Wireless control of LED display system using Bluetooth and Android

Sidharta Prahladsingh
Dennis Rutten

Bachelor of Science Thesis

TU Delft Delft University of Technology

Delft Centre for Computational Science and Engineering

Wireless control of LED display system using Bluetooth and Android

Bachelor of Science Thesis

For the degree of Bachelor of Science in Electrical Engineering at Delft University of Technology

Sidharta Prahladsingh

Dennis Rutten

July 11, 2014

Faculty of Electrical Engineering, Mathematics and Computer Science (EWI) · Delft University of Technology

Table of Contents

Preface

The authors, Sidharta Prahladsingh and Dennis Rutten, have written this document as part their thesis for their BAP project (Bachelor Afstudeer Project, Bachelor Graduation Project) "Wireless control of LED lighting system using Bluetooth and Android" for the degree of Bachelor of Science in Electrical Engineering at Delft University of Technology. The idea for the project came from a Graduate Student, Manjunath R. V. Ramachandrappa Venkatesh, under Prof.Dr. Q.C. Zhang.

Acknowledgements

Our thanks goes to Dr. J. Wei, Prof.Dr. G.Q. Zhang and Manjunath R. V. Ramachandrappa Venkatesh for their coordination and continued support during the development of this project, Mr. Ytai Ben-Tsvi, creator of the IOIO-OTG circuit board, for his support with the IOIO-OTG and Android programming, Mr. Rudy Tjin-Kon-Koen for his support and implementation ideas for the Android app, TU Delft for providing us a labroom and resources to work with and finally again Manjunath R. V. Ramachandrappa Venkatesh for the idea for this project.

Delft, University of Technology July 11, 2014

Chapter 1

Introduction

More and more information is available on the internet. However, there are not many cheap and easy ways to display information to large groups of people in public. In our project, we will design a low cost and low energy solution to this problem using a LED-Matrix made from flexible LED-strips as seen in Figure 1-1. In this thesis we will defend and explain how we implemented our contribution to this project.

Figure 1-1: Flexible LED-Strip with 16 RGB LEDs

In this project we are supposed to develop a system that allows an Android smartphone to control a LED-Matrix made from aforementioned flexible LED-strips. Our goal for this project is to at least support a 32 by 32 display and be able to control that using an Android app. When our goal is reached we are supposed to add extra features and functionality to the system to further enhance the experience with the system.

Chapter 2

Summary

In this chapter we will in short show what the full system design will be for the full project.

2-1 System Design Overview

As can be seen in Figure 2-1 the system contains of four major blocks.

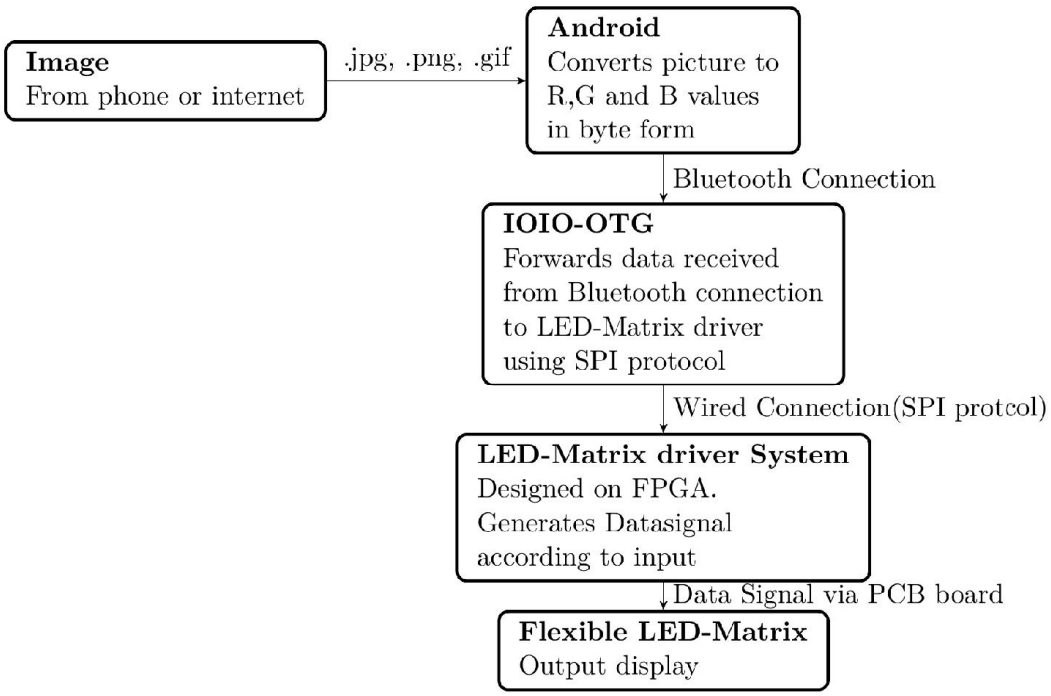

Figure 2-1: Block diagram for basic full system design

First you have the Android app, which retrieves a photo file from your phone or the internet and sends it to the IOIO-OTG I/O board via Bluetooth. The IOIO-OTG forwards the received data to an FPGA using SPI protocol. The FPGA will contain the design for a driver that controls a flexible LED-Matrix. This FPGA will process the received data and sends out a data signal to the LED-Matrix. The LED-Matrix itself include drivers for red, green and blue LEDs separately, which the aforementioned data signal coming from the FPGA board should give input for.

The LED-Matrix in our case is special when it comes to its cost but also in its application possibilities. The LED-strips are flexible, which allows a variety of applications on curved surfaces.

Chapter 3

Communication Protocols

In this chapter we will investigate some wired and wireless protocols that are available for our system.

3-1 Wireless Protocols

There are quite some ways to deliver information wirelessly: infrared, Wi-Fi, Bluetooth, etc. however the most applicable two protocols for our system would be Wi-Fi and Bluetooth. In the next few parts we are going to list some differences between the two protocols.

3-1-1 Bluetooth

Figure 3-1: Bluetooth logo

Bluetooth technology exchanges data over short distances using radio transmissions. Bluetooth technology operates in the unlicensed industrial, scientific and medical (ISM) band at 2.4 to 2.485 GHz, using a spread spectrum, frequency hopping, full-duplex signal at a nominal rate of 1600 hops/sec.[1]

Range is application specific and although a minimum range is mandated by the Core Specification, there is not a limit and manufacturers can tune their implementation to support the use case they are enabling.

Range may vary depending on class of radio used in an implementation[2]:

- Class 3 radios – have a range of up to 1 meter or 3 feet
- Class 2 radios – most commonly found in mobile devices – have a range of 10 meters or 33 feet
- Class 1 radios – used primarily in industrial use cases – have a range of 100 meters or 300 feet

3-1-2 Wifi

Figure 3-2: Wi-Fi logo

Wi-Fi is the name of a popular wireless networking technology that uses radio waves to provide wireless high-speed Internet and network connections. A common misconception is that the term Wi-Fi is short for "wireless fidelity," however this is not the case. Wi-Fi is simply a trademarked phrase that means IEEE 802.11x.

Wi-Fi works with no physical wired connection between sender and receiver by using radio frequency (RF) technology, a frequency within the electromagnetic spectrum associated with radio wave propagation. When an RF current is supplied to an antenna, an electromagnetic field is created that then is able to propagate through space. The cornerstone of any wireless network is an access point (AP). The primary job of an access point is to broadcast a wireless signal that computers can detect and "tune" into. [3]

While the official speeds of 802.11b, 802.11g, and 802.11n networks are 11, 54, and 270 megabits per second (Mbps) respectively, these figures represent a scenario that's simply not attainable in the real world. As a general rule, you should assume that in a best-case scenario you'll get roughly one-third of the advertised performance.

It's also worth noting that a wireless network is by definition a shared network, so the more computers you have connected to a wireless access point the less data each will be able to send and receive. Just as a wireless network's speed can vary greatly, so too can the range. [4]

3-1-3 Comparison

When it comes to cost, Bluetooth definitely is very attractive here. It is much cheaper to implement

Bandwidth seems to be on the low side with Bluetooth (approx. 800Kbps, depends on manufacturer), while Wi-Fi has more bandwidth (approx. 11Mbps, 54Mbps, and 270Mbps).

In our system bandwidth it is not much of a problem since we do not require much bandwidth when we do one on one connection between Android device and LED-Matrix. However if for applications where we connect more devices to the LED-Matrix, Wi-Fi would be better.

In our prototype system we do not need much higher bit-rates than 1Mbps. So Bluetooth bit-rates (2.1Mbps) would be sufficient compared to Wi-Fi (600Mbps).

Bluetooth does not require an AP or access point like a Wi-Fi connection does. Also Bluetooth range is much shorter (5-30meters or more, depends on manufacturer) and Wi-Fi has a much larger range (32-95meters, depends on manufacturer, location or communication frequency). For our prototype system we do not need much range unless you want to incorporate the system as a display for information from a source far away.

Power consumption for Bluetooth is low, which is quite nice if you want to keep your system green, but depending on the scale of the system, the power consumption wouldn't matter much compared to use of Wi-Fi.

Wi-Fi can be much more complex than Bluetooth to configure in both hardware and software, so again it would depend on the scale of the system. Only if the scale needs to be large, would the configuring of Wi-Fi be worth it.[5]

3-2 Wired Protocols

The IOIO-OTG supports a few standard communication protocols like UART, I^2C and SPI. We are now going to look at each of the possible communication protocols that the IOIO-OTG supports and list some situations where these protocols work best.

3-2-1 SPI

Serial Peripheral Interface (SPI) is a common interface used to communicate between integrated circuits. A single SPI bus enables a single master device to communicate with one or more slave devices at typical rates of up to tens of Mbit/sec. SPI is a full-duplex interface, meaning that transmission and reception can are taking place concurrently. SPI requires 3 wires shared by the master and all slaves and an additional wire between the master and each slave.

The master transmits pulses on the CLK line, to which all slaves listen. On each CLK pulse, the master writes one bit to the MOSI (master-out-slave-in) line, and reads one bit from the MISO (master-in-slave-out) line. Only a single slave may enabled at any given time, preventing the possibility of concurrent write to the MISO line. The enabled slave is selected by a SS (slave-select) pin, which the master controls. The master will never enable more than one slave at a time. A slave that is not selected using its SS pin must never affect the MISO line and also typically ignores the MOSI line. While normally the MISO is supposed to be connected, the IOIO-OTG can just pull up or pull down the MISO pins, where it allows a send-only situations.

The following illustration shows a typical SPI transaction, in which the master sends 5 bytes (0x01, 0x02, 0x03, 0x04, 0x05) and receives 4 bytes (0x0A, 0x0B, 0x0C, 0x0D) with a lag of 3. The total transaction length is thus 7 bytes. Time goes from left to write:[6]

Master	0x01	0x02	0x03	0x04	0x05	0xFF	0xFF
Slave	0xFF	0xFF	0xFF	0x0A	0x0B	0x0C	0x0D

In Figure 3-5 we see a block figure of an single Slave system with an example signal underneath. In this we see how the data signal (MOSI) will be send out with a sample CLK (SCLK) by

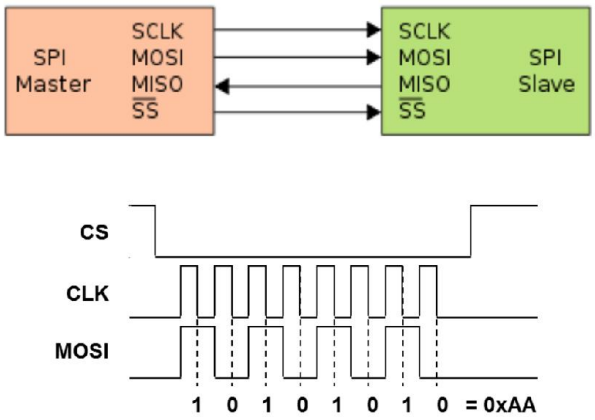

Figure 3-3: SPI signal example for 1 byte(8bits) transfer

a SPI Master. This SCLK would be used by the Slave to sample the MOSI signal to receive the right byte value. All this would be done as long as CS/SS is 0 for every package.

The SPI bus can operate with a single master device and with one or more slave devices.

3-2-2 I^2C

It is an interface which enables half-duplex serial communication between multiple devices, sharing the same bus, with as little as 2 wires (hence the name). Each of the devices can be either master or slave, where masters are the only ones that can initiate data transactions. Each slave device has a unique address on the bus, which is used to select it as the target of a transaction.

The two wires used are called SDA (data) and SCL (clock). The SCL line is normally controlled by the master, but in certain situations (clock stretching) may be pulled low by a slave device. The SDA line is used for data transfer in both directions. The protocol dictates who has access to the SDA line at any given time. Certain combinations of SDA/SCL states are used as special signals between the master and slave, such as beginning and ending a transaction.

A TWI transaction is comprised of one or more write steps or read steps. In a write step, the master sends data to a slave. In a read step, the slave sends data back to the master. Most commonly, TWI transactions will have either a write followed by a read, or only a single write / read.

The data rates 100KHz, 400KHz or 1MHz are supported. [7][8]

Let's look at a sample signal to get a sense of the timing done with I^2C. In Figure 3-4 we see an example system and sample signal. Here the following happens:

- Data Transfer is initiated with a START bit (S) signaled by SDA being pulled low while SCL stays high.
- SDA sets the 1st data bit level while keeping SCL low (during blue bar time.)
- The data is sampled (received) when SCL rises (green) for the first bit (B1).
- This process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high (B2, Bn).
- A STOP bit (P) is signaled when SDA is pulled high while SCL is high.

In order to avoid false marker detection, SDA is changed on the SCL falling edge and is sampled and captured on the rising edge of SCL.

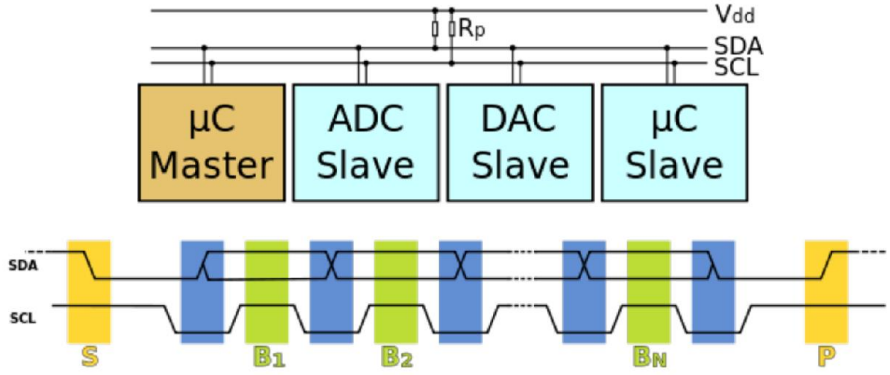

Figure 3-4: I²C Single master example system with 3 Slaves

The I²C bus is a multi-master bus which means any number of master nodes can be present. Additionally, master and slave roles may be changed between messages (after a STOP is sent).

3-2-3 UART

Universal Asynchronous Reception and Transmission (UART) is a very common, simple and useful serial communication interface. Its basis is a one-way communication channel, on which one end transmits and the other end receives on a single wire. Each byte is sent on the wire bit-by-bit, preceded by a start bit which is simply the bit '0' and followed by an optional parity bit (used for error correction, but commonly not used at all) and one or two stop bits (commonly only one), which are simply the bit '1'.

When the line is idle, it is HIGH. The rate at which bits are sent is called baud rate. Common baud rates are 9600, 19200, 38400, 115200, etc.

In the Table below you see a example for a signal send via UART. Here D stands for Data and the last Stop-bit is optional.

Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start	D. 0	D. 1	D. 2	D. 3	D. 4	D. 5	D. 6	D. 7	Stop	Stop

Transmitting and receiving UARTs must be set for the same bit speed, character length, parity, and stop bits for proper operation. [9][10]

3-2-4 Comparison

Let's compare these protocols, so we can later make a decision on which of the three is most suitable for our system. When it comes to bit-rates, the SPI protocols excel due to the wide choice of speeds ranging from 30Kbits/sec to 8Mbits/s. UART has this freedom too but the interrupts make it quite annoying to work with. I^2C however is either set to 100kbits/sec or 10kbits/sec, which does not really work for our prototype system.

UART is the easiest system to implement, with I^2C being the hardest. However, SPI shows to be more applicable to communication systems overall.

In some systems the amount of used pins for the communication system can be an issue. For this, SPI protocol uses the most pins (3 pins) in a single Master single slave setup, while UART needs 2 and I^2C needs 1.

3-3 Results, Discussion and Conclusion

We will now test and conclude our final decission on communication protocols.

3-3-1 Wired

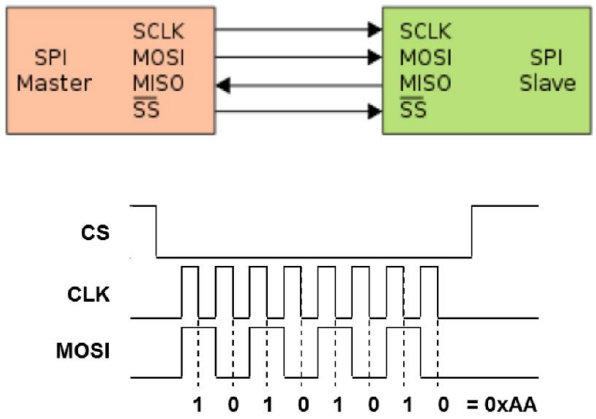

Figure 3-5: SPI signal example for 1 byte(8bits) transfer

Having seen our comparison in Section 3-2-4, we can now conclude which of the protocols is the most suitable for our system.

Our SPI system is accurately portrayed in Figure 3-5 except for the fact that we decided that we will be sampling on the rising edge (which was an option for the IOIO-OTG). To test if this was actually happening we send a similar alternating signal (0xAA valued constant

signal) to see if we got the right signals with an oscilloscope (alternating signals are of course easier to analyze with an oscilloscope) on all pins (CLK, SS/CS and MOSI). In Figure 3-6 we see a picture to the left and a signal on a oscilloscope to the right. The picture is created from a constant 0xAA value, which in theory should result in an alternating signal like in Figure 3-5. The resulting signal at the MOSI pin of the IOIO-OTG looks to be correct, since the frequency is 500kHz(half of CLK speed of 1MHz) and it's alternating.

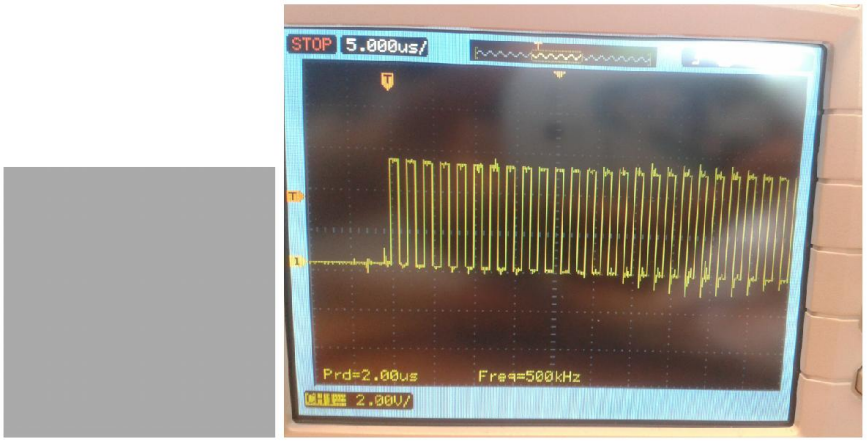

Figure 3-6: Left shows input picture (color with R,G and B = 0xAA) and right shows resulting signal at output at a bitrate of 1Mbits/sec

3-3-2 Wireless

When we choose a wireless protocol, it mostly depends on the scale of the system. When we need to include support for multiple users or for larger information travel distance then Wi-Fi is the better choice to implement. However, in our prototype system we are accounting for a one on one connection with short distance and low cost, so Bluetooth would definitely be more than sufficient.

We did some tests on some Bluetooth dongles that were supposedly compatible with the IOIO-OTG. However from the 3 dongles we ordered only 1 actually showed any results when trying to pair with android devices. Figure 3-7 shows a picture of the working dongle on which we will soon be looking at some test results from. To test the Bluetooth connection between the dongle used on the IOIO-OTG and an Android phone. To test this we plug the Bluetooth dongle up to a laptop running on Linux (Windows does not allow easy capturing of Bluetooth packages) and then look at the speed and the range we got at max. Before we actually start the test we kept a few things in mind:

- Most mobile phones have Bluetooth v2.0 systems with Class 2 radios, so the nominal bit-rate in theory would be about 3Mbits/s with a max range of 10m (30feet).
- The dongle in Figure 3-7 does say Class 1, but it is not sure if this is really the case since the dongle is really cheap and probably not of optimal quality. However, we know, like mobile phones, the phone has Bluetooth v2.0, which means that the nominal bit-rate is

Figure 3-7: Bluetooth dongle used in this Project

about 3Mbits/s and the nominal range (assuming the dongle actually works as a Class1 system) should be 100meters(300feet).

- Interferences can be a problem in all situations, which includes our testing environment, so the range and average rate might suffer because of this.

Keeping all above in mind, we expect a bit-rate of <3Mbits/s (<375MB/s) and a range of <10meters (<33feet).

After quite a few repetitions of our tests, our results were as follows:

- The max. bit-rate we got was 180kB/s which amounts to 1.440Mbits/sec
- The max. range we got without losing connection is about 5meters (16.40feet).

The equipment and resources we had for the tests were not very precise to test the decay of bit-rate against range. At about 4.5 meters we still got the max. bit-rate of about 1.416 Mbits/sec, so from that we know that the decay of the bit-rate should not be much of a problem in our system.

This shows that our predictions of the tests were actually not far off and now we can conclude that for wireless connection we will have a limit of 5meters range and 1.440Mbits/s bit-rate.

Chapter 4

Smartphones and Apps

In this chapter we are going to talk about our research of smartphones and its apps. We are finally going to motivate which

4-1 What are Smartphones and Apps?

Communication, entertainment, news-feed, information source, payment device, etc. are a few mentions of the many uses for smartphones and it is hard to find anyone not having one these days. Smartphones come with OS. Currently in the world the most notable two OSs found on smartphones are Android and iOS, developed by Google and Apple respectively.

Apps are being developed almost every day on every smartphone platform for many different purpose, whether for private or commercial use. These days, app development has been made really easy for anyone to get into, create content and spread it to the public.

The questions we are going to try to answer, are:

- What is de best OS platform to create apps for?
- How do we start creating apps for this platform?
- What functionality are we planning to have in the app?
- How are we planning to implement these functionalities?

4-2 Smartphone OSs

While it might seem obvious what smartphone OS we will choose to make our app in due to our project's title, it is always better to look at all options and be aware of the limitations

and advantages that we may face in our development process in one OS compared to the other. Earlier we already mentioned that the two most prominent OSs these days are Google's Android and Apple's iOS. Let's first look at each of these two and their positives and negatives.

4-2-1 Android

Figure 4-1: Google's Android Logo

Android is an operating system based on the Linux kernel with a user interface based on direct manipulation.[11] As of 2011 Android has the largest installed base of any mobile OS and as of 2013, its devices also sell more than Windows, iOS and Mac OS devices combined.[12] As of July 2013 the Google Play store, the Android app distribution site, has had over 1 million Android apps published, and over 50 billion apps downloaded.[13] In the third quarter of the year 2013, Android held 81,3% of the total Global Smartphone OS Market Share.[14] See Figure 4-2-1.

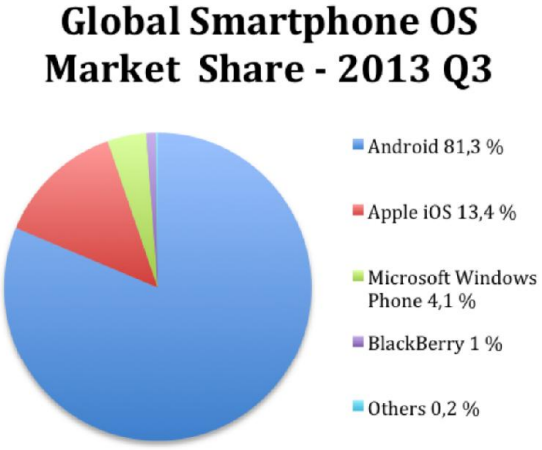

Figure 4-2: Global Smartphone OS Market Share Q3 2013

This OS's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software.[15] Android is popular with technology companies which require a ready-made, low-cost and customizable operating system for high-tech devices.

Android has a broad selection of first-and third-party apps, which can be acquired by users through app stores such as Google Play or by installing APK-files downloaded from third-party sites. These apps or applications, that extend the functionality of devices, are developed primarily in the Java object oriented programming language using the Android software development kit (ADK). The SDK is a spin-off of the Java development kit (JDK), made more compatible for mobile devices and its uses.

4-2-2 iOS

Figure 4-3: Apple's iOS Logo

iOS (previously iPhone OS) is a mobile OS developed by Apple Inc. and distributed exclusively for Apple hardware. It is the operating system that powers iPhone, iPad, iPod Touch, and Apple TV. Major versions of iOS are released annually. By late 2011, iOS accounted for 60% of the market share for smartphones and tablet computers. By Q3 2013, iOS accounted for 13,4% of the smartphone OS market.[16][14]See Figure 4-2-1. iOS, unlike Android OS, is a closed system. iOS devices are all according to a specific hardware design making app creation easier when keeping hardware compatibility in mind.

The iOS SDK was released on March 6, 2008, and allows developers to make applications for the iPhone and iPod Touch, as well as test them in an "iPhone simulator". However, loading an application onto the devices is only possible after paying an iPhone Developer Program fee. Objective-C is a thin layer on top of C, and moreover is a strict superset of C; it is possible to compile any C program with an Objective-C compiler, and to freely include C code within an Objective-C class. This makes functionality very broad and allows for many implementations.

4-3 What is the best OS platform to create apps for?

Android OS can't support Objective-C and iOS can't support Java, so we have to make a choice between the two systems.

When it comes to programming, in our case Java is the most accessible. We have been exposed to Java during the Bachelor Program of Electrical Engineering, making it easier to develop the app in Java. While Java has some restricted functionality, its ability to include C programs makes it as well as Objective-C for programming.

iOS is much easier to write apps in due to the hardware being fixed and not changing for a long time. For this project we want to be able to control various sensors and internal devices in the phone and Android provides this ability better. Also the strict policies from Apple might be hindering in app development process, not to mention the licensing cost.

These days and specifically since last year, the amount of Android users has been steadily towering over the amount of iOS users.

When we want to create an app, we also want to be sure that our app is accessible to as much users as possible and we want to have freedom to test and distribute as we want, so Android is the best choice.

4-4 App creation on Android

Since we have now decided that we are going to develop our app for Android platform, let us consider the process. We first need to select an IDE. We are selecting Eclipse as our IDE for various reasons. Eclipse is a very powerful program that makes developing programs easy and efficient due to its ability to download plug-ins and SDKs from the internet and its powerful compiler. It shouldn't matter which IDE is used, but stability is definitely guaranteed with Eclipse.

Furthermore, since we are only trained in basic level of Java/Object Oriented coding and first time building an Android app, we used sources listed in the bibliography to help us.

Chapter 5

IOIO-OTG

In this chapter we will look at the IOIO-OTG and its functions applicable to our project and then finally show how we finally apply the IOIO-OTG to our system.

5-1 What is the IOIO-OTG?

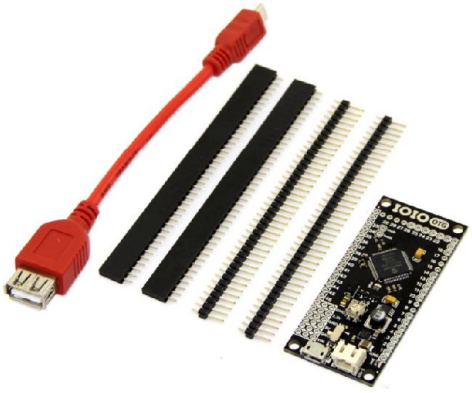

Figure 5-1: IOIO-OTG circuit board

IOIO-OTG (pronounced as "yoyo on the go") is a printed circuit board created by Ytai Ben-Tsvi. The board is small and light and is relatively cheap, but is not supposed to replace for instance a board like Arduino. It is only meant to provide I/O function where existing systems lack it and add in either Bluetooth connection or USB communication with Android devices. The IOIO-OTG is all open-source, software, firmware and hardware.

The IOIO-OTG board has the following main features:[17]

- USB-OTG dual-role (host, device).

- Input voltage: 5V-15V, from external source or through USB (when connected to a computer).
- Output voltage: 5V, up to 3A (!), 3.3V, up to 500mA.
- 46 I/O pins (digital I/O), built-in pull-ups / pull-downs / open-drain on all pins.
- 16 Analog inputs.
- 9 PWM (for driving servos, DC motors, dimming LEDs, etc.).
- 4 UART.
- 3 TWI (I2C, SMBUS).
- 3 SPI.
- 6 Pulse Input (precise pulse-width / frequency measurement).
- USB current limiting when acting as USB host (useful in Android mode).
- Switch for forcing host mode (for using non-standard USB cables, which are more common than the standard ones...)
- On-board LED under user control.

5-2 What is the purpose of the IOIO-OTG in this project?

We were provided the IOIO-OTG for our project to use it to drive the LED-Matrix. However, we quickly decided that it would be impossible for a four-man team to work on one Android code, so we decided to design our driver on an FPGA board and only use the IOIO-OTG as a communication device between Android devices and the FPGA board. As said before, the IOIO-OTG includes handy libraries for Android and also has included functionality to communicate via Bluetooth. Both of these features would be far too complex and time consuming to finish designing within the project time period, therefor using the IOIO-OTG in our prototype as proof of concept should suffice.

5-3 Communication Setup

Our communication setup is displayed in Figure 5-2. What the Android does to provide the RGB values can be found at Chapter 6. For now we will assume that the IOIO-OTG will receive 48 bytes of data, which will be forwarded to the FPGA via SPI protocol with a rate of 1Mbits/sec. As mentioned before the IOIO is capable only of Bluetooth connection as wireless connection and includes a variety of sending protocols for connections with its I/O pins. However this design is just a proof of concept so we need to consider for instance Wi-Fi as an option and motivate why we choose SPI protocol to send information to the FPGA.

In Chapter 3 we discussed a few of the communication protocols available protocols and motivated which why Bluetooth and SPI protocol is the right choice for our system. With these choices in mind we can now design our communication system between Android devices and the FPGA/LED-Matrix driver.

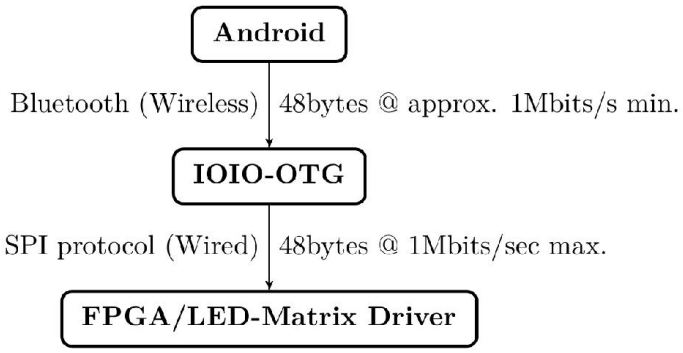

Figure 5-2: Communication Setup

5-4 Implementation

In Section 6-2-5 we show how we order the IOIO to forward information. Only implementation we have to do for the IOIO is to connect a Bluetooth dongle discussed in Chapter 3 and to connect the in the android assigned MOSI, CLK and an extra pin to signify when an image is being sent.

5-5 Results Discussion and Conclusion

The IOIO-OTG is very handy for Bluetooth connections with Android and providing I/O pins to the FPGA, however implementing Wi-Fi would be a bit more tricky. In Section 3-3 we see some results from the IOIO-OTG producing an SPI signal successfully. In Section 6-4, we see that we get positive results from the implemented IOIO-OTG and the coding, showing that this prototype is actually proven to be a success.

Chapter 6

Android Application

In this chapter we are going to explain the functionalities we want to implement into our app and how we implement these.

6-1 App Functionalities

Since we as Electrical Engineers can't be expected to instantly be able to develop complex apps and since this system is supposed to function as prototype or proof of concept, we are going to start with a basic design for the app. Earlier in the Chapter Project Goal we saw that we need to be able take a picture and send the R, G and B values of the picture per 16 LED per package to the IOIO-OTG via Bluetooth to be further sent to the FPGA(LED-Matrix Driver) as seen in Figure...This basic design will need to comply with the following requirements to achieve the goals mentioned above:

- It is safe in this level of design to assume the prototype LED-Matrix will be 32x32 pixels big, but later we want to be able to expand. So accounting for this during development is needed.
- Take an existing picture from the phone and resize and crop it to accommodate for the resolution of our LED-Matrix.
- This picture needs to be analyzed per pixel for the RGB value of the picture.
- The RGB value needs to be split in an R, G and B value in byte form.
- All collected values in byte form need to be sent to the IOIO-OTG via Bluetooth with instructions to forward the information to the FPGA via SPI protocol.
- Spi protocol on the IOIO-OTG only supports 64 bytes of data, so all collected info need to be prepared for communication.

- The moment this basic level of functionality is achieved in the app, we can expand the app further to include more functions and enhance the visual aesthetic of the app.

Now that we have a clear picture of our first milestone we can now list some functions we can try to implement after basic level of functionality is achieved:

- Get pictures from the internet.
- Scroll picture to change where the picture is cropped to.
- Splash screen and visual app design for a more representable design.
- Sending info obtained from Animations.
- Convert input text to picture.
- Include options like screen size or background color.

6-2 Functionality Implementation

Let's split the basic level functionalities we defined earlier:

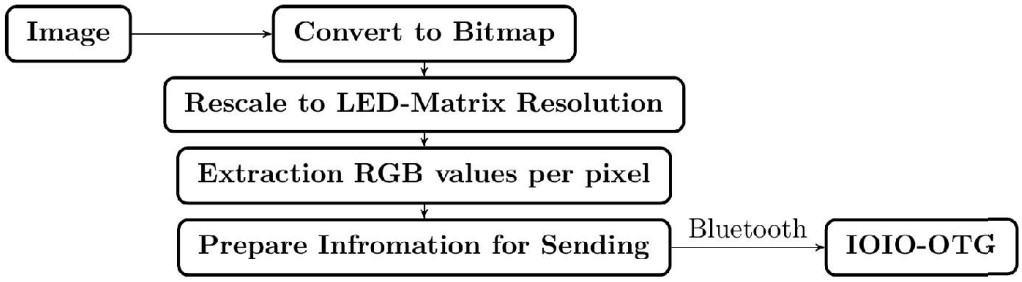

Figure 6-1: Visualization of basic level functionality of Android app

Above we can see the basic level functionality we defined earlier visualized. The 4 blocks in the middle represent the methods used in the app. We will now explain how each of these blocks will be implemented.

6-2-1 Grabbing a Picture from Phone

What we want is to make a button that brings you to a dialog to browse your phone for a Picture. Android provides preset action handlers. In this case, we can create a `Button` that will start an so called `Intent` to pick an image file. We see the implementation of this in the Appendix A-1-3.

6-2-2 Preparing Picture for Analyses

After choosing the picture, we take the chosen picture and converts it to a `Bitmap` class object. After conversion we want to resize the picture to the size of the LED-Matrix, preferably without stretching.

The reason we want to convert the picture to a `Bitmap` is because we can later use the method `Bitmap.getPixels(int[] pixels, int offset, int stride, int x, int y, int width, int height)` to retrieve the RGB data of each pixel in the picture, which we will discuss later.

To convert the picture we need the method `BitmapFactory.decodeFile(String pathName)`. This method needs the `pathName` which means we have to create a string that precisely show where the selected picture resides.

Since every phone compared to each other can have different storage names and since a phone can have multiple storages, we need to make sure the correct path is always "formulated" for it to work with the method `BitmapFactory.decodeFile(String pathName)`. So for this we use a `Cursor` to point in the right folder and then create a correct `String` notation of the path. This can be seen implemented in the Appendix A-1-4. From here we see from that the picture is then rescaled with `RGBdrive.rescale(Bitmap bm,int d_x, int d_y)` and then processed for sending with `RGBdrive.rgbDrive(Bitmap bm)`. Here `RGBdrive` is the `class` shown in the Appendix A-1-5 where we declared most methods used for `Bitmap` processing.

6-2-3 Rescaling `Bitmaps`

Our LED-Matrix is quite variable in resolution and to comply with this we will need to resize and crop our selected picture. The problem with this is that there are simple methods available to resize `Bitmaps` to a specific resolution however this does not take in account the ratio between height and width, resulting in a stretch or squished image.

What we do for that, is seen implemented in Appendix A-1-5. We check this by determining if either the height or width is smaller and calculating the scaling ratio between the largest of the two and the equivalent of the target picture and then finally using that ratio to rescale the picture. This way we fill the LED-Matrix with as much of the picture as we can.

See Figure 6-2 for a visual representation of what we are trying to implement. In this case we ensure that in most cases we will have filled the LED-Matrix with a decent amount of the picture. In some cases the picture will not completely fill the LED-Matrix, but we chose to allow this since in these cases the empty bits will appear black and we will have the full picture in the screen. Also note that during this rescaling design we keep the focus in the center.

Now that we have selected a suitable ratio, we can actually scale the image by preparing a Rescaling Matrix using the calculated rescale ratio with `Matrix.setScale(float xscale, float yscale)` and applying this to the image with both `xscale` and `yscale` set to our selected scaling ratio. Like seen in the Figure 6-2 we, in this manner, preserve our ratio of the image and avoid squishing or stretching.

After preparation of the Rescaling Matrix we send all info to the next method where we actually finish resizing the picture. We won't focus too long on what goes on in this method.

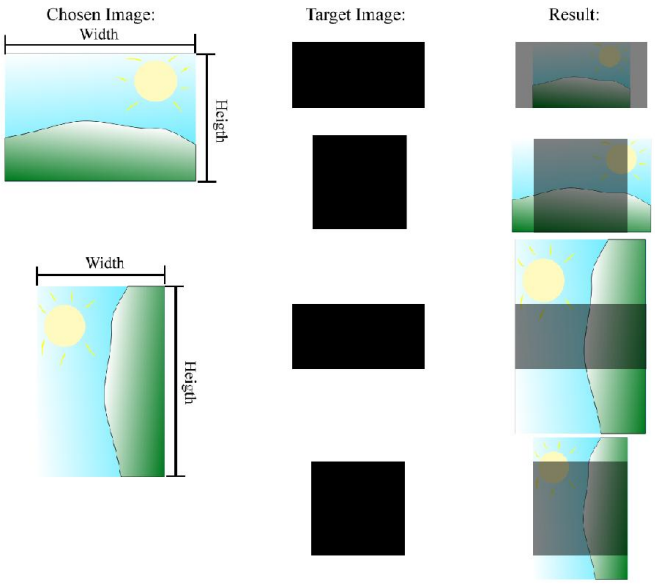

Figure 6-2: Vizualization of rescaling process

In short, we resize the original picture and then draw a rectangle with the `Rect` class to select a part of the resized picture to which we want to crop to. To achieve this we have to use some algorithms to every time calculate the needed info to finalize the resized picture as seen in the Appendix A-1-5. This part of the code also accomodates for the scroll feature we mentioned earlier, implemented as an extra feature.

After the result is achieved, the resulting `Bitmap` gets send to the next part of the process, which is the analyses of the RGB values per pixel.

6-2-4 Analyzing `Bitmap`'s Pixel RGB Values

What we want to do is dump all RGB values of a picture per pixel and split the R, G and B values and store these values in a `byte[]` "byte array" in such a way that we can send this info in an easy way to the IOIO-OTG.

RGB values for a pixel contain the information representing the level each color is apt to display the color seen in this pixel. Each color have 256 levels of shades. To represent these level we can use bytes containing 8 bits worth of data. So after getting the right R, G and B value we convert this value to byte form.

We are going to use the method `Bitmap.getPixels(int[] pixels, int offset, int stride, int x, int y, int width, int height)` to get all RGB values of the rescaled `Bitmap` per pixel, where `pixels` is the array to receive the bitmap's colors in. We need to consider a few requirements and notes before we get to work:

- The above method reads the pixels from left to right and from top to bottom. We need to check if this complies with the LED-Matrix.

- Each value in `pixels[]` will represent the RGB value of a pixel, numbered according to above mentioned reading rule.
- The LED-Matrix reads in the color values G, R and B in that order.
- For their implementation, our teammates in charge of designing the driver for the LED-Matrix using the FPGA, requested the information to be send for each LED-strip separate. Along with this all color codes need to be sorted per color per LED-strip (16 pixels). This means information for each LED-strip needs to be bundled and send in the following way: 16 Green values in bytes, 16 Red values in bytes and finally 16 Blue values in bytes for 1 LED-strip of 16 LEDs and repeat for next strip.

To comply with these requirements and notes, we need to create a solid algorithm to prepare the RGB data. The basic idea we are going to use here is that we are going to convert the picture in a large `byte[]` array that has a size of the picture's total amount of pixels times 3. This `byte[]` array will be of the following form:

Array entry	0-15	16-31	32-47	47-62	...
Color Values	16∗G	16∗R	16∗B	16∗G	...

So the algorithm needs to fill in the array at the correct location with the correct info.

Earlier we mentioned de array `pixels[]` to contain RGB values for 1 pixel per entry. To split this we just need to use a bit-wise shifting of the RGB value and then adding that to the hex value `0xff` to get R, G and B values seperate. This is implemented as seen in Appendix A-1-5. Using the help of strip numbering and pixel numbering we can easily fill in the right value for the colors of a pixel in the correct location in the `byte[]`.

6-2-5 Sending Information to IOIO-OTG

Finally we have to send our `byte[]` array of RGB values to the IOIO-OTG. Earlier in Chapter 5 IOIO-OTG Section 5-3, we saw that we selected to send information from the IOIO-OTG board to the FPGA using SPI protocol. SPI protocol can send 64 bytes of data per time so we need to control our data output to accommodate for this.

The IOIO-OTG came with a handy set of libraries containing pre-made methods and classes for controlling the IOIO-OTG with an Android phone. Due to the included preset libraries for the IOIO-OTG the Android app will automatically constantly be trying to connect with the IOIO-OTG (via Bluetooth or USB) and when connected will send information as requested as long as allowed. In the Appendix A-1-6 we see a class called `Looper` which initiallizes pins, opens them and finally in a constant loop tries to send data to the IOIO-OTG according to what is written in the `loop()` method. In the `loop()` method we just have to order the IOIO-OTG to send a `byte[]` array through the earlier opened SPI protocol connection to the FPGA. Also a pin sends a logic "1" for as long as we are sending an image and a logic "0" when idle.

For easy handling on FPGA side, we decided to split the data to 16 LEDs * 3 colors = 48bytes of data per package. Removing or adding 1 strip amounts to 48bytes of data, so having packages be consistently 48bytes big, globalizes our code for future expansions of the LED-Matrix. In the code at Appendix A-1-6 we see our code implemented.

6-3 Extra Implementations

Note: *Since our current coding for the app is much too long to include in our thesis we will only include the basic code needed to achieve basic functionality. The code including the extras is about twice as long.*

Since we just wanted to build upon our existing code, we just implemented a few classes that take a GIF animation and split it into an array of `Bitmaps`, which are being send to the LED-Matrix through the same way pictures are being send one after another.

Furthermore, we have added the ability to send a picture or GIF animation from the the web to the Led-Matrix, signifying the internet capability we have.

Another feature we added is to convert an input text to a picture in the form of `Bitmap` and send that to the LED-Matrix. This we did to show we can show custom info.

The last extra feature is the option to scroll last sent picture using its preview via touch commands. With this feature we can let the user select the correct position we want to use of the cropped picture. This shows that we can have some interactivity in our app.

An implementation, that at the moment of writing we are hoping to have ready at the demonstration of our system, is a feature that compliments the LED-Matrix in its flexible state. For instance, when the LED-Matrix is applied in a cilindrical display facing outwards, we could let an image or message circle around the display. This ofcourse will signify the possiblities with the LED-Matrix applied on a curved surface.

The last extra feature is the option to scroll last sent picture using its preview via touch commands. With this feature we can let the user select the correct position we want to use of the cropped picture.

6-4 Results, Discussion and Conclusion

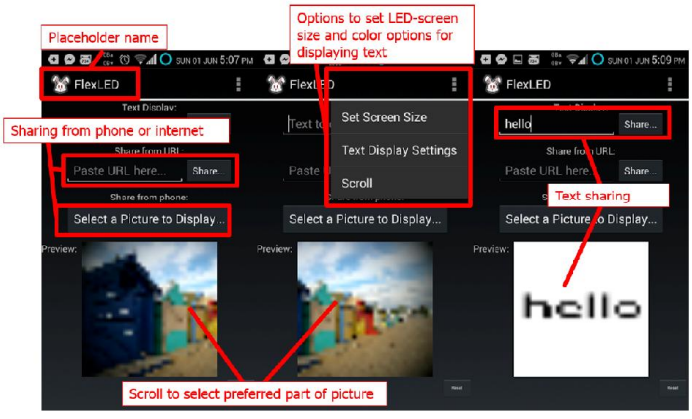

Figure 6-3: Resulting Androidapp

Our resulting app is shown at Figure 6-3. In this picture we can see a few of our features in action. The app is tested and works perfectly, although with some bugs for now. The

IOIO-OTG seems to be sending the right information every time we send a picture as seen in the Figure 6-4 so we can conclude that are current implementation must therefore be working correctly. In Figure 6-5 we see an example of a picture being resized and cropped to the required resolution and this seems to be going correctly using the algoritm we discussed in Subsection 6-2-3, proving that the used algoritm is good for our system's needs.

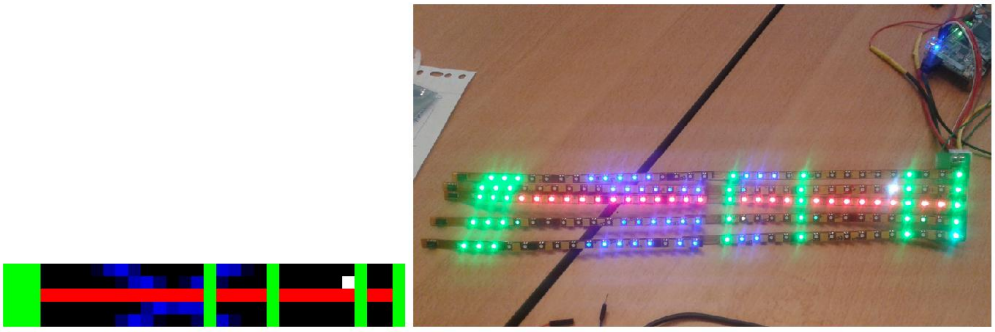

Figure 6-4: Left shows input picture and right shows resulting output at the LED-Matrix

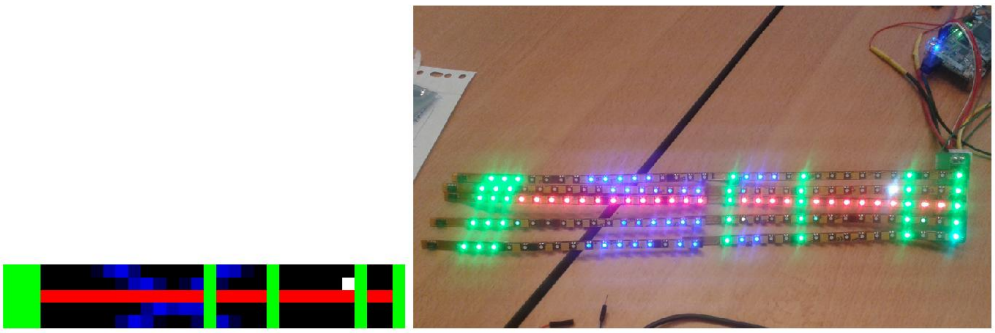

Figure 6-5: Left shows input picture and right shows resulting output at the LED-Matrix

There is however a few observation to made concerning stability and optimization. During our development of the app we have noticed a few key points that needs to be kept in mind:

- We need to keep the memory of the Android in mind. The amount of internal RAM memory a device can have is limited per device. To accommodate for this we need to keep our app memory efficient. This means that we, for instance, need to keep path names in our memory instead of full `Bitmaps` itself.
- The code is written in such a way that screen size modification at the LED-Matrix side will not impact the programming at all. Since we added an option to change the resolution in-app, the scalability of the LED-Matrix is very flexible when it comes to the Android app. However, this is still limited to the Android device itself since higher resolution can mean that more internal RAM is needed.
- Developing apps for Android devices is not limited to smartphones, as mentioned in Chapter 4. This means that the app is definitely usable on for instance Android Tablets as well. Any device running Android 2.0 and up and including sufficient amount of

RAM to handle the image processing, should be able to run the app. Only thing that might suffer on bigger screens is the User Interface layout, but the functionality won't be affected.

- Not all pictures can provide a clear picture on our LED-Matrix. For instance, JPEG file format performs less clear than PNG. This is of course due to the conversion quality of the picture format. To handle this we need to either perform better conversions or require PNGs as input. The latter is not favorable.
- Due to an issue in the Android libraries for the IOIO-OTG board, we are having issues displaying GIFs. This is a very rare problem so either we have to rewrite this part of the system to work around the problem or further troubleshooting the issue. As it is now, we are trying to find a solution with the creator of the IOIO-OTG, Ytai Ben-Tsvi. [18]
- SPI protocol has a range of bit-rates between 30Kbits/sec and 8Mbits/sec included in the IOIO-OTG standard programming. Therefore, we need to limit our sending rate to 1Mbits/sec max. Else the Bluetooth connection cannot keep up with the wired SPI protocol. In Section 3-3 we see that the nominal bit-rate we can get with our current prototype is about 1.440Mbits/s.

So, in short, using the Android and IOIO-OTG combination works well, but when the system would actually want to be commercial, a more complex development for specifically the communication between Android and LED-Matrix driver system is needed. Especially when Wi-Fi wants to be a possible wireless connection next to Bluetooth. Also the Android application as it is can definitely be optimized with processing speed and internal memory in mind.

Chapter 7

Hardware Design

In this chapter we will discuss three hardware designs related to our project.

7-1 Three Designs

Because the led strips are very flexible and transparent many designs are possible. The design we produced is a 32 by 32 matrix screen powered by the grid. However there are two other designs we would like to discuss. One is a design that can be fitted around a cup. This is a 5 by 16 design powered by a battery. We will also discuss a rounded ball design also powered by batteries. In the battery powered system power consumption is of very high importance and will be discussed thoroughly. However in the case of the grid powered system 240V AC needs to be converted into 5 V DC.

7-2 32 by 32 Matrix

To power the 32 by 32 matrix, we use a power supply that is connected to the grid. Firstly we use a transformer to convert the 230V of the grid to the 5V needed by the LED-strips. This voltage is then transformed into a DC voltage by a full bridge converter (the exact design is discussed later). To light the LED's about 1 A is used. The power consumption is about 5W. This is a very low power consumption, if we compare this with a single LEDARE Led-lamp E14 as advertised by IKEA (which uses about 7 W). A schematic of a 32 by 32 matrix can be seen in Figure 7-1.

7-2-1 Power Supply

The power supply is designed to produce enough current and voltage to supply for the LED strips. The LED strips need 3.3-5V and 24 mA per strip. Because all strips are connected in parallel the required voltage remains 3.3 to 5 V and the required current is about 1.5 A. The

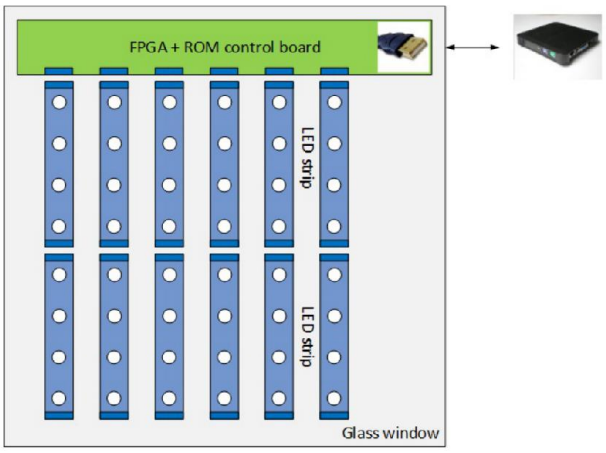

Figure 7-1: Schematics of 32 by 32 matrix

design for the Power supply can be seen in Figure 7-2. The design uses four diodes to convert AC to DC, capacitors to compensate ripple voltage and a resistor to reduce the current.

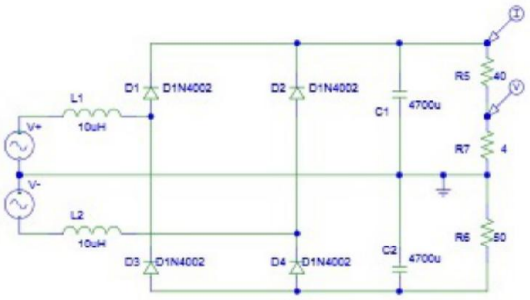

Figure 7-2: Schematics of powersupply

The power supply is also simulated by Pspice as can be seen in Figure 7-3. It can be noticed that both the current as the voltage have small ripples. The voltage is about 4.5 and the current about 1.5 A.

7-2-2 Final Power Supply

The final power supply that we used can be seen in figure 7-4. This is due to time constraint and long producing time of PCBs. However this power supply meets our specs in voltage and power and is therefore very suitable. [19]

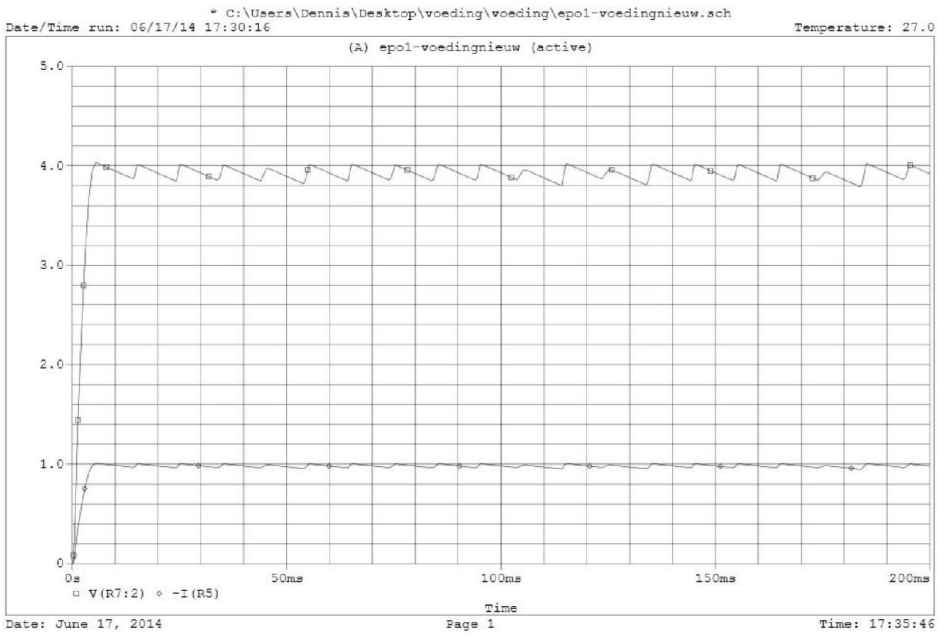

Figure 7-3: I-V Characteristic of Powersupply

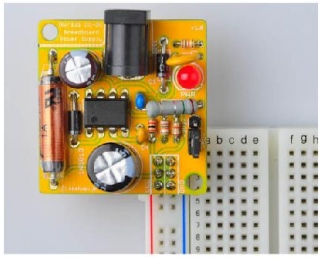

Figure 7-4: Picture of powersupply used

7-3 5 by 16 Cup Wrapper

One of our designs is a 5 by 16 cup wrapper. This cup wrapper can be used by coffee places like Starbucks to display advertisements. In this application power consumption is very important, because it has to be mobile and therefore carry its own power supply. As mentioned before the led drivers operate at a voltage of 3.3-5 V and use a current of up to 24 mA. Because there are 15 drivers in this 5 by 16 LEDs, the total power consumption will be about 66-100 mW. Conventional Duracell batteries 1.5 V (three in series) have a power storage of about 2,2 Ah this is enough for about three hours. If we want a system that is sustained for about nine hours we simply add 2 more sets of battery's in parallel.

Figure 7-5: LED-Cup

7-4 LED Ball

Another design that can be investigated is the LED Globe. This LED Globe could be used by companies to show their activities on the globe or as a circular display which can be viewed from all sides. This design can be manufactured in all sizes however our design will use 30 LED strips of each 16 LED. The power consumption of this device will be about 400-600mW. With the same Duracell batteries, 15 batteries (5 sets of 3 in parallel) are required to power the system for about three hours. However if nine hours of power is desired, 45 batteries are needed. Because 45 batteries is too much for a consumer product, it is recommended to use a larger power source(like an lithium-ion battery pack) or connect it to the grid as we do with the 32 by 32.

Chapter 8

PCB

8-1 Non Flexible 32 by 32

To create the 32 by 32 matrix, a PCB is needed. This PCB was designed using the software create by Designspark and produced by EuroCircuit. To reach low energy losses and keep the prototype affordable we used non flexible PCB design. The PCB contains connections all the input signals to all the LED-strips. There are 32 ten pin connectors to connect all 32 LED strips. In Figure 8-1, the ten pin connector used by the led strips is shown and in the table in Figure 8-3. There are also 32 data pins that collect all the data from the FPGA, one Vcc pin and one ground pin. In the design of the PCB different layer are used. In the top level there is a silk screen which can provide the design with text. Under this level there is a level that defines the drill holes. There are also two copper levels. We use two copper levels because more lines can be fitted on our PCB and lines can cross without connecting. The top copper layer and the bottom copper layer is present on the entire PCB except where the signal lines are present, which are insulated to prevent interference between the different connecting lines. All connector lines are made as thick as the connector holes to prevent as much energy losses in the PCB as possible.

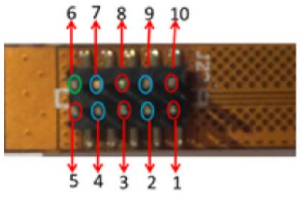

Figure 8-1: Ten pin connector

In Figure 8-4, the top level and silkscreen can be seen. This is how the PCB can be seen from above and were the pins will be soldered.

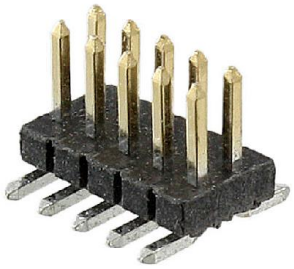

Figure 8-2: Ten pin connector

Connectorsnumbers	Signals
1,2,3	Vcc
4,5,6	GND
7	CLK
8	LE
9	OE
10	Data-in

Figure 8-3: Connector Information

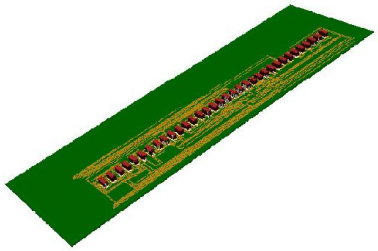

Figure 8-4: Toplevel of the PCB

PINS	diameter	distance between pins
10 pin connectors	0.65mm	1.27 mm
32 data in pins	0.75mm	2.5 mm
Vcc and ground pins	0.75	2.5 mm

Figure 8-5: Pin Information

8-2 Flexible PCB

For some applications (like globes or other round forms) flexible PCB technology is needed. To produce such flexible PCBs other materials are used. For this technology materials like PEEK, polyimide or transparent conductive polyester are used. For some applications transparent materials would be very good because if the display is turned off it would be almost entirely transparent. [20]

Chapter 9

Conclusion

In this chapter we will draw some conclusions and discuss our findings and experience during the process of development of the different blocks we had our parts in.

9-1 Android

Android is generally easy to use and a very cheap and user-friendly debut platform to build apps on. The only problem with this OS is the fact that your are very limited in communicating with other apps. So for instance, if you wanted to create an app that reads your mood according to the music you play or the messages you sent or post, you would find this is virtually impossible due to Androids Dalvik process virtual machine system.

Other than that, the Android OS system is great for developing apps, especially when you want to use most of the Android Device's integrated sensors and modules like the altitude sensors or WiFi module. That combined with the fact that the apps are Java based and that most engineers have the capability to write code in Java Environment.

The scalability of Android is also quite nice. If written correctly the app should be able to be used on a variety of Android devices, however there is one problem with this. Every Android device's infrastructure is different so there possibility that an app might not work on some devices is possible and needs to be debugged when problems arises.

The developed app works nicely for what we wanted to achieve with it. We are able to successfully send images to a screen as seen in Figure 9-1. In the Figure you can even see the App controlling the screen here. The app includes some extra features in which we can display images obtained from other means than from the device's memory. One way is by putting in a text and returning an image containing that text and a way by grabbing information from the internet, as seen in Figure 9-2

In Figure 9-2 the rescaling effect is also seen in action. This shows that our algoritm to get as much of the picture in the screen is working correctly.

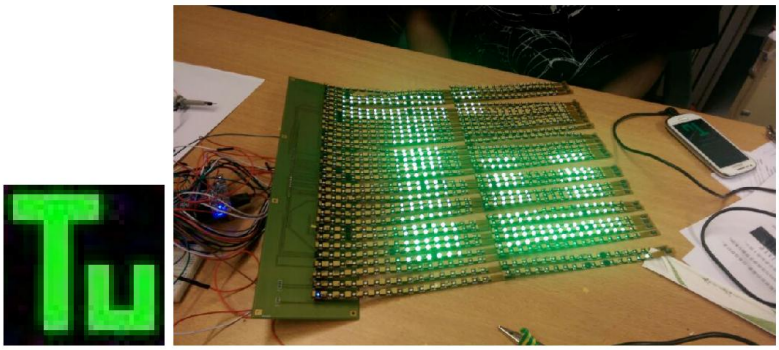

Figure 9-1: Left shows input picture and right shows resulting output at the LED-Matrix

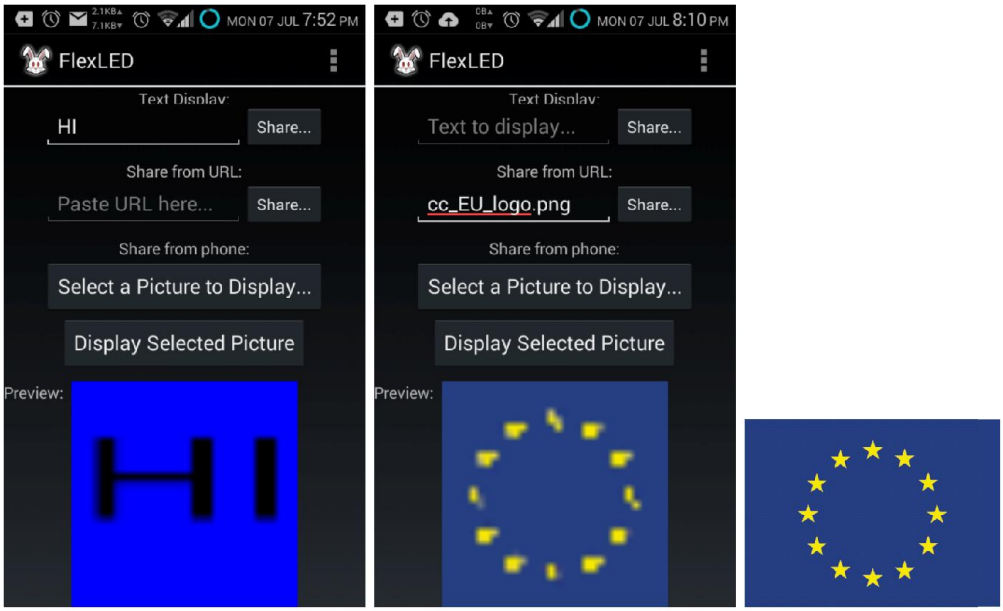

Figure 9-2: Left shows the custom text feature at work, in the middle the rescaling (32x32) and internet feature is shown with it's input image (944x694,*connectedcities.eu/downloads/coordination/cc_eu_logo.png*) on the right.

The last extra feature is Animation. This was very early in development and was not finished at the end of the project, however we were able to show a moving dot from one side of the screen to the other side with a nice framerate, although we could not set this to optimal due to some communication issues existing between the IOIO-OTG and the FPGA. The FPGA design to accept SPI data flow was not optimal and was very hard to debug. Also in the FPGA we saw that rewriting an image to the RAM wasn't optimal with it's result for seqquences of images and needed to be developed further to make animations an easier task. Either we needed to debug this, which required more time, or, like mentioned in Section 9-5, wrote the firmware of the IOIO-OTG to replace the FPGA board so timing and storage would be easier to account for.

9-2 IOIO-OTG

The IOIO-OTG proved to be very useful for the project as a communication device between Android and hardware. The only problem that arised with it is the fact that the device does not have the best documentation and support. Meaning when there is a problem with the device, there is little support you can expect. In our experience we saw that our developed app was not sending information back from the IOIO-OTG. The cause of this problem could not be identified by the creator of the device, forcing us to figure out the solution ourselves. This solution was found by us later to be a problem of not having included the build path to the IOIO libraries. This library was being called externally so if there was any request for a variable, it wouldn't arrive to the app.

This allowed us to finally check if the SPI protocol was working. The result for this would be to big to produce in this Thesis. What we did is have the IOIO-OTG loop back the information it was sending and make a file with the bytes received and compared that with the input. This proved to be succesful.

If we want to use this system to use Wi-Fi communications, then the IOIO-OTG would not be needed or it's firmware would have to be mostly rewritten to accomodate for Wi-Fi. So to do this, more time needs to be invested in creating Android libraries and writing a firmware for the android. Like we saw in Chapter 3 we would then be able to accomodate for more users at once and have bigger signal range.

9-3 PCB

For desigining a PCB, it is advisable to make your design ahead of time so there is enough time to either show your design to someone with more experience or have the design be checked by the involved producers, so changes can be made early on and the board will be finished on time. This process was not known to us and taken for granted, leading to the board being ready to use at the last moments of the project.

While our designed PCB works correctly as proven in Figure 9-1, we soon realized that the signal for LE and CLK was decaying in power due to the load of the strips. This means that that more strips are connected, the weaker the two signals will be. To compensate for this we would have to have a seperate LE and CLK signal per approx. 5 strips or we would have to make seperate PCBs for 5 strips seperately. Next to the LE and CLK we experienced some power issue with for instance displaying full white screen. This would result into a red screen since there isn't enough power available for the full 32x32 screen. To compensate for this we would as mentioned earlier have to probably power the strips per 5 strips with the same power supply or seperate PCBs per 5 strips with their own power supply.

9-4 Power Supply

Due to various time constraints and, in hind sight, bad time scheduling, we could not finish the design on the power supply and couldn't order a premade one. Instead we used a standard power supply with a rating of 7V and 6A. The LEDs worked at a voltage of 3.3-3.9V and the

current stayed around 1.80A for the full 32x32 matrix. This was not enough power to fully display a white screen. Either the power supply needs to be more persistent (not recommended due to component safety) or we need to integrate power supplies into the PCBs per 5 strips as mentioned in Section 9-3.

9-5 Overall System

Ultimately to have a complete and much smoother working system than our current system, we would have to integrate everything from the IOIO-OTG functions and the FPGA functions into one module. We need to have the IOIO-OTG firmware be written in such a way that the FPGA would be replaced and therefor the communication between IOIO-OTG circuit board and FPGA would be nullified. This last mentioned communication was giving us the most problems so if that communication does not exist, our system would give much better result and would even allow animations to be an easier implementable feature, assuming you install a memory to the IOIO-OTG circuit board. Another option is to create a system that would integrate both the wireless communication option and the driving of the LED-Matrix into one chip. In both these cases we can use the same Android app and the modified version of the PCB and Power Supply to have a robust, fast and power efficient system overal.

Appendix A

Android Codes

Below you will find all codes mentioned throughout Chapter 6 "Android".

A-1 Basic Functionality Program Code

For better readability the code is split in separate paragraphs with their own respective titles as referenced in the Chapter 6 "Android". All paragraphs together make up the full code for basic functionality. This does not include the extra features.

A-1-1 Library Imports

```
1  package com.flexled.main;
2
3  import java.io.BufferedInputStream;
4  import java.io.File;
5  import java.io.FileOutputStream;
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.net.MalformedURLException;
9  import java.net.URL;
10 import java.net.URLConnection;
11 import java.nio.ByteBuffer;
12 import java.util.ArrayList;
13 import java.util.Arrays;
14 import java.util.List;
15 import java.util.concurrent.ExecutionException;
16
17 import ioio.lib.api.exception.ConnectionLostException;
18 import ioio.lib.spi.Log;
19 import ioio.lib.util.BaseIOIOLooper;
20 import ioio.lib.util.IOIOLooper;
```

```
21 import ioio.lib.util.android.IOIOActivity;
22 import android.R.style;
23 import android.graphics.Bitmap;
24 import android.graphics.Bitmap.Config;
25 import android.graphics.BitmapFactory;
26 import android.graphics.Canvas;
27 import android.graphics.Color;
28 import android.graphics.Matrix;
29 import android.graphics.Paint;
30 import android.graphics.Rect;
31 import android.graphics.Paint.Style;
32 import android.os.AsyncTask;
33 import android.os.Bundle;
34 import android.os.Environment;
35 import android.view.Menu;
36 import android.view.MenuInflater;
37 import android.view.MenuItem;
38 import android.view.MotionEvent;
39 import android.view.VelocityTracker;
40 import android.view.View;
41 import android.view.WindowManager;
42 import ioio.lib.api.DigitalInput;
43 import ioio.lib.api.DigitalInput.Spec.Mode;
44 import ioio.lib.api.DigitalOutput;
45 import ioio.lib.api.SpiMaster;
46 import ioio.lib.api.SpiMaster.Rate;
47 import android.view.View.OnClickListener;
48 import android.annotation.TargetApi;
49 import android.app.AlarmManager;
50 import android.app.Dialog;
51 import android.app.PendingIntent;
52 import android.content.Context;
53 import android.content.Intent;
54 import android.database.Cursor;
55 import android.net.Uri;
56 import android.os.Build;
57 import android.provider.MediaStore;
58 import android.widget.ArrayAdapter;
59 import android.widget.Button;
60 import android.widget.CheckBox;
61 import android.widget.EditText;
62 import android.widget.ImageView;
63 import android.widget.LinearLayout;
64 import android.widget.Spinner;
65 import android.widget.TextView;
```

A-1-2 The Main Activity

Initiation global variable and start of app.

```
1
2 /**
3  * This is the main activity of the FlexLed app.
```

```
4  */
5 public class MainActivity extends IOIOActivity{
6   //initiation attributes
7   private Button picdisplayButton;
8   private Button rstButton;
9   private ImageView pictest;
10  private static TextView test;
11  private static TextView test2;
12
13  //intiation variables
14  //width and height of the LED-matrix
15  private static int picw =32;
16  private static int pich = 5;
17  private static byte[] buffer = new byte[picw*pich*3];
18  private static byte[] bufferin= new byte[picw*pich*3];
19  private static boolean picdone = true;
20
21  /**
22   * Get attributes from layout.
23   */
24  @Override
25  protected void onCreate(Bundle savedInstanceState) {
26    super.onCreate(savedInstanceState);
27    //show splashscreen for atleast 1 second.
28    try {
29      Thread.sleep(1000);
30    } catch (InterruptedException e) {
31      e.printStackTrace();
32    }
33
34    setTheme(style.Theme_Holo);
35    setContentView(R.layout.fin_main);
36
37    //create all listners for buttons.
38    addListenerOnButton();
39  }
```

A-1-3 Add Listeners to Button

Adds onClickListeners to button. Asks user to choose an image.

```
1
2   /**
3    * Setting all onClickListeners() for buttons
4    */
5   public void addListenerOnButton() {
6
7         //Buttons
8         picdisplayButton = (Button) findViewById(R.id.button3);
9         //end Buttons
10
11        //ImageViews
```

```
12          pictest = (ImageView) findViewById(R.id.imageView1);
13          //end ImageViews
14
15
16           this.getWindow().setSoftInputMode(WindowManager.LayoutParams.
                SOFT_INPUT_STATE_ALWAYS_HIDDEN);
17          //start setting of onClickListners() for Buttons
18      /**
19       * Button should take a picture chosen from phone and
20       * pass it to onActivityResult()
21       */
22      picdisplayButton.setOnClickListener(new OnClickListener() {
23        @Override
24        public void onClick(View v) {
25          if (v.getId() == picdisplayButton.getId()) {
26            if (picw == 0){
27              picw = 32;
28            }
29            if (pich == 0){
30              pich = 32;
31            }
32            buffer = new byte[picw*pich*3];
33            Intent intent = new Intent(Intent.ACTION_PICK,
34                android.provider.MediaStore.Images.Media.
                    EXTERNAL_CONTENT_URI);
35            startActivityForResult(Intent.createChooser(intent, "Select
                Picture"), 0);
36          }
37
38        }
39      });
40    }
```

A-1-4 Processing of Chosen Picture

Takes chosen picture and distinguishes if animation or not. Prepares picture to be send.

```
1
2     /**
3      * Further processing of Pictures or GIFs gotten from phone storage.
4      * Picture and GIF processing is done separated.
5      * Extension check is done to distinguish.
6      */
7     @TargetApi(Build.VERSION_CODES.GINGERBREAD_MR1) @SuppressWarnings("
          deprecation") @Override
8     protected void onActivityResult(int requestCode, int resultCode, Intent
          data) {
9       super.onActivityResult(requestCode, resultCode, data);
10      if (requestCode ==0){
11        if (resultCode == RESULT_OK) {
12          Uri targetUri = data.getData();
13
```

```
14          String[] proj = { MediaStore.Video.Media.DATA };
15          Cursor cursor = managedQuery(targetUri, proj, null, null, null);
16          int column_index = cursor.getColumnIndexOrThrow(MediaStore.Video.
                Media.DATA);
17          cursor.moveToFirst();
18
19          //Check if chosen picture is a gif.
20          if (cursor.getString(column_index).substring(cursor.getString(
                column_index).lastIndexOf(".")).equals(".gif"))
21          {//GIF File
22            gifView.setGif(cursor.getString(column_index));
23
24
25            decoder = new GifDecoder();
26            decoder.read(gifView.getInputStream());
27
28            for(int i = 0;i< decoder.getFrameCount();i++){
29              //check if GIF is animation
30              if (!(decoder.getDelay(i)<= 0))
31              {
32                for (int j = 0 ;j< decoder.getDelay(i);j++)
33                {
34                    bmlist.add(RGBdrive.rescale(decoder.getFrame(i),0,0));
35                  try {
36                    RGBdrive.rgbDrive(RGBdrive.rescale(decoder.getFrame(i)
                          ,0,0));
37                  } catch (ConnectionLostException e) {
38                  // TODO Auto-generated catch block
39                    e.printStackTrace();
40                  } catch (InterruptedException e) {
41                    // TODO Auto-generated catch block
42                  e.printStackTrace();
43                  }
44                }
45              }
46              else
47              {
48                bmlist.add(RGBdrive.rescale(decoder.getFrame(i),0,0));
49                try {
50                  RGBdrive.rgbDrive(RGBdrive.rescale(decoder.getFrame(i)
                        ,0,0));
51                } catch (ConnectionLostException e) {
52                  e.printStackTrace();
53                } catch (InterruptedException e) {
54                  e.printStackTrace();
55                }
56              }
57              try {
58                Thread.wait(1000);
59              } catch (InterruptedException e) {
60                e.printStackTrace();
61              }
62            }
```

```
63            pictest.setImageBitmap(RGBdrive.rescale(decoder.getFrame(0)
                  ,0,0));
64          }
65
66          else
67          {//Image File
68            Bitmap bitmapb;
69            Bitmap tempbm = BitmapFactory.decodeFile(cursor.getString(
                  column_index));
70            int h = tempbm.getHeight();
71            int w = tempbm.getWidth();
72            //save last chosen picture. TODO: remember picture path instead
                   of picture.
73            if (h>w)
74            {
75              lastfullbm = Bitmap.createScaledBitmap(tempbm,(int) (w*500/h)
                    ,500, true);
76            }
77            if (h<w)
78            {
79              lastfullbm = Bitmap.createScaledBitmap(tempbm, 500,(int) (h
                  *500/w), true);
80            }
81
82            bitmapb = RGBdrive.rescale(tempbm,0,0);
83            lastpic = bitmapb;
84            pictest.setImageBitmap(bitmapb);
85            try {
86              RGBdrive.rgbDrive(bitmapb);
87            } catch (ConnectionLostException e) {
88              e.printStackTrace();
89            } catch (InterruptedException e) {
90              e.printStackTrace();
91            }
92          }
93        }
94      }
95    }
```

A-1-5 Custom Processing Methods for Bitmaps

First method: Rescaling of bitmaps to requested size, while avoiding stretching.

Second Method: Comparing two Bitmaps.

Third Method: Transforming Bitmap using requested settings.

Last Method: Turning Bitmap in byte array containing RGB information. Array is ready to be split and send.

```
1
2   /**
3    * Custom Bitmap processing
```

```
4     * @author DeviousSiddy
5     *
6     */
7    static class RGBdrive{
8
9      /**
10      * bm is cropped to a square and then rescaled to requested size
           picwxpich
11      * Also crops to center of picture when picture is not 1:1 ratio.
12      * @param bm
13      * @return
14      */
15     public static Bitmap rescale(Bitmap bm,int d_x,int d_y){
16       if (bm!=null){
17         float scale;
18             if (bm.getWidth() < bm.getHeight()) {
19                 scale = picw / (float) bm.getWidth();
20             } else {
21                 scale = pich / (float) bm.getHeight();
22             }
23             Matrix matrix = new Matrix();
24             matrix.setScale(scale, scale);
25             Bitmap thumbnail = transform(matrix, bm, picw, pich,
26                 0x1 | 0x0, d_x,d_y);
27             return thumbnail;
28       }
29       return bm;
30     }
31     public static boolean bmEquals(Bitmap bitmap1, Bitmap bitmap2) {
32         ByteBuffer buffer1 = ByteBuffer.allocate(bitmap1.getHeight() *
              bitmap1.getRowBytes());
33         bitmap1.copyPixelsToBuffer(buffer1);
34
35         ByteBuffer buffer2 = ByteBuffer.allocate(bitmap2.getHeight() *
              bitmap2.getRowBytes());
36         bitmap2.copyPixelsToBuffer(buffer2);
37
38         return Arrays.equals(buffer1.array(), buffer2.array());
39     }
40      private static Bitmap transform(Matrix scaler,
41                 Bitmap source,
42                 int targetWidth,
43                 int targetHeight,
44                 int options,
45                 int xdev,
46                 int ydev) {
47             boolean scaleUp = (options & 0x1) != 0;
48             boolean recycle = (options & 0x2) != 0;
49
50             int deltaX = source.getWidth() - targetWidth;
51             int deltaY = source.getHeight() - targetHeight;
52             if (!scaleUp && (deltaX < 0 || deltaY < 0)) {
53                 /*
```

```
54                  * In this case the bitmap is smaller, at least in one
                        dimension,
55                  * than the target.  Transform it by placing as much of
                        the image
56                  * as possible into the target and leaving the top/bottom
                        or
57                  * left/right (or both) black.
58                  */
59                  Bitmap b2 = Bitmap.createBitmap(targetWidth, targetHeight
                        ,
60                  Bitmap.Config.ARGB_8888);
61                  Canvas c = new Canvas(b2);
62
63                  int deltaXHalf = Math.max(0, deltaX / 2);
64                  int deltaYHalf = Math.max(0, deltaY / 2);
65                  Rect src = new Rect(
66                  deltaXHalf,
67                  deltaYHalf,
68                  deltaXHalf + Math.min(targetWidth, source.getWidth()),
69                  deltaYHalf + Math.min(targetHeight, source.getHeight()));
70                  int dstX = (targetWidth  - src.width())  / 2;
71                  int dstY = (targetHeight - src.height()) / 2;
72                  if (Math.abs(xdev)>dstX)
73                {
74                  if (xdev<0)
75                  {
76                    xdev = -dstX;
77                  }
78                  else
79                  {
80                    xdev = dstX;
81                  }
82                }
83                if (Math.abs(ydev)>(dstY))
84                {
85                  if (ydev<0)
86                  {
87                    ydev = -dstY;
88                  }
89                  else
90                  {
91                    ydev = dstY;
92                  }
93                }
94                  Rect dst = new Rect(
95                          dstX+xdev,
96                          dstY+ydev,
97                          targetWidth - dstX,
98                          targetHeight - dstY);
99                  Log.d("Rudy",Integer.toString( dstX+xdev )+Integer.
                        toString(dstY+ydev));
100                 c.drawBitmap(source, src, dst, null);
101                 if (recycle) {
```

```
102                     source.recycle();
103                 }
104                 c.setBitmap(null);
105                 return b2;
106             }
107             float bitmapWidthF = source.getWidth();
108             float bitmapHeightF = source.getHeight();
109
110             float bitmapAspect = bitmapWidthF / bitmapHeightF;
111             float viewAspect   = (float) targetWidth / targetHeight;
112
113             if (bitmapAspect > viewAspect) {
114                 float scale = targetHeight / bitmapHeightF;
115                 if (scale < .9F || scale > 1F) {
116                     scaler.setScale(scale, scale);
117                 } else {
118                     scaler = null;
119                 }
120             } else {
121                 float scale = targetWidth / bitmapWidthF;
122                 if (scale < .9F || scale > 1F) {
123                     scaler.setScale(scale, scale);
124                 } else {
125                     scaler = null;
126                 }
127             }
128
129             Bitmap b1;
130             if (scaler != null) {
131                 // this is used for minithumb and crop, so we want to
                        filter here.
132                 b1 = Bitmap.createBitmap(source, 0, 0,
133                 source.getWidth(), source.getHeight(), scaler, true);
134             } else {
135                 b1 = source;
136             }
137
138             if (recycle && b1 != source) {
139                 source.recycle();
140             }
141
142             int dx1 = Math.max(0, b1.getWidth() - targetWidth);
143             int dy1 = Math.max(0, b1.getHeight() - targetHeight);
144             if (Math.abs(xdev)>(dx1/2))
145             {
146               if (xdev<0)
147               {
148                 xdev = -dx1/2;
149               }
150               else
151               {
152                 xdev = dx1/2;
153               }
```

```
154             }
155             if (Math.abs(ydev)>(dy1/2))
156             {
157               if (ydev<0)
158               {
159                 ydev = -dy1/2;
160               }
161               else
162               {
163                 ydev = dy1/2;
164               }
165             }
166             Bitmap b2 = Bitmap.createBitmap(
167                     b1,
168                     dx1 / 2 + xdev,
169                     dy1 / 2 + ydev,
170                     targetWidth,
171                     targetHeight);
172
173             if (b2 != b1) {
174                 if (recycle || b1 != source) {
175                     b1.recycle();
176                 }
177             }
178
179             return b2;
180         }
181     /**
182      * bm is converted to a byte array that stores
183      * Green values in entries 1 to 16, 49 to 65...
184      * Red values in entries 17 to 32, 66 to 81...
185      * Blue values in entries 33 to 48, 81 to 96...
186      * @param bm
187      * @throws InterruptedException
188      * @throws ConnectionLostException
189      */
190     public static void rgbDrive(Bitmap bm) throws ConnectionLostException
         , InterruptedException{
191       if (bm !=null){
192
193         int width = bm.getWidth();
194             int height = bm.getHeight();
195
196              int[] pix = new int[picw * pich];
197         bm.getPixels(pix, 0, picw, 0, 0, picw, pich);
198         for (int stripn = 0;stripn<pix.length/16;stripn++) {
199           for (int pixn = 0;pixn<16;pixn++){
200             byte r = (byte) ((pix[pixn+(stripn*16)] >> 16)&0xff);
201             byte g = (byte) ((pix[pixn+(stripn*16)] >> 8) &0xff);
202             byte b = (byte) (pix[pixn+(stripn*16)]&0xff);
203             buffer[((16)*(stripn*3))+pixn] = g;//Green assignment
204             buffer[((16)*((stripn*3)+1))+pixn] = r;//Red assignment
205             buffer[((16)*((stripn*3)+2))+pixn] = b;//Blue assignment
```

```
206             }
207           }
208           picdone = false;
209         }
210       }
211     }
```

A-1-6 Sending Information

Class to communicate with the IOIO-OTG. Splits array from last mentioned method in sendable packages.

```
1
2   /**
3    * IOIOLooper class. Repeats continuously as IOIO is connected.
4    * @author DeviousSiddy
5    *
6    */
7   static class Looper extends BaseIOIOLooper {
8     //initiation of SPI protocol variable
9
10    private SpiMaster spi_;
11    private DigitalOutput pin8;
12
13
14
15    /**
16     * Called every time a connection with IOIO has been established.
17     * Typically used to open pins.
18     *
19     * @throws ConnectionLostException
20     *             When IOIO connection is lost.
21     *
22     * @see ioio.lib.util.AbstractIOIOActivity.IOIOThread#setup()
23     */
24    @Override
25    protected void setup() throws ConnectionLostException {
26      pin8 = ioio_.openDigitalOutput(8, false);//image_enable
27    }
28
29
30
31    /**
32     * Called repetitively while the IOIO is connected.
33     *
34     * Takes created byte[] buffer and splits the it into byte[] of size
         64
35     * since spi can only send 64 bytes per time.
36     *
37     * @throws ConnectionLostException
38     *             When IOIO connection is lost.
39     * @throws InterruptedException
```

```
40       *
41       * @see ioio.lib.util.AbstractIOIOActivity.IOIOThread#loop()
42       */
43      @Override
44      public void loop() throws ConnectionLostException,
            InterruptedException {
45
46        if (picdone == false)
47        {
48            //picture send process
49            spi_ = ioio_.openSpiMaster(new DigitalInput.Spec(4,
50                   Mode.PULL_UP), new DigitalOutput.Spec(5),
51                   new DigitalOutput.Spec(3),
52                   new DigitalOutput.Spec[] { new DigitalOutput.Spec(2) },
53                   new SpiMaster.Config(Rate.RATE_1M, false, true));
54            byte[] buffer_= buffer;
55            int buffernum = (int) Math.ceil(buffer_.length/48);
56
57            byte[] buffer1_= new byte[48];
58            pin8.write(true);
59            Thread.sleep(200);
60            for (int i =0;i<buffernum;i++){
61
62              System.arraycopy(buffer_, i*48, buffer1_, 0, 48);
63              spi_.writeRead( buffer1_, buffer1_.length,
64                  buffer1_.length, null,0);
65
66              Thread.sleep(50);
67
68            }
69
70            Thread.sleep(200);
71            pin8.write(false);
72            spi_.close();
73          picdone =true;//must be true to send 1 image per time.
74        }
75
76      }
77
78
79
80    }
81
82    protected IOIOLooper createIOIOLooper() {
83      return new Looper();
84    }
85
86
87  }
```

Bibliography

[1] "Bluetooth fast facts." http://www.bluetooth.com/Pages/Fast-Facts.aspx, visited on 2014-6-10.

[2] "Bluetooth basics." http://www.bluetooth.com/Pages/Basics.aspx, visited on 2014-6-10.

[3] "Wi-fi - webopedia." http://www.webopedia.com/TERM/W/Wi_Fi.html, visited on 2014-6-10.

[4] "Wireless network explained." http://www.webopedia.com/DidYouKnow/Computer_Science/wireless_networks_explained.asp, visited on 2014-6-10.

[5] "Bluetooth vs wi-fi - difference and comparison." http://www.diffen.com/difference/Bluetooth_vs_Wifi, visited on 2014-6-10.

[6] "Spi." https://github.com/ytai/ioio/wiki/SPI, visited on 2014-5-10.

[7] "Twi." https://github.com/ytai/ioio/wiki/TWI, visited on 2014-5-10.

[8] "I2c - wikipedia." http://en.wikipedia.org/wiki/I%C2%B2C, visited on 2014-5-10.

[9] "Uart." https://github.com/ytai/ioio/wiki/UART, visited on 2014-5-10.

[10] "Uart - wikipedia." http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter, visited on 2014-5-16.

[11] Wikipedia, "Android - wikipedia." http://en.wikipedia.org/wiki/Android_(operating_system), visited on 2014-6-6.

[12] L. Mahapatra, "Android vs. ios: What's the most popular mobile operating system in your country?." http://www.ibtimes.com/android-vs-ios-whats-most-popular-mobile-operating-system-your-country-1464892, visited on 2014-6-6, Nov 2013.

[13] Phonearena.com, "Android's google play beats app store with over 1 million apps, now officially largest." http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680, visited on 2014-6-6, Jul 2013.

[14] J. Patel, "Key difference between submitting your app on google play store and apple app store." http://www.whatech.com/mobile-apps/21330-key-difference-between-submitting-your-app-on-google-play-store-and-apple-app-%store, visited on 2014-6-6, Jun 2014.

[15] R. Amadeo, "Google's iron grip on android: Controlling open source by any means necessary." http://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/, visited on 2014-6-6, Oct 2013.

[16] Wikipedia, "ios - wikipedia." http://en.wikipedia.org/wiki/IOS, visited on 2014-6-6.

[17] Y. Ben-Tsvi, "Go, go, ioio-on-the-go!." http://ytai-mer.blogspot.nl/2013/01/go-go-ioio-on-go.html, visited on 2014-5-6.

[18] Y. Ben-Tsvi, "Ioio users google group." https://groups.google.com/forum/#!forum/ioio-users, visited on 2014-5-10, Oct 2013.

[19] "Description of used powersupply." http://www.eztronics.nl/webshop2/catalog/PowerSupplies/MiscPower?product_id=278, visited on 2014-6-10.

[20] D. Shavit, "The developments of leds and smd electronics on transparent conductive polyester film," *Vacuum International*, 2007.

[21] "Stack overflow." http://stackoverflow.com/, visited on 2014-5-6.

[22] "Android developers." http://developers.android.com, visited on 2014-5-6, Jan 2013.

[23] Y. D. Liang, *Introduction to Java Programming, sixth edition.* 2004.