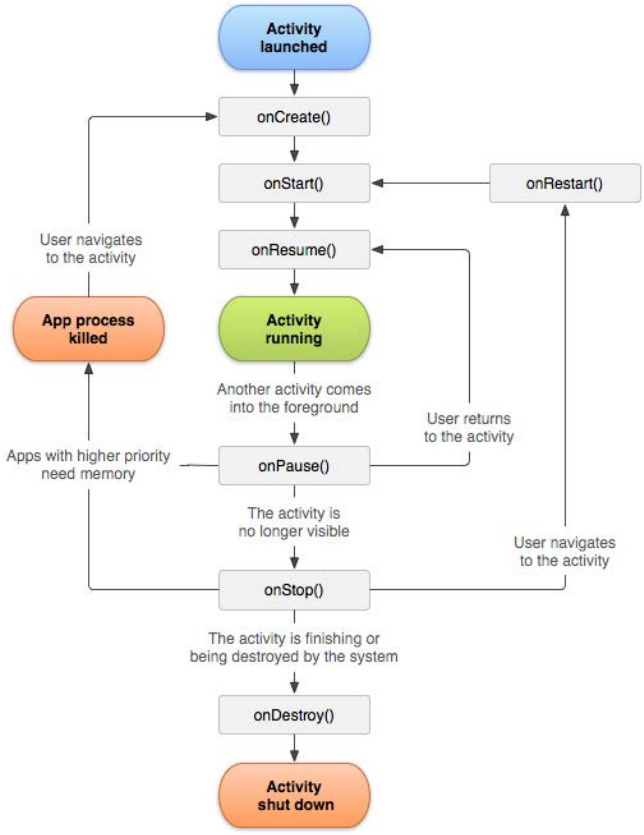

Fig: 2.3.5: The Life Cycle of an Activity, Source: [8]

An Activity subclass needs to be created to set up a new activity. In this Activity subclass, callback methods will have to be implemented. These methods are called when the activity changes between different states of its life cycle and they bring about certain changes that have been programmed within these methods. All these lifecycle callback methods are used in unison to create a perfectly functioning application that is equipped to tackle any interruptions or discrepancies during the execution of the application to provide an uninterrupted user experience.

Any activity begins with an **onCreate()** callback method. Within this method, all the fundamental parameters are initialized. This ensures that all the essential components are present

when the application is launched. Most notably, this method encapsulates the setContentView() command which delineates the activity's user interface layout.

The **onStart()** method follows the **onCreate()** method. It is also called after the **onRestart()** method and precedes the **onResume()** method. The **onResume()** method enables the user to communicate with the activity. This method is aptly named "on Resume" since it is called when the application is paused and then restarted. This resumes the user interaction with the activity. Animations and exclusive access devices are usually set up in this method. Many parameters need to be re-initialized in this method. When the activity is resumed, the application is on the user's screen ready for any user interaction.

The lifecycle of an activity is dependent on its interaction with other activities, its task and also the back stack. An activity can exist in any of the three states:

- Resumed

  When the activity is visible on the user's screen for user interaction (i.e. when the app is running)

- Paused

  Takes place when another activity is present in front of it but it is still partially visible on the user's screen behind the current activity. Even though it is visible, it is not available for user interaction at that moment. The activity is still alive and it is kept in the device's memory storage, where it retains the state, data, and information of the activity

- Stopped

  The activity is no longer visible. This means that the activity is in the background and not available for user interaction. The activity is still alive and it is kept in the device's memory storage, where it retains the state, data, and information of the activity.

In both cases, i.e. when an activity is paused or stopped, it is kept alive; however, it can be aborted when the memory runs low on the device. The system can accomplish this by calling the **finish()** method, or simply by killing its process. If the activity is reopened after the device drops the activity, it needs to be created again.

The **onPause()** method is called as soon as the user decides to leave an activity such that another activity takes the user focus. The activity does not get destroyed, however, this method is where the developer can dictate a few commands that the application can act upon in an event that the activity remains in the paused state for an indefinite time, for instance, if the user does not intend to resume the activity session.

The **onStop()** method follows the **onPause()** method, where the activity is obscured from the user's view. The visibility is eclipsed either because the activity is on the verge of being destroyed or a new activity is opened in front of it. It returns to **onRestart()** if the activity becomes visible again allowing it to interact with the user. Otherwise, it is followed by **onDestroy()** where the activity is being killed.

The **onDestroy()** method is called when the activity is being destroyed. An activity can be destroyed when the activity is finishing because the **finish ()** method is called. In addition, the activity can also be temporarily destroyed when the device has low memory space in order to save space. There are three key loops worth mentioning within the lifecycle of an activity:

- The moment from when the **onCreate()** method is called by an activity to the point when the activity is destroyed using the **onDestroy()** method is referred to as the entire lifetime of an activity.
- The moment from when the **onStart()** is called until a corresponding **onStop()** method is called, is known as the visible lifetime. The user might be able to view the activity; however, it cannot be interacted with since it is not in the foreground.
- The moment from when **onResume()** is called until a corresponding **onPause()** method is called, is known as the foreground lifetime.

Before an activity can perform its designated tasks, the app components and permissions must be declared in the manifest file.

2.3.6 The Manifest File

All the app components must be listed within the manifest file. This informs the Android system about the app components that are present in the application. All the components used by the application must be declared in this file. In addition to listing all the components in use, the manifest file also contains user permissions that request a permission that might be required by the app to operate correctly, such as permission to access camera flashlight, permission to access the gallery etc. Apps cannot function properly below a specified API level. The manifest file states the minimum API Level required by the app to function properly. If the API level is incompatible, the application will not run on the device. The manifest file also mentions the hardware and software features utilized by the app e.g. the speakers, Wi-Fi etc. Moreover, the API libraries that the app needs to establish a link with are also declared within this file. The manifest file forms the fundamental block of the app project directory. It is necessary for a functioning app because it includes important information regarding the application

2.4 Android Sensor Programming

2.4.1 Overview

As mentioned previously, most android devices come equipped with certain sensor features, like the ability to monitor and record motion, orientation, and other environmental conditions. These features are collectively pigeonholed into three broad categories: Motion Sensors, Environmental Sensors, and Position Sensors. In order to be able to create a functioning light signal receiver, we must first be acquainted with the basic sensor programs that Android has to offer. These sensors are either hardware-based, software-based or both. The sensors may have some physical components built within the device that obtains the data directly by measuring specific parameters. Others, like the virtual sensors or synthetic sensors, might obtain their data from one or a combination of the hardware-based sensors. Android allows us to procure raw data from these sensors, which can then be employed in different applications to suit our needs. Android has some predefined classes that forms a part of the android sensor framework, to facilitate this.

2.4.2 Android Sensor Framework

Android sensor framework is used to access the sensors and retrieve the recorded sensor data. Android sensor framework belongs to the android Hardware package. The android sensor API comes with many classes and interfaces that carry out a variety of functions that help the sensors record data. The prominent ones are discussed as follows.

SensorManager class creates an instance of the sensor service by calling the method getSystemService() along with the argument SENSOR_SERVICE. SENSOR_SERVICE is a constant string that is used with the SensorManager class method to access the sensors. As a result, the system can list the sensors and it can also register and unregister listeners. This is accomplished by using the following code:

```
SensorManager mSensorManager;

mSensorManager= (SensorManager)getSystemService(SENSOR_SERVICE);
```

Afterwards, **Sensor** class is declared by calling getDefaultSensor(), a method that is a part of the Sensor Manager class, with the desired argument. This argument is also a constant string that describes the sensor type that is being instantiated. For instance, TYPE_LIGHT describes that a light sensor type is being used. Sensor class is used to access specific sensors and to determine their capabilities by calling relevant methods. This is achieved in the following way:

```
Sensor mLight;

mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
```

SensorEvent class is used to create a sensor event object. This returns information related to the sensor event. Raw sensor data and its accuracy are retrieved from the sensor event object. It also provides the system with information regarding the type of sensor that brought about the event and it's the timestamp. The values array contains the raw data of the sensor. Its length and value are dependent upon the type of sensor that is being observed and recorded. For a light sensor type, the values array (i.e. float[] values) contains the ambient light intensity measured in SI lux (lx) units whereas, it contains the atmospheric pressure in hPa (millibar) for a pressure sensor type. Whenever a change in the relevant parameters is observed, a sensor event is created. This

records the sensor type, raw data, and the accuracy of the data and the timestamp of the sensor event that took place.

SensorEventListener is used to monitor change in value or accuracy of the data being observed. It uses two callback methods for this purpose. The callback methods are notified through sensor events whenever the value or accuracy of the data being monitored fluctuates.

Each component of this framework is designated a set of tasks and they work cohesively together to record and monitor the sensor parameters. However, the availability of the sensor type must be ensured since most devices do not have all the sensors built into them.

2.4.3 Availability of Sensor Type and its Capabilities

It is possible to identify which sensors are available on the device. To confirm whether a sensor is available or not, the SensorManager class needs to create an instance to access the sensor. The method used to accomplish this is mentioned previously. An if/else conditional statement can be used afterwards to specify what actions to take if the sensor is available and what to do if it is not. The condition passed within the parenthesis can check whether the default sensor for the inquired sensor type is present within the device or not.

Another way of verifying the availability of the different sensor types is to list the sensor type by calling the getSensorList() method. TYPE_ALL is a constant that is passed as an argument of this method and lists all the sensors that are available on the device. To list all of the sensors belonging to a specific type, the argument can be changed accordingly e.g. TYPE_LIGHT, TYPE_GRAVITY etc.

Additionally, the capabilities and specifications of the sensor can also be determined through the use of certain methods belonging to the Sensor class. This ensures optimal use of the sensors by different applications. The power requirement information, the sensor's resolution and other attributes can help the developer tailor the application to suit the requirements and capabilities of the sensor. Sensors on different devices or different versions of the same sensor may have

different capabilities and attributes. The developer needs to adjust their applications accordingly for the optimal use of the sensors.

2.4.4 Sensor Events and SensorEventListener Callback Methods

One of the most important parts of Android sensor programming framework is the Sensor Event, which monitors the sensor's activity. It requests the assistance of the SensorEventListener and its callback methods to monitor the raw sensor data. These methods are namely onAccuracyChanged() and onSensorChanged(). These methods are called when there is a change in the sensor's accuracy or the sensor records a new value respectively.

When there is a change in accuracy, the onAccuracyChanged() method is called upon. Accuracy is represented by SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM, SENSOR_STATUS_ACCURACY_HIGH, or SENSOR_STATUS_UNRELIABLE. This refers to the Sensor object that has changed and updates the sensor's accuracy after the change. When there is a new value recorded by the sensor, a SensorEvent object is created by the onSensorChanged() method. This object retains the new value recorded along with the sensor type that brought about the value change and the timestamp of the event. The raw sensor data can be displayed using a TextView if needed.

The unregisterSensor() and registerSensor() methods are used to register and unregister sensors. When the sensors are not required by the application or the application is stopped, the sensor might need to be suspended to save battery life. The registerSensor() reestablishes the sensor features and the sensor can resume its function. When this method is invoked, it takes SENSOR_DELAY_NORMAL as one of its arguments. This is the default delay of the sensor; this can be altered to suit the developer's requirements. The default delay uses a delay of 200,000 microseconds. Other available delay speeds are SENSOR_DELAY_GAME (20,000 microsecond delay), SENSOR_DELAY_UI (60,000 microsecond delay), or SENSOR_DELAY_FASTEST (0 microsecond delay). Android 3.0 onwards can use absolute values in microseconds as the delay.

The concepts of the Android framework and Android sensor framework discussed up till now were used collectively to create the light signal transceiver application. This application uses the integrated Smartphone sensors (e.g. ambient light sensor) and the camera flashlight for short scale data exchange. The construction of the transceiver is discussed in detail in the next chapter.

3. DESIGN METHOD

3.1 Overview

This study deals with small scale data transfer using the camera flashlight and inbuilt ambient light sensor of an Android device. Data transmission is achieved by transmitting encoded data messages through light pulses of variable length. Each pulse generated is received by another Android device running the application. The receiver detects the ON/OFF light signals using the ambient light sensor built into the device. The application that was programmed for the purpose of this study is compatible with Android devices. The application is a transceiver that transmits and receives data signal. The transmitter is essentially a flashlight application that sends out a succession of long and short light pulses in correspondence to the message the user intends to transmit. The data message is encoded using the Morse code coding scheme for easy transmission of data with the aid of visible light communication. The receiver part of the application makes use of the ambient light sensor that is used in most devices to adjust screen brightness. The light sensor detects the encoded message and converts it back to the original data that was transmitted. Initially, we began the study with the transmission and reception of binary data bits (i.e. "1" and "0"). Later, the same application was modified and refined for the transmission and reception of short scale data messages.

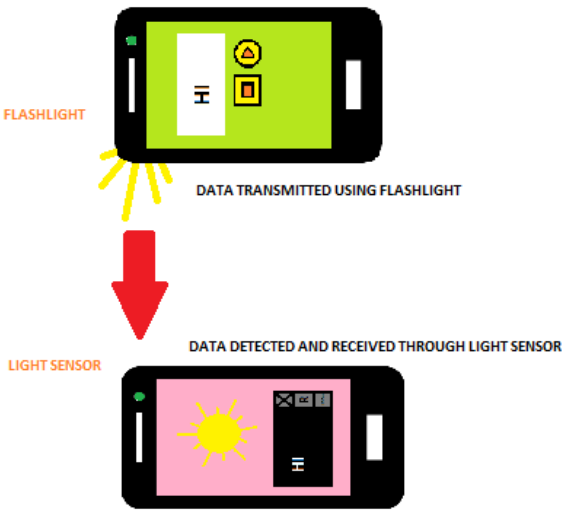

Fig: 3.1: Basic System Design

3.2: Transmitter Application Design

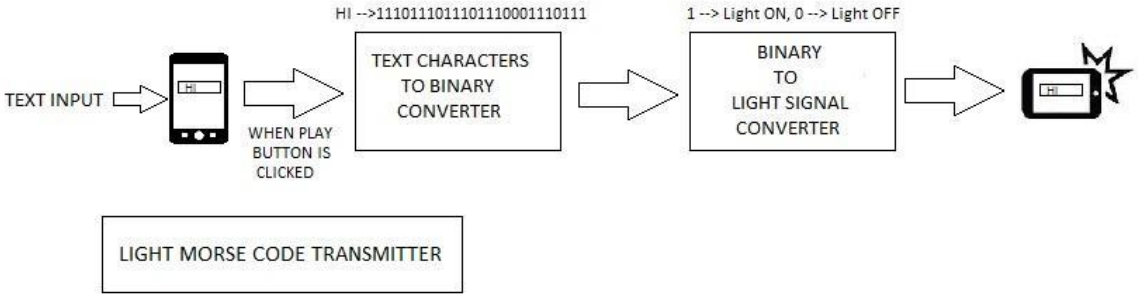

Fig: 3.2: Basic Transmitter architecture

3.2.1 Accessing the Flashlight

At the base of the transmitter is a simple flashlight application that uses the camera flashlight of the device to send out data messages. For the flashlight to turn ON and OFF when the application wants it to, the flashlight feature of the device must be accessed first. To achieve this, the application is required to request a set of permissions from the Android device. These permissions are listed within the manifest file of the APK file. It asks for the permission to access the camera hardware and related features that are required to toggle the device's camera flashlight. The following lines of code are added to the manifest file for this purpose.

```
<permission
    android:name="android.permission.FLASHLIGHT"
    android:permissionGroup="android.permission-group.HARDWARE_CONTROLS"
    android:protectionLevel="normal"/> -->


<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera"/>
```

Fig: 3.3: Permission to Access Camera Flashlight

Once the hardware and the features required have been accessed, the flashlight app makes use of the if/else conditional statements in the MainActivity.java file to turn the flashlight ON and OFF. The condition that is used within the parenthesis checks whether the flashlight is ON or OFF. A boolean named "isFlashOn" is initialized. It is set to false since the flashlight is OFF when the application is launched. When a button that is available on the application UI is pressed, it enters this if/else conditional statement. if the flashlight is OFF, it enters the "if(!isFlashOn)" and turns the flashlight ON. Consequently, isFlashOn is changed to true. When the button is pressed again, now that the boolean isFlashOn is set to true, it enters the "else if(isFlashOn)" and the flashlight is turned OFF. It also changes the isFlashOn back to false. When the back button is pressed, the application releases the camera hardware and features that were being used. The flashlight is turned OFF if it was ON and remains OFF if it was not turned ON. Pressing the back button closes the application. This is how the flashlight can be toggled with the help of a button. However, we need to send streams of data with the click of a button. Just turning the flashlight ON and OFF does not fulfill our needs.

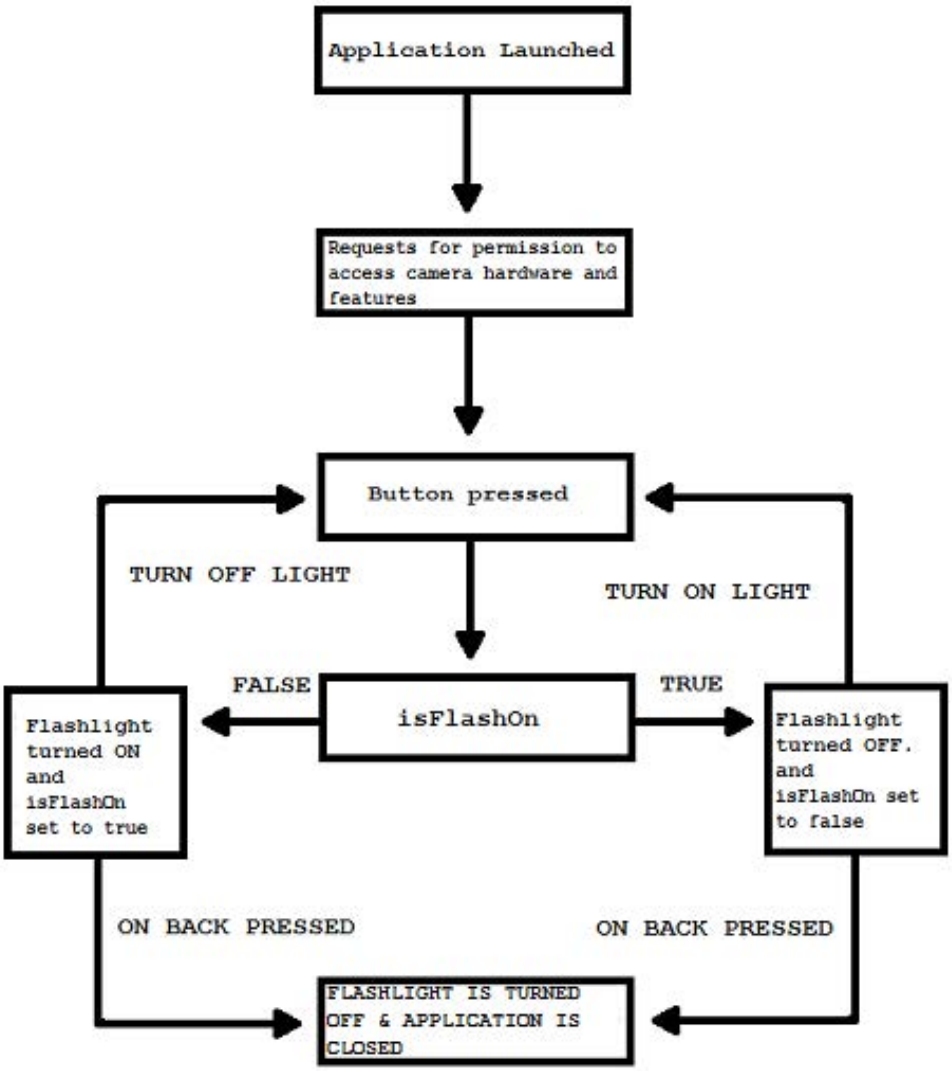

Fig: 3.4: Turning Flashlight ON and OFF

3.2.2 Transmission of Binary bits

In order to send streams of data with the click of a button, we need to make the flashlight fluctuate accordingly. During the initial development of the application, no input was taken from the user. This part of the study focused on making the flashlight strobe with the click of a button.

Hence, we took a fixed string variable as the input and included it in the app code. Consider that the string variable name is **myString**. The choice for the constant value of myString was "10101010101". This meant that when the button was pressed, a periodic light pulse would be transmitted by the flashlight. For each character of this string, the flashlight would either turn ON or OFF. For every "1" the light would be ON and for every "0" the light would be OFF. Hence, it is slightly different from the basic flashlight app that only checked whether the flashlight was ON or OFF using the isFlashOn condition. Here, the condition is changed to **myString.charAt(j) == '1'** to turn ON and **myString.charAt(j) == '0'** to turn OFF.

In order for the application to repeatedly turn the flashlight ON and OFF, the if/else conditional statement needed to be encapsulated within a FOR loop. This loop would have to continue for the entire length of the string value. Hence, the condition for the FOR loop would be something like

for(**int** i = 0; i<**myString**.length(); i++){

}

This means that the loop will repeat for the length of myString times, increasing the counter by one till it reaches the end. The integer number used as the counter is also used to specify the character position. So, for each loop that takes place, the character position of myString shifts one place. This character indicator is important since in each loop, the value present at that character position of myString determines, whether the light will be ON or OFF.

Since myString is given as "10101010101", the loop will repeat 11 times. The counter starts from 0 and with each loop increases by 1 until the counter reaches 11. With each loop the character position indicator also follows the counter. For instance, when the counter **i** is 0, **myString.charAt(i)** selects the character at 0 of myString. In this example, "1" is at position 0 hence the light will turn ON. The counter then increases by one. Now **i** is 1 and that does not infringe the looping condition hence, the loop runs again with **i** at 1 and therefore, **myString.charAt(i)** selects the character at position 1. Since the character at that position is "0", the light turns OFF. This goes on for as long as the looping condition is fulfilled. Once the condition is violated, the loop breaks. In this way, the flashlight can be programmed to strobe.

The amount of time that the light stays ON or OFF also needs to be specified. This value needs to be carefully selected so that it is received properly when we proceed to transmit data using this application. This is achieved using a Threadsleep() method that takes time delay in milliseconds as its argument. This is place within the loop and after the if/else conditional statement that turns the light On and OFF. A variable frequency can be set by using seek bar in the layout file and passing an argument of (250ms – blinkDelay) within the method, where blinkDelay is a variable that is dependent on the progress of the seek bar (i.e. blinkDelay = 2 * progress). This means the time delay range will be between 50ms to 250s. For simplicity, let's take the method delay argument as 200 ms instead. This value is preferred because it will not cause problems in the future when the Morse encoded data is received by the receiver. This assumption will be discussed later on in this chapter. This means that after completion of each loop, the application is forced to wait for 200ms before executing the next line of code. Therefore, periodic transmission of light pulses will be achieved where each pulse width is 200ms.

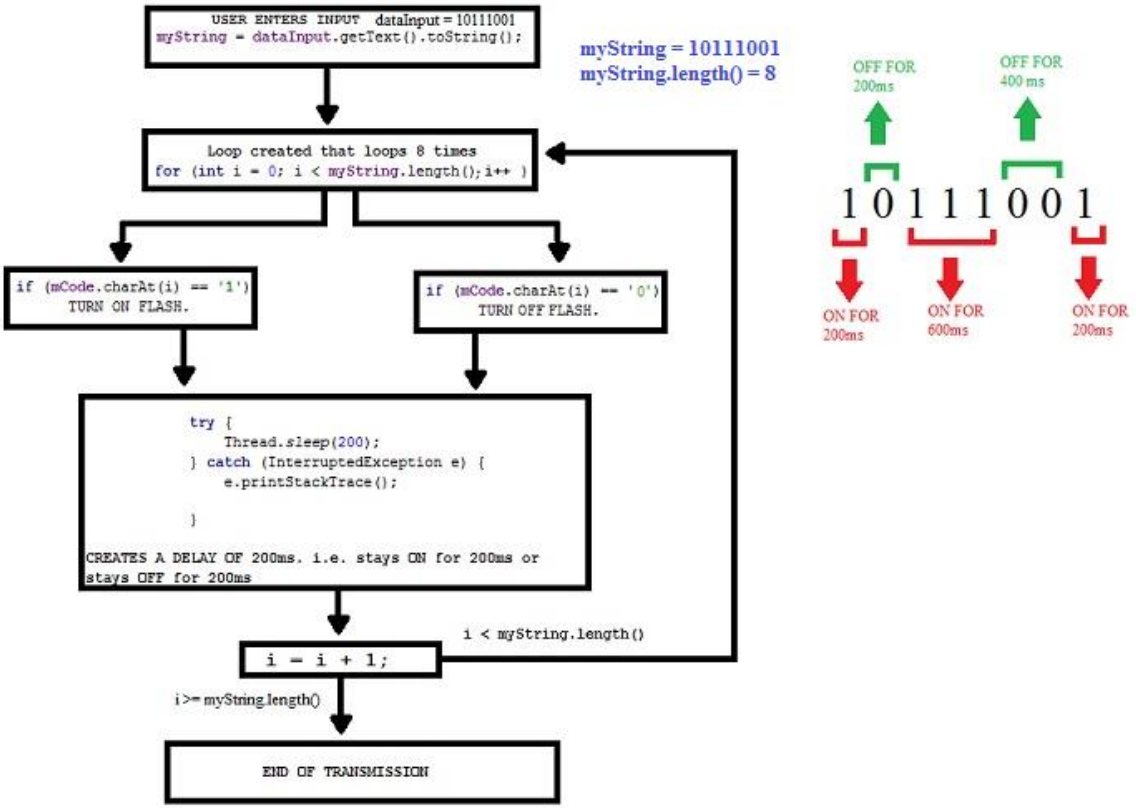

Fig: 3.5: Binary bit Transmission

After the flashlight started fluctuating at a desirable pace with the click of a button, it was necessary to omit the fixed string and take the user input instead. Hence, the myString variable was programmed to take the string variable from the user by using **dataInput.getText().toString()**, where dataInput is the name given to an EditText box where the user enters the data input. The input is saved in the dataInput variable which is then placed in myString using above mentioned command like

myString = dataInput.getText().toString();

Now, instead of a fixed string, the flashlight will fluctuate based on the input entered by the user. This means that, the loop will now run for the length of the entered data. For now, the input is limited to binary bits "1" and "0". It is important to note that the input command must be placed within the loop for it to work. Using these binary bits, letters and words will be encoded into their Morse code counterparts for data transmission.

3.2.3 Implementation of Morse Code

As discussed previously, each unit in Morse code is highly dependent upon the pulse width of the signal being emitted and received. Consequently, varying the pulse width would also alter the unit conversion. It is wise to imagine each ON/OFF signal in terms of its binary equivalent, where every ON signal represents a '1' and every OFF signal represents a '0'. For example, if each pulse width is 200ms, every 200ms ON time is represented by a binary bit of '1' and every 200ms OFF time is represented by a binary bit of '0'. Therefore, a binary sequence of "10111" would result in the light being ON for 200ms, then OFF for another 200 ms and ON for another 600ms. This concept will be useful during the construction of the Light signal transmitter. Using the unit conversion, Morse code can also be represented in its binary counterpart by following a few rules.

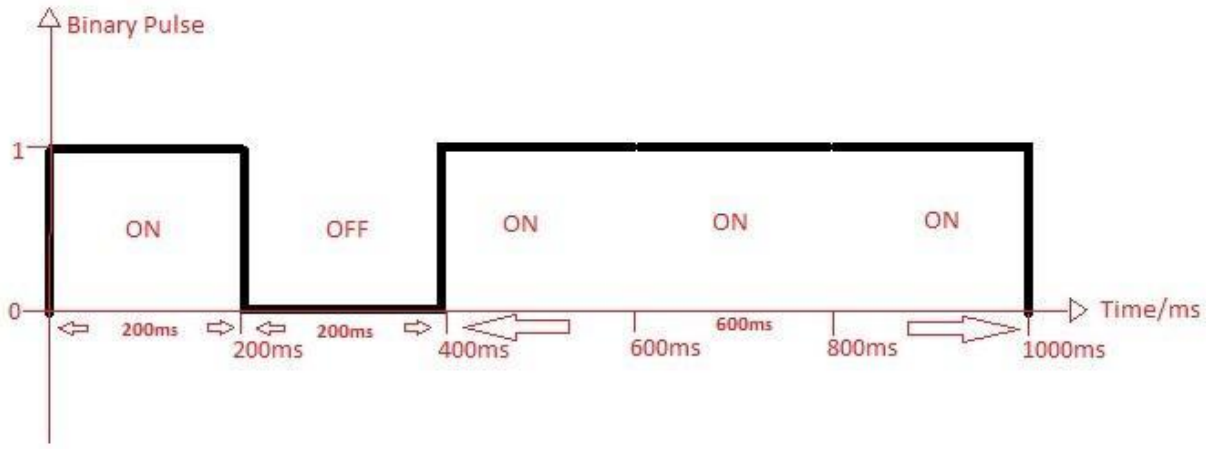

Fig: 3.6: Binary Pulse representation of the "10111" Binary Sequence

These rules are enumerated as follows:

- '1' and '0' is the single unit used for the conversion.
- A dot is represented by a '1' binary bit.
- A dash is represented by three consecutive '1' bits in a sequence i.e. '111'.
- Space required in Morse code within each letter is represented by a single '0' bit.
- Space required in Morse code between two letters is represented by three consecutive '0' bits in a sequence i.e. '000'.
- Space required in Morse code between two words is represented by seven consecutive '0' bits in a sequence i.e. '0000000'.

For example"**-. ... -**" signifies "TEST" in Morse code. If it was to be represented in its binary form, it would be "11100010001010100011 1". This concept of Morse code being represented in its binary form is useful when we construct the transmitter since devices respond better to binary codes.

These guidelines delineate how each letter and character can be transformed to their corresponding Morse code using binary bits. The dots and dashes have been conveniently transformed into short and long light pulses respectively. This is exactly how the international Morse code has been followed to encode the data that is entered for transmission. "1" represents

a short light pulse and "111" represents a long light pulse. Once this conversion is implemented within the transmitter, it can be used to transmit any letters, numbers or characters individually or in combination. The conversion is manually written into the application code. Few of the conversions are listed below all of the Morse code will use the same conversion technique:

LETTER/CHARACTER	MORSE CODE	MORSE CODED BINARY FORM
A	.-	10111
B	-...	111010101
C	-.-.	11101011101
D	-..	1110101
BAD	-... .- -..	11101010100010111000111010 1

Fig: 3.7: Table of Conversion

3.2.4 Transmission of Morse Coded Data

When multiple character input is taken from the user and not only binary input, the code needs to be readjusted to implement the conversion of the characters to its Morse code binary form. The FOR loop used for binary bit transmission before is left more or less unchanged but it is placed inside another FOR loop. The minor change that occurs within the former FOR loop is that, instead of the loop running for the length of the user input (i.e. **myString)**, it runs for the length of **mCode** instead. This is a variable that is defined to save the Morse coded binary form of each character of the input data after conversion. Since it is housed within another FOR loop, the **mCode** only contains the conversion of the current character of the input data and toggles the flashlight accordingly. The new FOR loop repeats for the length of the input data that is saved in **myString**. This loop converts the specific character that is selected, using the counter as the character position pointer, to its Morse coded binary form. The Morse coded binary form is then saved in the **mCode** variable. Afterwards, the second For loop is entered using **"i<mCode.length()"** as the loop condition. The light now flashes ON and OFF according to the saved binary Morse code of the character. Once the entire loop has been executed for the

condition mentioned times, the first loop counter increases by 1 and the character position pointer shifts by one place, the first loop is again executed for the next character input. This character is also converted to its Morse coded binary form which replaces the previous conversion in the **mCode** variable. It then enters the second loop again and executes it for the length of the new **mCode** and the light again fluctuates accordingly to transmit the data. This happens over and over again for the entire length of the user's input.

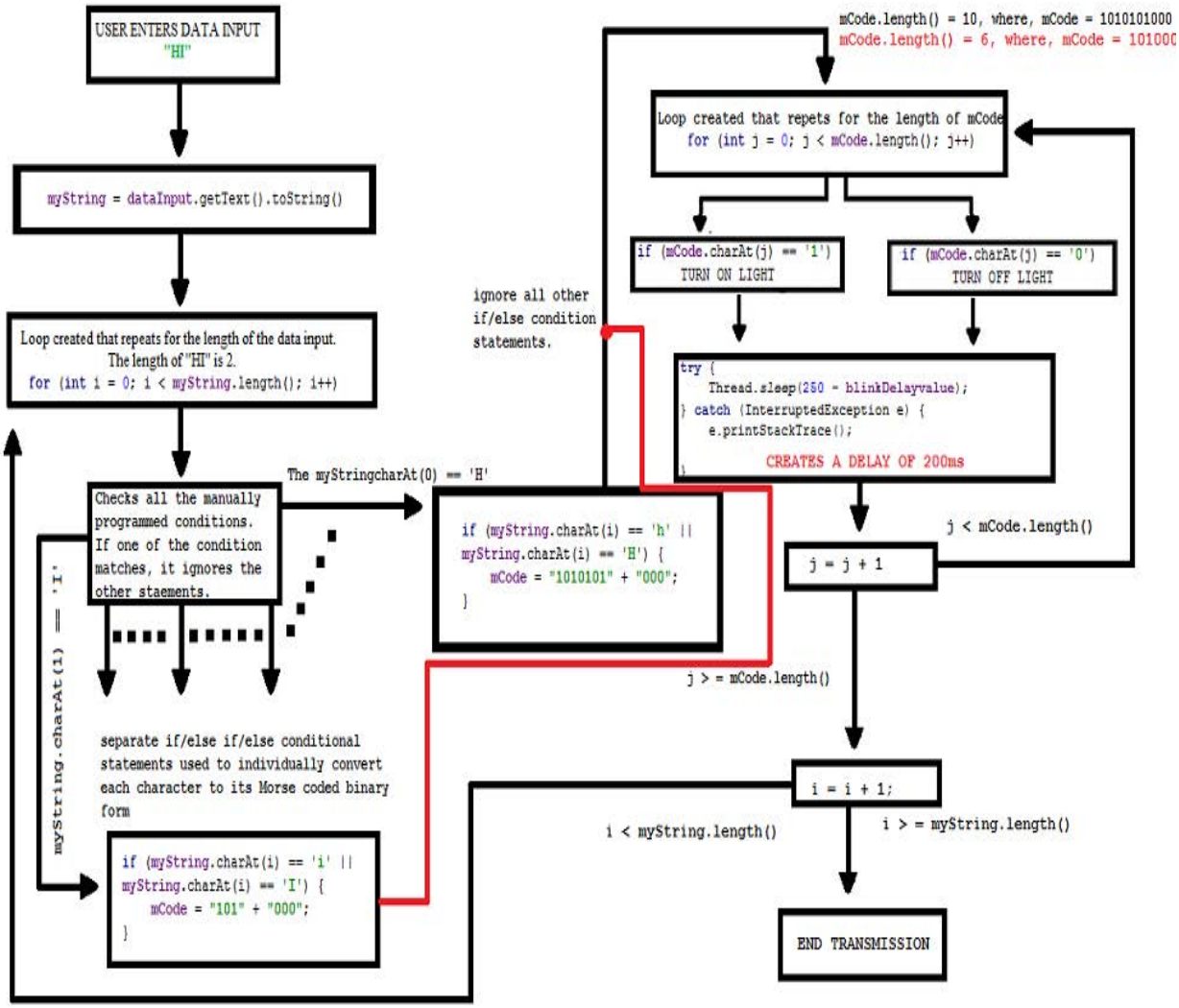

Fig: 3.8: Morse code Light Signal Transmission

This transmission of data can be better explained using an example. For instance, if we were to transmit the word "HI" through this transmitter, the user would have to enter the word in the empty box available on the UI of the application and press a button to transmit. As soon as the user presses the button, the first loop is initiated. Here, the data input is saved within a variable called **myString.** The variable now consists of a word that has two characters "H" and "I". The length of myString is 2. Hence, the first loop will be repeated two times. Array index position starts with a 0. Similarly, character index position also starts with a 0. The loop starts with a counter, **inti**, initialized at 0. The first FOR loop contains if/else if/else conditions that are used for the conversion of each of the character to its Morse coded Binary form. Each character is converted in each loop cycle. Since at the beginning, **i=0,** the character at position 0 is checked with one of the if/else if/else statements. Whenever there is a match with the inquiry, the code within the conditional statement is executed. Since the character at position 0 is an "H", the following condition is fulfilled:

"if (myString.charAt(i) == 'h' || myString.charAt(i) == 'H') { mCode = "1010101" + "000"; }"

Hence, **mCode** now contains "**1010101000".** Since one of the conditions has matched, it ignores the others and moves on to the subsequent FOR loop that is housed within the first loop. It then executes the FOR loop just like before. The loop repeats for the length of mCode, which is 10 times. The Thread.sleep(200) ensures that each pulse width is 200ms. The light is ON for 200ms if there is a "1" bit and OFF for 200ms if there is a "0". Consecutive three 1's will result in light being ON for 600ms and consecutive three 0's will result in the light being OFF for 600ms. Once the entire mCode is transmitted through the flashlight, the code returns to the first loop code and increases **i** by 1 (i.e. **i=1**). When it reaches the end of the loop, it checks whether the condition is fulfilled. Since i=1, it still fulfills it so the code enters the first loop again. This time, it again checks whether one of the manually programmed conditions matches. Since the character at position 1 is now "I", it fulfills the following condition:

if (myString.charAt(i) == 'i' || myString.charAt(i) == 'I') {

mCode = "101" + "000";}

It ignores all the other statements when one of them is met. Now the **mCode** is replaced by the Morse coded binary form of "I". So, **mCode** now holds **"101000"**. The next loop commences that will repeat 6 times (i.e. the length of **mCode**). Each time the light will be ON or OFF for 200ms depending on the character that is present at the index character position. The light is ON for 200ms if there is a "1" bit and OFF for 200ms if there is a "0". Consecutive three 1's will result in light being ON for 600ms and consecutive three 0's will result in the light being OFF for 600ms. After the second FOR loop goes around 6 times, it returns to the first FOR loop. Here the counter increases by one again (i.e. i=2). Once the end of the first loop is reached, the condition of the FOR loop is checked again (i.e. is **i**<myString.length()). Since the length of myString is 2, the condition is not met and the loop ends.

The transmission of the user entered data is complete. This will now be received by the receiver on the other end. The receiver design is less complex than the transmission. The encoded message is decoded by the receiver with the click of a button.

3.3 Receiver Application Design

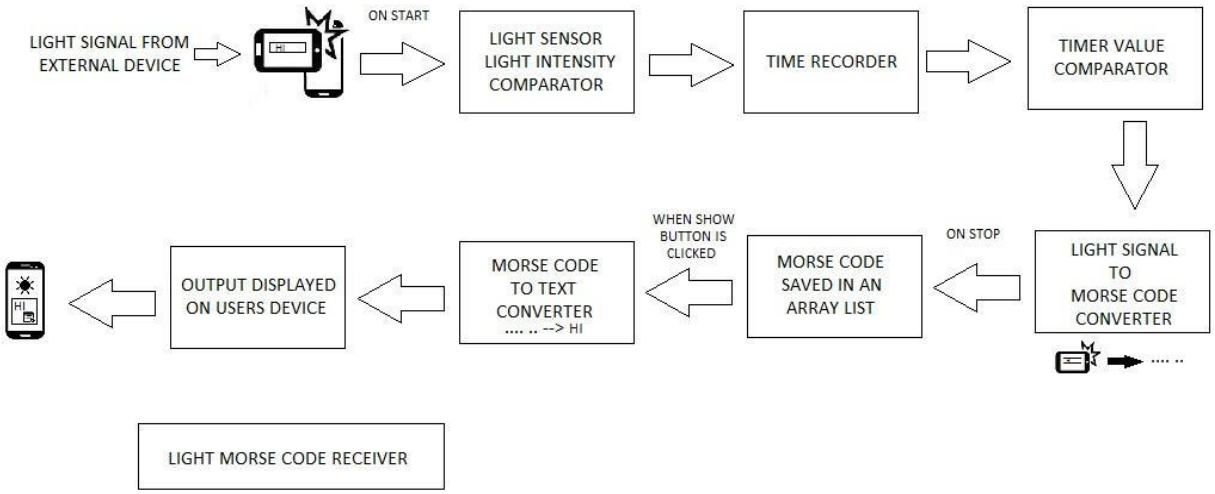

Fig: 3.9: Light Signal Receiver and Decoder Architecture

The receiver makes use of the inbuilt ambient light sensor available in Android platform devices. The light sensor is accessed by the application to receive the transmitted light pulses. As the flashlight shines upon the receiving device, the change is recorded by the sensor. Whenever there is a change in the light signal, i.e. light turning ON or light turning OFF, a sensor event is created that calls the onSensorChanged() callback method. This is where the necessary code is added to record the incoming Morse code light pulse. At first, the light pulse is just received as it is and saved in an Array List. When a button is clicked, it decodes the Morse code and displays the deciphered message on the screen.

3.3.1 Ambient Light Sensor Design

The receiver is essentially a simple light sensor application. The raw data obtained from the light signal does not need to be calibrated, filtered or modified. The receiver combines the time and light intensity to receive the signal. To create the basic light sensor app, we need to create an instance of the SensorManager class. A Sensor class is then used to access the light sensor. This was achieved by passing Sensor.TYPE_LIGHT as an argument of the getDefaultSensor() method. Within the onResume() method, the sensor listener needs to be registered and it must be unregistered when the app is paused. Hence, the onPause() method will have to include the code that unregisters the sensor listener. SensorEventListener has two callback methods that receive sensor events whenever there is a change in the accuracy or the value. Hence, both the onAccuracyChanged() and onSensorChanged() callback methods must mentioned in the code. Incoming sensor data can be modified or used within the onSensorChanged() method as desired by the developer.

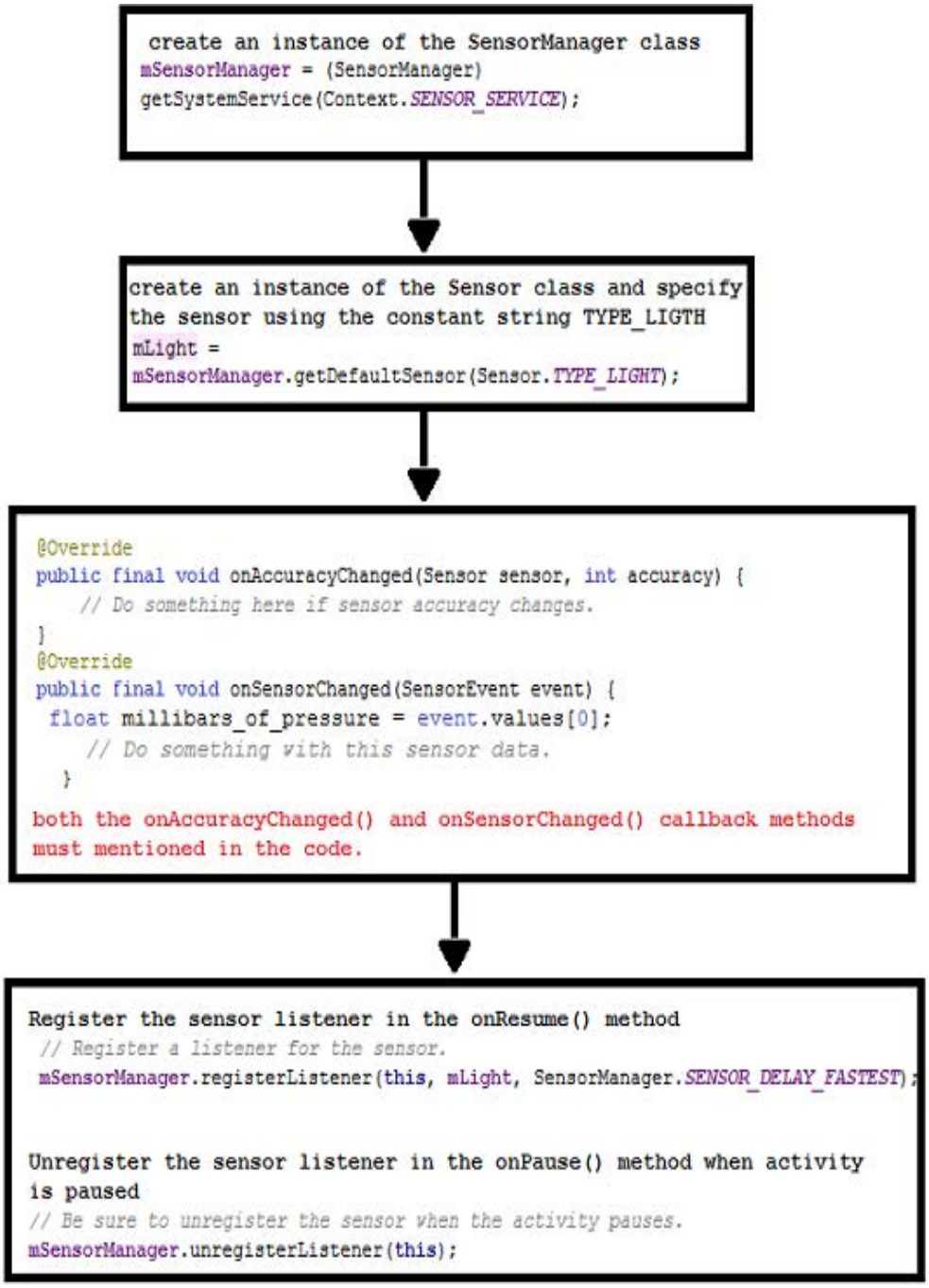

Fig: 3.10: Basic Light Sensor

3.3.2 Modification of the Ambient Light Sensor

The modification of the light sensor data will be done in the onSensorChanged() method. The ambient light sensor records the immediate light intensity. This recorded data is saved an array named values. The "**event.values[0]"** provides the user with the ambient light level in SI lux units. The receiver will use the OOK (On-Off Keying) method of detection. When the light level is above a given value, the flashlight on the data transmitting device is considered to be ON and when the light level falls below this specified value, the light is considered to be OFF. This event.values[0] is used to obtain the current light level and it is saved in a variable. For the ease of discussion, we will call this variable lightLevel. The lightLevel is used within if/else conditional statements. This value is compared against a fixed value that is picked with careful consideration of the backlight. If the surrounding light intensity is not canceled out, it can interfere with the data reception. The fixed light intensity chosen was 500 lux. When the lightLevel is above 500, there is an incoming light and if the lightLevel is below 500, there is no incoming light. So, now that we have a condition that detects whether the light is on or not, we need to record how long each light pulse lasts. We obtain the exact time when the light turns ON (we will call this **startTime)** and the time when the light turns OFF(we will call this **endTime)** using the **System.currentTimeMillis()** command. This is used within the IF statements that checks when the light is ON and when the light is OFF. We then calculate the time of each of the light pulses (we will call this **timeDifference**) by subtracting the startTime from the endTime (timeDifference = endTime – startTime). Within the light OFF condition, each light pulse width time is again checked using new conditional statement. The transmitter transmits light pulses of variable pulse width. A dash is represented by a 600ms pulse width light pulse and a dot is represented by a 200ms light pulse. Hence these new conditions will check if the light was ON for 600ms or less. If it was on for more than 600ms, a dash is added to and array list and if it was ON for less than 600ms, a dot would be added. The result is saved in an ArrayList since it makes data retention dynamic.

Similarly, the gaps in the light pulses are also recorded. This is calculated inside the condition that runs when the light is ON. A new variable (named timeGap) records the time when

there exists no light pulse. To calculate this we subtract the endTime from the startTime. This gives us the time gap between each pulse. There are a set of new conditional statements that run when the light is ON. This check the gapTime. The space in between Morse code is used to separate letters, words and sentences. Similarly, here we will consider a gapTime between 450ms to1050ms to be the space between letters in a word, and a gapTime that exceeds 1400ms to be the space between two words. To differentiate between the two, we will add one blank space to the array list for the space between letters and two blank spaces for the space between words. Now that the reception is complete, the data is saved in an Array List that can then be decoded when needed.

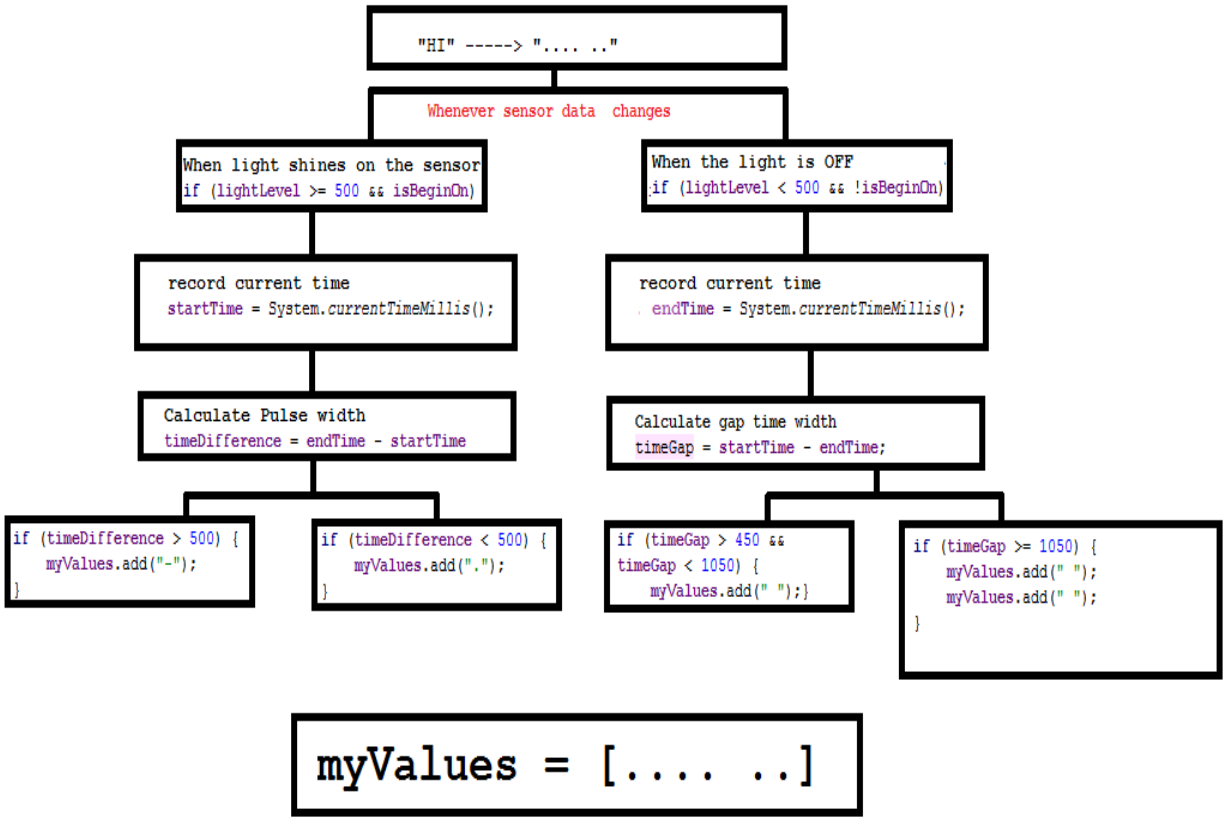

Fig: 3.11: Reception of Transmitted Light Pulses

For example, if we are receiving a data message "HI". The transmitting device will send Morse code light pulses of "**. . . . . .** ". Each dot will be sent in the form of a 200ms long light pulse and every dash will be a 600ms long light pulse. The gap in between the pulses will be 200ms, 600ms and 1400ms. The gap in between letters is 200ms, between two letters is 600ms and between two words is 1400ms. As the light is being received, when the lightLevel is above 500, the time is recorded. As soon as the light pulse ends, the lightLevel goes below 500 and the exact time at that moment is recorded. The timeDifference is then calculated. Since a 200ms pulse was sent, the timeDifference would also be around 200ms. Since the timeDiffrence is below 600ms, a dot will be registered in the Array List. Again, the lightLevel>500 condition is met and the gapTime is calculated. Since the gapTime recorded is 200ms, the application does nothing and waits for the light to turn ON again. When the light turns ON again, the timeDifference is yet again calculated and since it is 200ms, the Array List registers another dot. This happens four times and four dots are registered that translate to the "H" in Morse code. Now when the light turns OFF, the gapTime is around 600ms. This means that the transmission of a letter has been accomplished and now another letter will be transmitted. The application now adds a blank space to the Array List, such that the contents are now "**. . . .** ". Afterwards, the "I" will be received just like "H" and the Array List will contain "**. . . . . .**".

After the light pulses are received by the receiver, it is stored in an Array List. The light pulses are not decoded while they are being received. Once all the data has been received, then the data can be decoded with the press of a button.

3.3.3 Deciphering the Morse code

The received light pulses are saved in an Array List. The name given to the Array List is "**myValues**". The received message is saved in its Morse code form within the Array List. The data transmission has to be completed before deciphering the code to display the message. The Morse code conversion is done within a FOR loop. The predefined Morse code conversions are

manually programmed inside this loop. The loop runs for the length of the myValues Array List. It checks the combination of dots and dashes in the Array List and replaces them with the corresponding letters or characters. This loop keeps running until the data is decoded. Once it has been deciphered, it displays the message.

One thing to keep in mind is that it checks the set of dots and dashes and not the dots and dashes individually. In other words, the string of dots and dashes in between the spaces are checked. A single space at the beginning and end of the string of dots and dashes signifies that it represents a letter. So with each loop, a letter is decoded and added to a new Array List called "**myResult**". All the letters are then added together to form the decoded message that was received and is displayed on the screen.

For example, if the receiver app received the Morse code light signal of "HI", that is "**. . . . . .**". The Array List myValues will contain the Morse coded data as "**.... ..**". When the SHOW button is pressed, the FOR loops runs. The integer h is initialized at 0 and used as the counter and index position indicator. The condition states that it will run for the length of myValues. Hence, the loop is programmed to run for 8 times. It will not run exactly the specified times because the counter value will increase as per the Morse code letter length. The FOR loop contains if/else if/else statements that contain the conversion. If one of the conditions is fulfilled, it executes that code and ignores all other conditions. In this case, the following condition is met. Hence the code it encompasses is executed

```
else if (myValues.get(h).equals(".") &&myValues.get(h + 1).equals(".") &&myValues.get(h + 2).equals(".") &&myValues.get(h + 3).equals(".") &&myValues.get(h + 4).equals(" "))
{
  myResult.add("H");
  h = h + 5;
}
```

As you can see, the contents of each position of myValues are checked. Since myValue has "**....**" the values at position 0, 1,2,3,4 are "dot dot dot dot space". This Morse code translates to an "H"; hence, an "H" is added to the myResult Array List. As mentioned before, the counter does not increase uniformly. It increases by the character length of the Morse code of the letter including the space at the end of the Morse code. Since the Morse code of "H" is "**....**" the counter will increase by 5. The Morse code has 4 characters and including the space makes it 5. Now that the counter is 5, it checks the looping condition. Since 5 is less than the length of myValues (8), it will loop again. Now, the remaining myValues is being checked the same way. The integer value of h is now 5. Hence the following condition is met and the code executes:

```
else if (myValues.get(h).equals(".") &&myValues.get(h + 1).equals(".") &&myValues.get(h + 2).equals(" "))
{
myResult.add("I");
h = h + 3;
}
```

The values of myValues Array List at positions 5, 6 and 7 are "dot dot space". This Morse code is converted to an "I" and an "I" is added to myResult Array List. The counter increases by 3 since the character length of the Morse code of "I" is 2 and including the space makes it 3. After the counter increases by 3, the counter is now 8. Since h is now 8, it no longer fulfils the looping condition (h<myValues.size() where myValues.size()=8). So, the loop breaks and the code after the loops are executed. All the contents of the myResult Array List are summed together within a loop to form the complete data message. The data message is saved as a string variable and the string value is displayed on the user's screen.

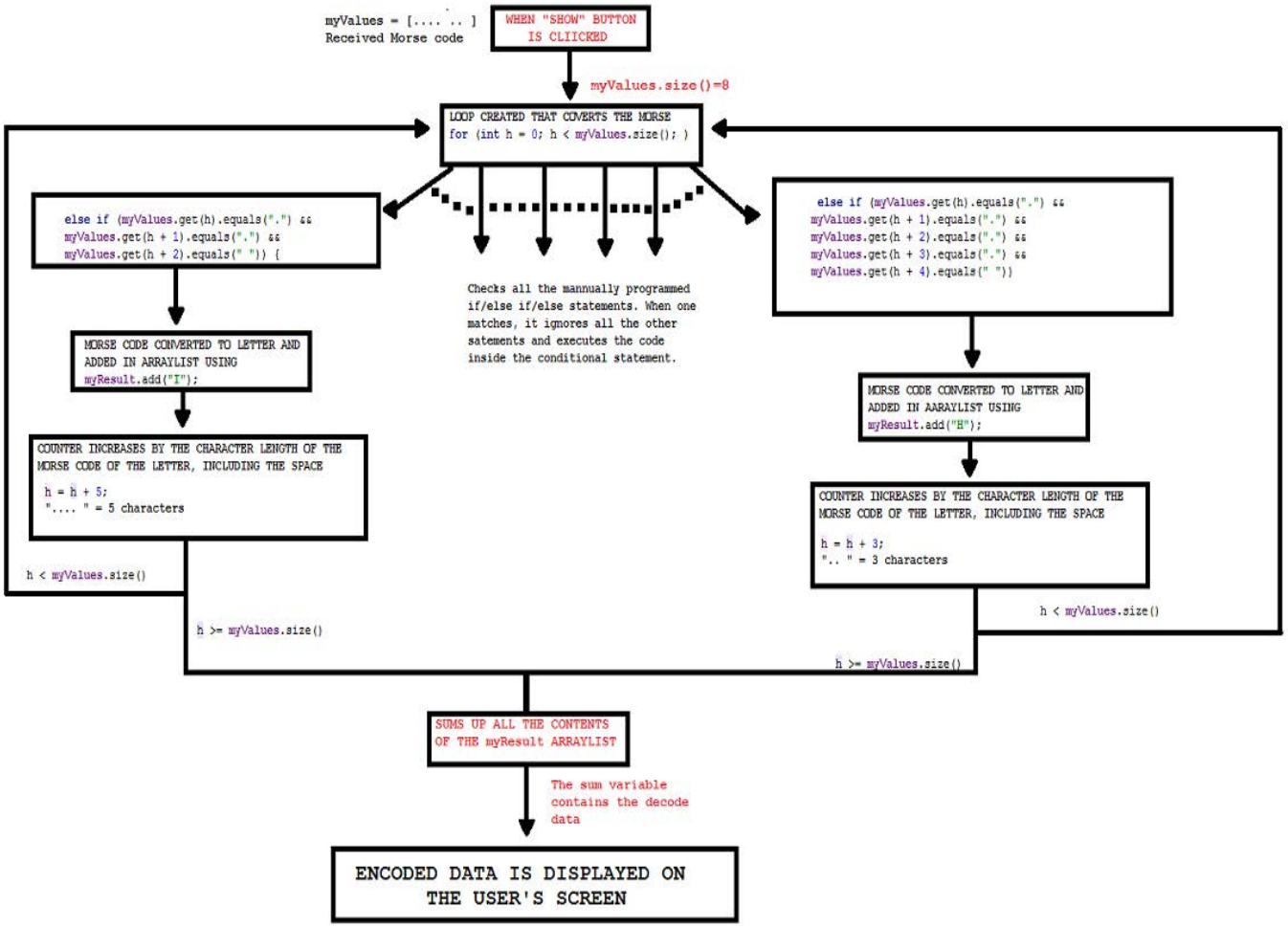

Fig: 3.12: Received Data Decoder

This completes the entire process of data transmission and reception. The received data is decoded and displayed on the user's screen. During the designing of the transmitter and receiver, there were a few assumptions that were made. We need to discuss the reasons in details to understand why some values were fixed and such assumptions were taken into consideration.

3.4 Assumptions and Challenges

During the design process and preliminary testing of the application, certain values were taken into consideration for accurate transmission and reception of the data. We were mainly concerned with the accuracy of the data during transmission and reception. As a result, the speed was traded off. The data was transmitted at variable pulse width (i.e. 50ms, 100ms, 150ms and 200ms). Data transmission was initiated using the lowest value of the pulse width. The unit conversion of the Morse code that has been discussed previously was used accordingly by the application for the different pulse widths. The same data message was sent and received using different pulse width. We started with the lowest value and the value was gradually increased to monitor any changes. It was observed that for 200ms, the data accuracy was the best. Values lower than 200ms would result in incorrect detection of dots and dashes because, the gap time between two dots, two dashes or a dot and a dash was insufficient. Even though the light was OFF, the sensor had troubles detecting since the gap time in between letters was insignificant. So, 200ms was chosen as the transmission rate.

Furthermore, the On Off Keying detection method used by the receiver used a fixed light level as the reference level. If the light sensor records any value above the reference, the light is considered to be ON and if the sensor records anything below it the light will be considered to be OFF. This reference level had to be place above 0 lux level because most of the time, there is ambient light. To avoid interference from the surrounding light, the level should be high enough so that the sensor does not consider the light to be perennially ON. Hence, it was selected to be 500 lux, considering the brightest backlight scenario.

The detected transmitted light pulses were stored in the form of Morse code dit and dah or dot and dash. Light pulse ON and OFF time are calculated within the application. The pulse time was used to differentiate between a dot and a dash. The gap time was used to recognize whether a word, letter and sentences. However, during programming, the exact pulse width conversion could not be applied. A range was set to compensate for the time lost between data transmission and reception. For example, even though the pulse width for a dot is 200ms and dash is 600ms, a dot and dash were observed to be 200 to 300ms and 600 to 700ms long. As we already know, the

gap in between letters was 200ms, between two letters was 600ms and between two words was 1400ms. However, the time lapse recorded by the application was not always exact. Hence, a range was used for detection within the conditional statements that saved the received data in an Array List in its Morse code form.

In addition to that, we also had to ensure that data transmission could be aborted when needed. In the case of incorrect data being entered, the user might need to immediately abort the transmission. So this was tackled with the use of a button. During data transmission, as soon as the button was pressed, it would turn on a flag within the application which indicates that the app needs to immediately break the loop and it complies.

4. OUTCOME OF THE THESIS

4.1 End Product

4.1.1 Overview

The application prepared for this study is a transceiver with the purpose of transmitting and receiving small scale data messages with the help of visible light communication. The transmitter part of the application takes the help of the built-in flashlight of the device to transmit a series of light pulses of variable length corresponding to the data the user wishes to send. The data input by the user is encoded using the Morse code coding scheme and then transmitted by using the flashlight to send out a light signal encoded with the message. The receiver, using the ambient light sensor of the device, detects the transmitted light pulses and records these readings, then proceeds to decode it to obtain the original message that was sent by the user.

4.1.2 Transmitter User Interface

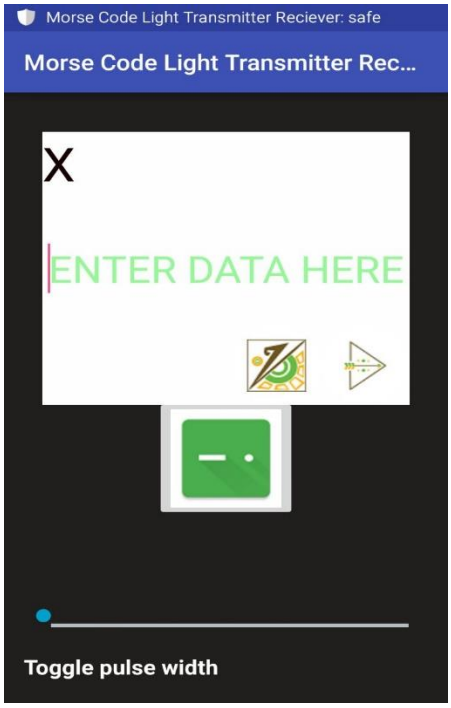

Fig: 4.1: Transmitter User Interface

The transmitter includes an edit text box where the user inputs the message they wish to be transmitted. The cross in the white edit text box clears the data entered. There is also a toggle bar in the transmitter with the help of which the pulse width of the light pulse can be modified. The start button included initiates the transmission of the data signal once the data is input, whereas the stop button can be used to terminate the transmission in case the user has made an error while entering the data and wishes to end transmission and start anew. The transmitter is also equipped with a button, the one with a dot and dash in the middle of the screen that is used to shift to the receiver part of the application.

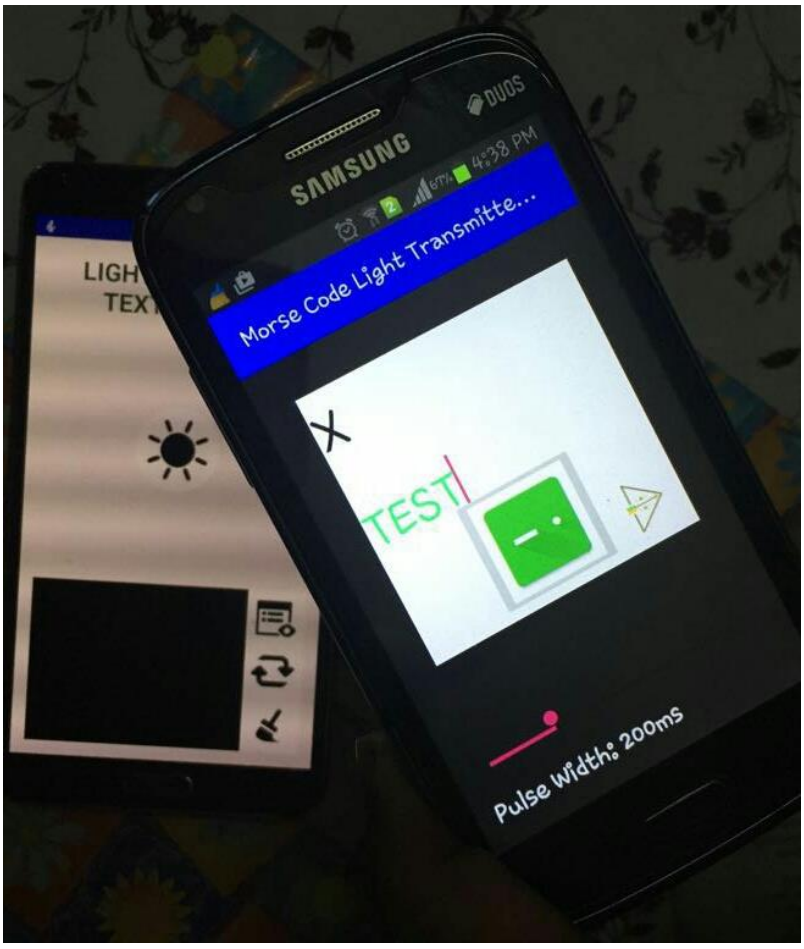

Fig: 4.2: Data being transmitted

4.1.3 Receiver User Interface

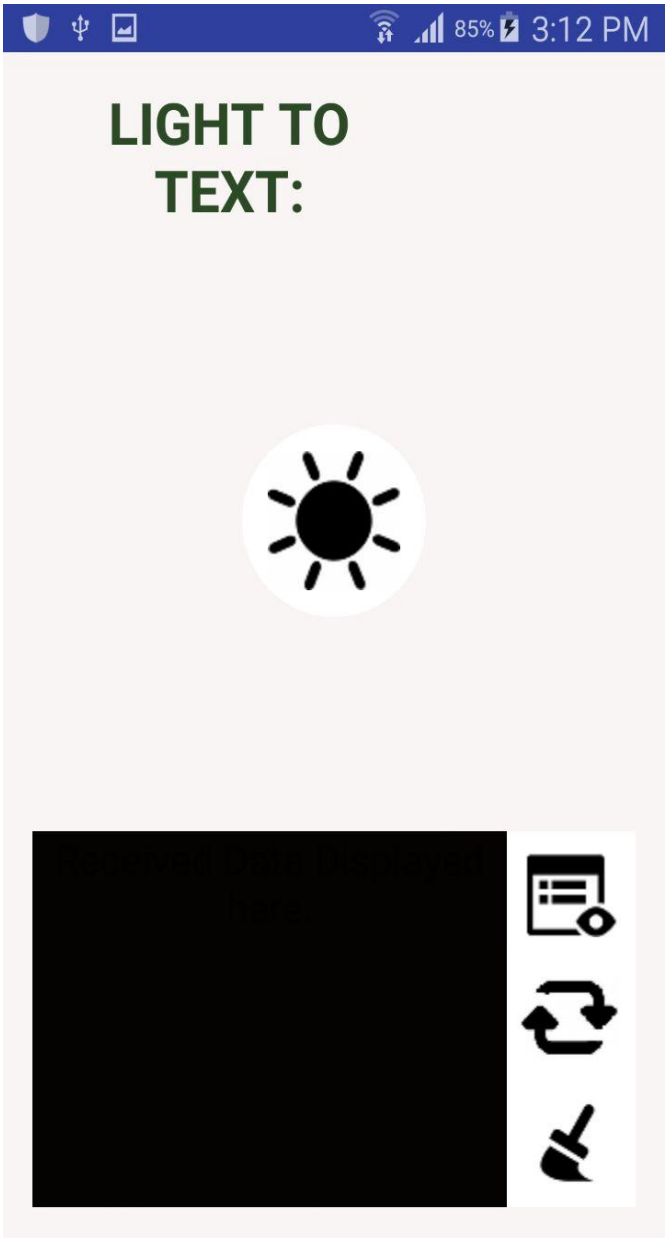

Fig: 4.3: Receiver User Interface

The receiver is comprised of a light sensor ON and OFF button, situated in the middle of the screen that needs to be switched on at first in order to start the detection of light pulses and begin receiving the data message. Switching off this button results in the receiver to stop detecting the light pulses once the transmission is over, or in the middle of transmission in case of error or if the user wishes to terminate the reception. After receiving the data message the show button is pressed which then displays the decoded message in the black Text View box of the receiver. A clear button included in the receiver can be pressed to clear the displayed message in the Text View box, whereas the replay button can be used to display the previous message again. Lastly, pressing back takes us back to the transmitter part of the application again.

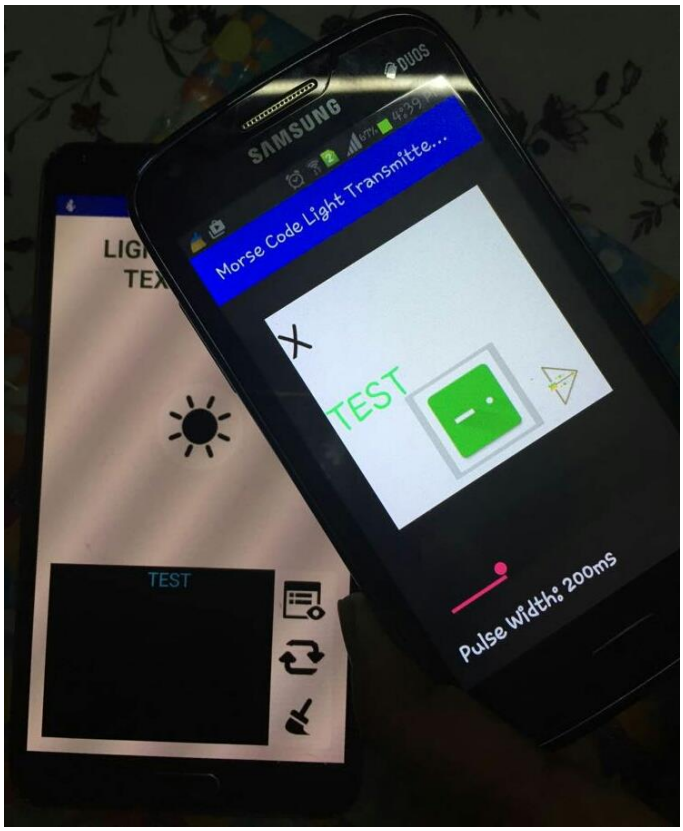

Fig: 4.4: Received Data Displayed

4.1.4 Tests Performed on Application

Considering the application uses the Morse code as its coding scheme, the pulse width of the transmitted and received signal had to be chosen after careful consideration. Using Morse code means not only is the duration of the light pulses important but the gap timings between each light pulse also need to be precise in order for proper and error-free transmission and reception of data signals since '1' and '0', which correspond to an ON signal and an OFF signal respectively, are taken as the single unit used for the conversion in this study as explained earlier. To ensure that data was being transmitted and receiver properly a series of tests were carried out to assess the performance of the application.

During the initial tests, a Text View box was included in the receiver UI which displayed two separate timers on the screen. Timer 1 displayed the calculated time of each pulse of light detected. Timer 2, on the other hand, displayed the time between the pulses of light, i.e. the gap timings. With these two timers, it was possible for us to observe the duration of individual dots and dashes received, as well as the gaps in between them, all of which are important in the transmission and reception of data. This made it easier to virtually see the effect and identify any problems with the reception of data so we could make the necessary adjustments.

To investigate the accuracy of the data transmission and reception and determine the optimum pulse width for transmission of the data signal, we sent known data messages, at first a single letter and then a word, at various pulse widths and observed the displayed message at the receiver to see if the expected result was obtained. In case of a failure to do so, we sent the same message again while keeping a close eye at the timers to identify exactly where the error was occurring.

Starting with the lowest pulse width at 50ms, we transmitted the letter 'A' using the transmitter. The Morse code for "A" is "**.** **–**". Considering this, if the data is received properly, the letter "A" would also be displayed in the receiver while timer 1, which shows the calculated pulse time, would give a reading of 50 ms and 150 ms, corresponding to the dot and dash respectively, and timer 2 would give the gap timing as 50 ms for the gap in between the light pulses for the dot and dash.

In our case, however, it was observed that instead of the timers showing the above-mentioned readings, timer 1 gave a reading of 200ms and no gap time in timer 2 was recorded. The gap time of 50 ms between the dot and the dash was too small for the light sensor in the receiver to detect properly. Consequently, the data message was recorded as "–" and not as ". –", giving us an incorrect result.

Making the necessary changes to the receiver, the experiment was repeated at higher pulse widths of 100ms, 150ms, and so on while verifying the results. It was found that errors, while gradually less frequent, still occurred during reception for each increment of the pulse width until we reached pulse width of 200ms. At this pulse width, the accuracy was found to be the best, and hence was chosen as the preferred pulse width while transmitting and receiving data. For 200 ms pulse width, a dot would be 200ms, a dash would have a pulse width of 600ms.Similarly, the gap in between letters is 200ms, and 600 ms between two letters while it was 1400 ms between two words. These corresponding changes in the unit conversions were made in the receiver so that it could decode light pulses of said pulse width. The timers were then removed from the receiver UI.

Moreover, since our application uses a fixed light level as the reference level for the On Off Keying detection method used in the receiver, we had to make sure the surrounding ambient light did not affect the reception of data in any way. The light is taken as ON if the light sensor in the receiver detects any value higher than the reference, otherwise the light is considered to be OFF. Thus it was necessary that we used a reference level that was higher than the surrounding light level of any environment we might come across while using the application to transfer and receive data so that the sensor only detects the light to be ON while we are transferring data, and the surrounding light does not interfere with the reception of the signal.

To determine the reference level, another Text View box was initially included in the receiver which displayed the light level in unit of Lux. At first, the flashlight was shone on the light sensor of the receiver and the reading noted down. The reading when the flashlight was shone was around 1000 Lux to 70,000 Lux. This meant that we had to take a reference lower than 1000 Lux to ensure the receiver will consider light to be ON when the flashlight shines on the sensor. The receiver was then taken to different ambient light conditions and the light level noted down. When it was taken in a dark room the reading given was 0 Lux. In a brightly lit room the light

level was found to be around 100 Lux to 300 Lux. Considering these readings, we decided to take the reference as 500 Lux to avoid interference caused by backlight conditions. The Text View box displaying the light level was also removed in the final version of the application.

4.2 Key Findings

While carrying out the tests on the application, we noticed that the shorter the pulse width of the light pulses, the more errors we were encountering in the final message displayed on the receiver. We later realized it was due to the receiver light sensor not being able to distinguish the gaps between the light pulses since the gap times were too small. Increasing the pulse width was enough to rectify this error. So, in order to improve the accuracy during the data transfer, we had to trade of the transmission and reception speed.

While carrying out the tests for determining the best pulse width, it was also observed from the timers that the durations of the light pulses corresponding to that of dots and dashes, and the gaps between them were not exactly equal to their expected pulse widths but instead had slight variations. This was due to the time that was lost between the data being transmitted and then received by the receiver. The time lapse between transmission and reception, however, was not always consistent. As a result, we opted for programming the receiver to use a range instead of exact pulse widths to detect and decode the dots, dashes, and the gaps between them.

For example, when we sent data messages at a pulse width of 200ms, it was observed that the timers showed the durations of the dots to be 250ms instead of the expected 200ms. Discrepancies in the dash pulse widths were also noticed. These variations ended up causing errors in the reception of data. Thus the dot pulse width was set at a range of 50ms to 500msfor detection in the receiver, and any pulse of light longer than 500ms was considered as a dash.

The ambient light was also found to be an issue during the reception of data. With reference level at 0 Lux in the receiver, the backlight was seen to cause errors during the data transfer process because the receiver light sensor would detect the light to be ON due to the surrounding light

even when it was not. Consequently, after taking ambient light intensity readings from different scenarios, we took the reference as 500 Lux which we found to best suit our case.

During our initial tests, it was noted that including the pulse width toggle bar in the transmitter part of the application helped us avoid any further modifications in the transmitter. We did not encounter any major problems in transmitting the data. All adjustments were carried out in the receiver. After the final modifications, the data could be transmitted and received accurately without encountering any errors.

5. CONCLUSION

5.1 Summary of findings

From the design process and the basic testing of the application, certain values were analyzed for an accurate transmission and reception of data. Since, we mostly emphasized on the accuracy of the data during transmission and reception, the factor “speed” was traded off. Initially certain values of pulse width like (50 ms, 100 ms, 150 ms and 200 ms) were chosen and we started with the lowest value and gradually increased the value and note down the changes. After monitoring the values and their corresponding changes, 200 ms was chosen to be the right pulse width for the data accuracy. Then, while using the ON OFF keying detection method, in the receiver side we faced the problem of different backlight levels. Thus, after experimenting in different backlight levels, 500 Lux was considered the brightest backlight scenario and hence the reference level. The detected transmitted light pulses were thus stored in the form of Morse code dot and dash and light pulses ON time and OFF time were calculated within the application. Using the pulse time we differentiate between a dash and a dot and the gap time was used to recognize whether a word, letter or sentences have been passed. Thus, we calculated that the pulse width for a dot is 200ms , for a dash is 600ms and the gap between letters was 200 ms, between two letters was 600ms and that of between two words was 1400ms. Moreover, during the transmission we need to make sure that the transmission can be ended when needed. So we took a help of a button which would be pressed when an incorrect data is being sent by the user. The button being pressed would in turn switch on a flag within the application which would indicate that the app needs to immediately break the loop and get compiled.

5.2 Future Works

In this project we were mainly concerned with the accuracy of the data during transmission and reception. As a result both speed and time were traded off. However, more work could be done in order to compensate for these. One of the possibilities can be using different encoding techniques like ASCII code, Huffman code. Here, we basically used Morse Code since it is

universally known and accepted however in the future slight modifications like creating some modified convention rules as changing the ratio between dot and dash, increasing the time gap between a letter, or between two letters and words can enable the pulse width to come to 100 ms consequently reducing the time.

We used a fixed reference level. But in the future if a variable light level reference is used, it will make reception more dynamic. A toggle bar can be used to set the light level according to the user's surroundings. This way we can have separate reference light levels in different ambient light situations. This will make reception of data more dynamic and accurate. Also, this will let the user choose the detection level and just in case if there are light conditions that user could not test the application for, it will take those into considerations and improve the application further. This also allows user of the application to have significant control over the transmission and reception of the data.

REFERENCES

[1] Visible-Light-Communications (n.d). Retrieved from http://en.wikipedia.org/wiki

[2] Visible-Light-Communications (n.d). Retrieved from www.bemi.org

[3] How Is Technology Used to Help Communications (n.d) Retrieved from www.Techwalla.com.html

[4] Information-Technology (n.d). Retrieved from www.idc-online.com/technical-references

[5] Visible-Light-Communications (n.d). Retrieved from www.technologies.cttc.es/m2m

[6] Activity. (n.d). Retrieved from www.fireflywirelessnetworks.com
Activity.(n.d.). Retrieved fromhttps://developer.android.com/reference/android/app/Activity.html
Sensors Overview.(n.d.) Retrieved from https://developer.android.com/guide/topics/sensors/sensors_overview.html
Environment Sensors.(n.d.). Retrieved from https://developer.android.com/guide/topics/sensors/sensors_environment.html

Application Fundamentals.(n.d.) Retrieved from https://developer.android.com/guide/components/fundamentals.html?hl=nn

[7] State diagram for an Android Activity Lifecycle. n.d. accessed 15 November 2016 <https://developer.android.com/images/activity_lifecycle.png>.

[8] Rhey T. Snodgrass & Victor F. Camp, 1922 Chart of the Morse code letters and numerals. Accessed

20 November 2016, <https://commons.wikimedia.org/wiki/File:International_Morse_Code.svg

[9] Hesselmann, T., Henze, N., & Boll, S. (2010). Flashlight. *ACM International Conference on Interactive Tabletops and Surfaces - ITS '10*. doi:10.1145/1936652.1936679

[10] Park, K.-S., Kim, S.-H., & Kang, Y. (2014).Development of transceiver using flashlight and camera in VLWC. *u- and e- Service, Science and Technology*. doi:10.14257/astl.2014.77.06

[11] Shirazi, A. S., Winkler, C., & Schmidt, A. (2009).Flashlight interaction. *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services - MobileHCI '09*. doi:10.1145/1613858.1613965

[12] "Ubiquitous Computing". *En.wikipedia.org*. N.p., 2016.Web. 3 Dec. 2016.

[13] Danakis, C., Afgani, M., Povey, G., Underwood, I., & Haas, H. (2012).Using a CMOS camera sensor for visible light communication. 2012 IEEE Globecom Workshops.doi:10.1109/glocomw.2012.6477759

[14] Corbellini, G., Aksit, K., Schmid, S., Mangold, S., & Gross, T. (2014).Connecting networks of toys and smart phones with visible light communication. IEEE Communications Magazine, 52(7), 72-78. doi:10.1109/mcom.2014.6852086

[15]McAnlis, C., &Haecky, A. (2016). *Understanding compression: Data compression for modern developers*. Sebastopol, CA: O'Reilly. Retrieved from https://www.safaribooksonline.com/library/view/understanding-compression/9781491961520/ch04.html