Smart Phone Powered Laptop

Nick Steele, Kevin Ogando, Ameer Hakh, Anirudh Singh

University of Central Florida School of Electrical Engineering and Computer Science, Orlando, FL

Abstract – An unique computer and electrical engineering project where we utilize the many components of a smartphone (CPU, RAM, Storage, Wi-Fi Card) to run a windows like experience of android on a laptop shell. The laptop itself is just a screen, battery, keyboard, and touchpad. The phone is charged when it is put inside the laptop. The keyboard, touchpad, and screen all communicate wirelessly with the smartphone via Wi-Fi and Bluetooth.

I. INTRODUCTION

In today's world you will notice the majority of people have a cell phone. This is because Smartphones are widely accepted as a necessity in today's society. Smartphones are the modern technological multi-tool; they provide security, entertainment, social interaction, and almost infinitely other things. In the United States, children are getting Smartphones as early as age 6. This fact speaks volumes about how accessible phones are in today's Society. Unfortunately, with the advancement in technology has also come with an increase in price. Smartphones are costing anywhere from seven hundred to a thousand dollars each. In our modern market, this is very similar with today's prices of laptops. To some, this price comparison makes sense; we do a lot of similar work on our smart phones as we do our laptops today. This includes things such as checking email, browsing the web, watching videos, and even playing games. Even from a hardware level these devices have a similar setup. Both devices include a central processing unit (CPU), a graphics processing unit (GPU), a high-resolution screen, a battery, a camera, and storage. Showing how similar both devices are and how commonly we use them for the same tasks makes you wonder, why do we have to buy both? Why not have devices that can serve as both a laptop and a smartphone? Not only would this save the consumer hundreds of dollars on average, but it could also make laptop style computer more accessible to those who cannot afford both. Laptops are a common learning tool in today's schools and with most students and children already having these smart devices, making it even that much more crucial that laptops are affordable and accessible to students. Our project makes this Possible because were not using a fully functional machine, the goal of our project is to use a non-functional computer and apply functionality to it with the help of a microcontroller unit, Bluetooth module, and Wi-Fi module housed on a printed circuit board will give our product the same functionality as a normal functional computer. However, the main component required to demonstrate the design is an Android device. The entire design is dependent on the smartphone since the laptop will only display the contents of the smartphone. In conclusion our product offers the use of a computer through an Android smart phone.

II. SYSTEM COMPONENTS

The system is best presented in terms of system components which are the individual physical modules whether purchased or designed that are interfaced to create the final product. This section provides technical detail of each component used.

A. Microcontroller

For the development of the final prototype of the smart phone powered laptop, two microcontrollers were required. First, was the ATMega 2560 and second was the Raspberry pi. The ATMega 2560 is taking the inputs from the ASUSG50V laptop keyboard and touchpad and sending that to the android phone via the RN42HID Bluetooth module. The other choices instead of ATMega2560 were the ATMega328 and the MSP430F4551PN. The reason behind picking ATMega2560 over the ATMega 328 is that the ATMEGA 2560 consists of 54 Digital Input/output pins whereas the ATMega 328 only consists of 14 Digital Input/output pins. The Digital Input/output pins was a huge factor for deciding on which Arduino mcu to choose because the keyboard consists of a ribbon cable since it is the keyboard of the shell laptop (ASUSG50V) and the ribbon cable has 24 pins so ATMega 328 was clearly not the right choice since it does not contain enough digital I/o pins to establish a successful connection between the keyboard and the Arduino mcu, hence the ATMega 2560 Arduino was used.

The second reason for choosing the ATMega 2560 over the MSP430f4551PN was the desired clock rate and the core size. The MSP430F4551PN has a clock rate of 8MHz and a core size of 16 bit whereas the ATMega 2560 has a clock rate of 16MHz and a core size of 64 bit which allows it to produce the output at a faster rate than the MSP430F4551PN.

The Raspberry pi is the heart of the project because the soul function of the raspberry pi is to mirror the android phone screen onto the LCD screen of the shell laptop. The reason behind choosing the raspberry pi for screen mirroring is that it matches the Wi-Fi direct standard for screen mirroring and it has enough computing power to process the data that is sent from the android phone to the LCD screen of the shell laptop.

B. Keyboard

Using our original Asus Keyboard (part# 04GNED1KUK10), we established our Bluetooth connectivity pairing this with the atMega2560. We learned that keyboards function from a matrix to properly output the desired key. We looked online for blueprint for our existing keyboard but found no data. In an attempt to identify each key on our own, we used a multimeter and started independently testing connection combinations while pressing the keyboard keys. We found this to be incredibly time consuming and started looking for alternatives to map out our key matrix. We came across a project using a teensyduino and a sketch that ran an automated continuity tester that would print out our whole matrix. This worked successfully with our keyboard. Once this was working we needed to find a library that would work with our matrix to output the keys. We originally tried to use the keyboard.h library but found out that it only worked on Arduino devise using the ATmega32u4 processor. Because we really wanted to use the appropriate Arduino library to make our keyboard work, we attempted to use work around that would allow us to use the keyboard.h library on other Arduino boards. This method had to use one of the two processors on the ATmega2560 and have us use it as a ATmega32u4 processor. This method didn't work for us because we still didn't have enough pins to map out our 16 by 8 matrix to use our whole keyboard. Finally, we found the keypad.h library which could be used on any Arduino and used a matrix just like that of a keyboard. Our issues with the library were that it was difficult getting the multi-key press support to work properly because there was no documentation. Once we had this working, we developed our Bluetooth code to send our raw HID packets. The only issue left was the Bluetooth function required hexadecimal, but our keypad library code could only output characters. We simply then wrote a function to convert the characters to hexadecimal.

To summarize our functionality When a key is pressed its state is recorded by the keypad library, the library outputs a character which is sent through our custom transfer function which changes the code from its character value to the proper hexadecimal code for Bluetooth. Finally, our Bluetooth function packages the HID output with the proper hex from the transfer function. This then outputs the proper keys to our android phone.

C. Synaptics Touchpad

The TM- is a touchpad revived from the ASUS G50 laptop. The Synaptics touchpad communicates using a bidirectional synchronous serial protocol, which is known for transmitting signals on data and clock lines. In this project the touchpad will act just as it usually acts on a computer, except wirelessly to an Android phone. Due to its raw data, a program must be made to follow the PS/2 communication. This protocol consumes roughly 100mA at 5V, which equates to 0.5 W power consumed.

D. Bluetooth

The RN42 is a fully certified Class 2 Bluetooth module with a built in antenna, shield, and an onboard Bluetooth stack. Furthermore, this module supports HID firmware, which allows for easy do it yourself computer peripherals, such as a mouse, keyboard, and even a joystick. Communication can be established using UART or a simple USB hardware interface with a baud rate of up to 115200 bps. In this project, UART is used due to the project complexity and necessary I/O pins.

E. Regulators

Regulators are fuel for the engine, which makes them so important when picking components. In this case, each module, component, or IC needs its own fuel and they all have various current requirements.

F. System Hardware Concept

In a system, software and hardware work together to achieve an overall working system. Now let's look at how the software is implemented through hardware. This block diagram indicates hardware flow starting at the lower level components located inside the Atmel 2560 chip and flowing all the way to the user peripherals such as the keyboard, touchpad, and LCD.

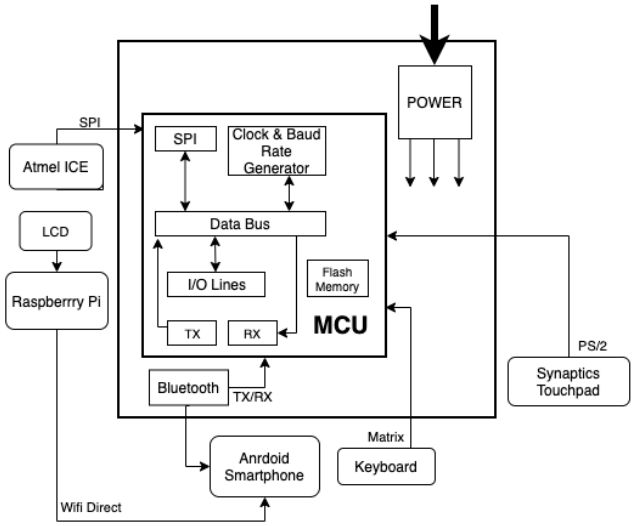


Fig. 1. Block Diagram showing major internal components on the MCU and external components.

The main communication busses shown in the system are:

(1) SPI to program the Atmega 2560 using Atmel Ice Debugger.
(2) UART to configure Bluetooth and transmit wirelessly.
(3) Wi-Fi Direct to communicate with Android phone wirelessly to LCD display.

G. USB 2.0 Phone Charging

While having the Wi-Fi and Bluetooth on in a cellular device, these peripherals will take up a lot of battery power. In order to keep the Android device powered on and to charge the Android device to an efficient battery level. We do this by implementing USB 2.0 charging, with the LM7805 chip. Also, particularly with Android devices, there must be a minimum threshold of 1.2V between the data lines (D- and D+) in order to charge the phone at a higher current than the minimum 500mA that is mandated in the USB 2.0 power specifications.

H. Battery Charging

In this system we are using lithium ion batteries, discussed in section G, so we can implement a charging system to recharge the lithium ion cells. We do this by using the BQ24600 chip from Texas Instruments. We have an input voltage from a 12V AC adapter plugged into the wall outlet to the DC jack on our charging circuit. The charging circuit then outputs a charging current of 3A to the battery, while also monitoring the temperature of the battery via a thermistor and can monitor the battery capacity in order to either charge or terminate the charge to the battery.

I. Battery LED Indication

In order for us to identify what percentage the battery is currently at, we have implemented an LED indication circuit by using an LED driver chip called LM3914 by Texas Instruments. With this LED driver circuit we can accurately display the capacity of the battery at about a 0.1V discrepancy. We have a total of 10 LEDs, 6 Green, 3 Yellow, and 3 Red LEDs. The green LEDs indicate a higher capacity, while the yellow and red LEDs indicate that the battery be plugged in to charge.

J. Voltage Levels

Along with the software components of the system, let's take a look at the power requirements for the main hardware components that make up our system. The Raspberry Pi module requires 5V, rated at 3A. USB 2.0 phone charging requires 5V, rated at 1A but draws 0.66A of current. The Bluetooth module, RN42, requires 3.3V which has a low power draw and a built in DC-to-DC regulator which steps down a 5V input to 3.3V. Next is the LCD screen, which needs a minimum 10V input rated at 3A of current, but draws about ~0.7A. Touchpad and keyboard are low power hardware components and only draw under 200mA of current at 5V.

The battery LED indication circuit is also low power, drawing about ~40mA of current and can measure the capacity of any battery within the voltage range of 8V to 11.1V. The battery charging circuit with the BQ24600 Texas Instruments chip can take an input voltage of 28V rated at 10A of output charging current. But, we have set the BQ24600 chip to handle a 12V input, from a 12V AC adapter, and output a charging current of 3A towards the battery. Finally, the battery management system is a 3-Series system that manages the charging current distribution to each individual cells in order to balance them properly. Also, the BMS can properly discharge the battery without changing the capacities of the battery cells.

Below shows the overall power design for our system.

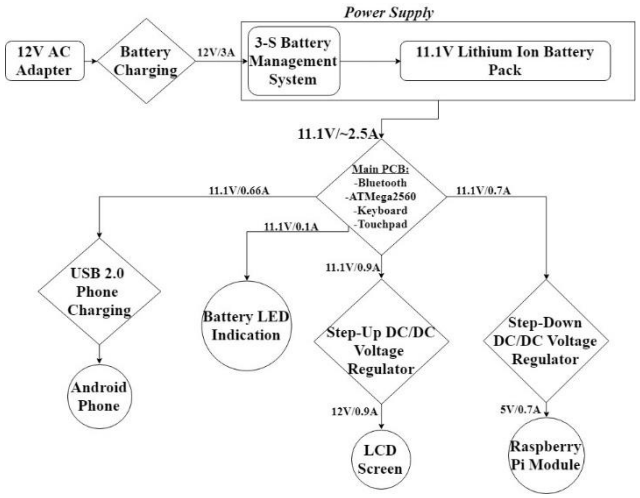


Fig. 2. Block diagram showing the power design with inputs and outputs for each circuit.

J. Laptop Battery

In this design, we chose a laptop battery that would power the laptop for about 2-3 hours before needing to be charged. We decided between lithium polymer and lithium ion batteries, comparing the two choices we came to a conclusion that lithium ion batteries hold a higher energy charge density at a much cheaper cost than Lithium polymer. But, at the expense of price and energy charge density, comes the bulkiness of the Lithium Ion batteries and adds more of a size to the laptop design. Instead of creating our own 6-cell Lithium Ion battery pack, which can be potentially dangerous due applying a very hot soldering iron to the ends of each battery which can change the capacitance of each individual cell, we went with the option of buying a ready-made laptop battery. We purchased a 6-Cell Lithium Ion battery pack rated at 11.1V and 4400mAh.

III. SOFTWARE DETAIL

To understand the complete system form a software point of view, this section will discuss in general detail the connection of the Android smart phone to the raspberry pi and the communication between the mcu, Bluetooth, and Android smart phone.

To begin with the connection of the Android smart phone to the raspberry pi first a Wi-Fi Direct connection had to be established between the android phone and the raspberry pi. Now Wi-Fi Direct uses Wi=Fi Protected Setup (WPS) for authentication that mainly consists of only two modes push button control (PBC) and Pin code for this project we have utilized the pin code authentication factor.

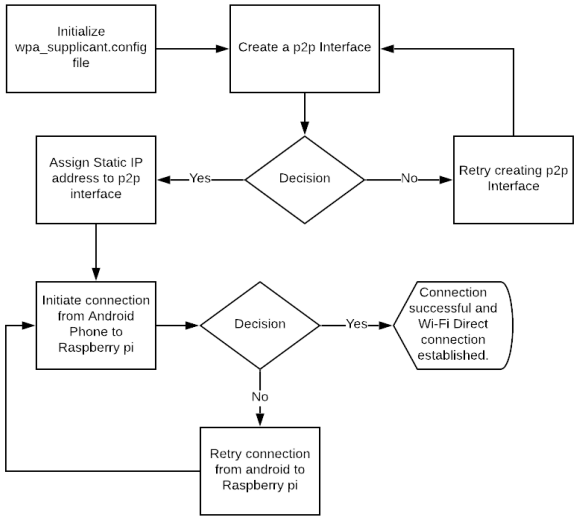


Fig. 3. Wi-Fi Direct Connection flowchart

Wi-Fi Direct is organized in groups and every group has one group owner (GO) also only the group owner is allowed to run the DHCP server to ensure that only one DHCP server is running per group. A DHCP server is a network server that automatically assigns IP addresses, default

gateways, and other network parameters to client devices, in this case the android smart phone is the client device and the raspberry pi is the group owner. Furthermore then the wpa_supplicant.config file is accessed and under the device name "p2p_go_ht40=1" this is added in order to support 802.11 for the group owner. Next the p2p interface is assigned a static IP address and the DHCP server is enabled on it which also assigns IP addresses to the client that requests a connection to a group owner. The figure above depicts a better description of how the peer to peer communication is established.

Once the Wi-Fi Direct connection has been established then sockets along with RTSP (Real Time Streaming protocol) are used to get the data from the android phone to the raspberry pi. Next sockets were utilized to send data packets from the android phone to the raspberry pi. In the sockets RTSP (Real Time Streaming Protocol) is used along with UDP transfer protocol. There are several reasons for choosing UDP over TCP and they are as follows. First reason is connection, TCP is connection oriented and UDP is connectionless which means that TCP requires a connection between the server and client whereas UDP is connectionless it does not require a connection between client and server. Second is sequencing, TCP numbers each packet that is sent so that the packets can be rearranged by the recipient whereas UDP sends packets without numbering. Third was the speed TCP is slower because the packets are heavy due to header size which allows for error checking and recovery if packets are lost whereas UDP is faster because the packets are not heavy due to header size and corrupted packets are discarded and not sent again. However the main reason for choosing UDP was because it supports applications like broadcasting which is very similar to what is being done in this project. However RTSP is commonly used with TCP but in this case RTSP will be used to communicate the player to the server and then UDP will be used to get the data packets from the server to the client.

First the TCP socket will be created and then it will be bound to the server which is the android phone, then the UDP socket will be created and this will be bound to raspberry pi using the local host address and port. Now the method of delivery for the audio and video from the phone to the raspberry pi will be Unicast delivery which uses RTP (Real Time Protocol) over UDP transport protocol. In this method the player (h.264) that is used to display the streamed information establishes a control connection to the server using RTSP (Real Time Streaming Protocol). The player is started with an "rtsp://URL" this URL includes the server IP address and the stream id. After some back and forth communication between the player and the server, during which the server sends the client an SDP file describing the stream, the server begins sending video to the client over UDP. Now since the packets are being sent individually over UDP a GUI (Graphic user Interface) is initialized which launches the player with the media stream in it.

The software we are going to use to simulate a windows like environment on our laptop through the android operating system is called "Sentio Desktop". It is free and available through the Google play app store from a Los Angeles based company named Sentio. The software is compatible with any android device regardless of manufacturer. The current version of software available in the Google play store has amassed over one million downloads.

IV. Hardware Detail

Each component in the system will be described in further detail in this section, with the exception of the microcontroller due to possible redundancy. Also, the keyboard and touchpad will be combined into one section due to how the overall system is integrated.

A. Bluetooth

In this project Bluetooth requires two wires for communication, which meant minimal knowledge was needed to properly configure the Bluetooth. The two wires are:

(1) Transmit line (Tx), which is described as the data being transmitted from the Bluetooth to the Atmega 2560. Only during configuration is when the transmit line is needed.

(2) Receiving line (Rx), is the line that receives data from the Atmega 2560 and transmits that data wirelessly to the Android phone. Furthermore, logic level shifting is needed to properly wire the Bluetooth. Since the Atmega 2560 uses 5V logic and the Bluetooth uses 3.3V logic, a voltage

divider is used to safely apply voltage to the Bluetooth on its receiving line.

The Bluetooth module has factory defaults which sets the baud rate to 115200, 8 bits, no parity, and 1 stop bit. In order to properly configure the Bluetooth module for the purpose of this project, settings need to be changed. Luckily the RN42 has AT commands that makes it relatively simple to configure, as long as a proper program is written. The RN42 also has two status pins that help identify what state the Bluetooth chip is in, through LEDs. One indicates the connection state (ON/OFF) and the other toggles at two different frequencies depending if the Bluetooth is waiting for connection or if it's in command mode. There are two modes that the RN42 can be in, which are command mode and data mode. Command mode is used to configure settings within the chip and data mode is used to transmit data packets via a data pipeline.

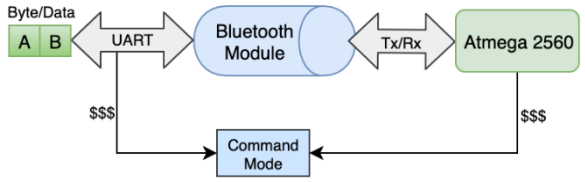


Fig. 4. A visual diagram of how command mode and data mode flow in a Bluetooth system.

Nevertheless, the overall system of the Bluetooth and how it sends keyboard and touchpad signals will be described in the two following sub sections.

B. Synaptics Touchpad and Asus G50V Keyboard

The Synaptics touchpad was properly configured using trial and error. Essentially, the touchpad had no schematics indicating which pins were power, ground, clock, or data. Due to prior experience, ground was found as the biggest plane on the touchpad. Power was a little tricky, however it was found at the node that connected the capacitor to ground at the input. The Synaptics touchpad came with an extra connector that had LEDs which illuminated when power was being received, indicating the touchpad was powered correctly. The data and clock lines required extensive research and a lot of trial and error to find the correct two lines.

Communication is allowed to begin or transmit when both lines are high also known as "idle". The touchpad generates the clock signal, but the host has total control over the clock and can pull the clock low to inhibit communication. Next the host can pull data low and releases clock, which is the "Request-to-Send" state.

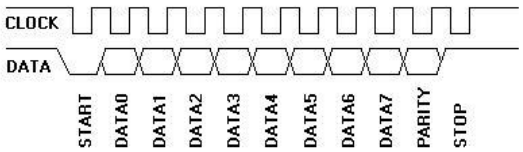


Fig. 5. A diagram of how clock and data work to send data from the touchpad.

As seen in Fig. 5 the packets being sent contain 11 bits. 1 start bit, 8 data bits, 1 parity bit, and 1 stop bit.

Next, the keyboard is configured using a matrix format. The ribbon cable from the keyboard has 24 pins, which means there are pins allocated to being inputs and outputs. In this case, 8 pins are used for inputs and 16 for outputs. When a button is pressed the input goes high and a key is sent. The overall programming of the keyboard will be described in more detail in the Software Detail section.

Now let's see how the Bluetooth, touchpad, and keyboard all communicate to make a system. The Atmega 2560 is at the heart of operation and the essential function of communicating to the Bluetooth module.

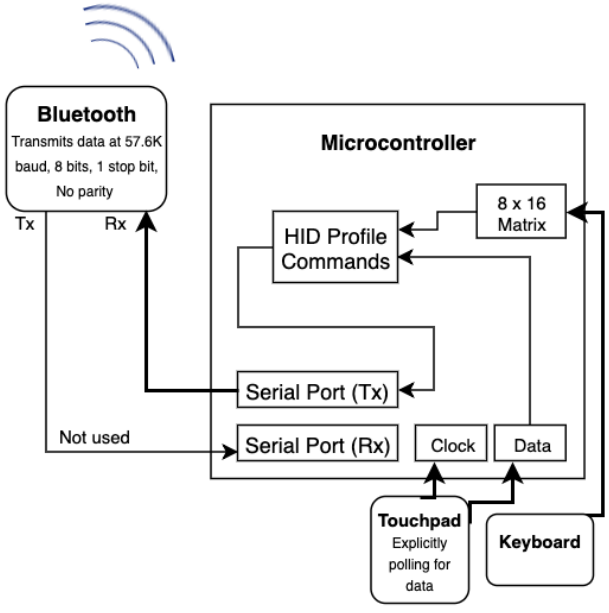


Fig. 6. An overall system of how Bluetooth works with the Keyboard and Touchpad.

C. USB 2.0 Phone Charging

As mentioned in a previous section, we used the LM7805 linear voltage regulator from Texas Instruments to power our USB 2.0 phone charging circuit. This circuit comprised of an input ranging from 7V to 35V, since our power supply is an 11.1V battery our input to this USB circuit will be at an 11.1V max input. This circuit will output 660mA of charging current to the Android phone. For the minimum threshold of 1.2V between the data lines of the USB 2.0, we used a voltage divider from the output of the LM7805 chip, which has a 5V output, to divide the voltage in half to 2.5V between the data lines. By having a data line voltage greater than 1.2V, we have access to a faster mode of charging for Android devices rather than having a minimum of 500mA of slow charging. If the Android phone were to be at about 50% battery, the USB 2.0 charging can have the phone charged up to 100% in about 2 hours.

D. Battery Charging

As mentioned in a previous section, we used the BQ24600 Li-Ion battery charger chip from Texas Instruments. This chip can have an input from 5V to 28V and up to a 10A charge current. But, we are using a 12V input for the circuit and have set the output current to 3A using a resistor divider for the VFB pin on the BQ24600 chip. We used the following equation to set the value for the resistor divider:

$$V_{BAT} = 2.1V * [1 + R2/R1]$$

Using R1 as 100kΩ and V_{BAT} as 11.1V, then solving for R2 with a value of 430kΩ. We also set the charging current to 3A by using the following equation for current regulation:

$$I_{CHARGE} = V_{ISET} / 20*R_{SR}$$

Using I_{CHARGE} as the value we want as 3A and a default value for the sense resistance of 10mΩ. We get a value of 0.6V for V_{ISET} which falls within the operating range of -0.3V to 3.3V.

E. Battery LED Indication

As mentioned in a previous section, we used the LM3914 Dot/Bar Display LED Driver by Texas Instruments. This chip can drive a total of 10 LEDs at the same time, can also be chained with other LM3914 chips to drive more than 10 LEDs but for our application 10 LEDs is ideal. Each LED indicates a voltage level of 0.1V to 0.2V, so at max capacity all 10 LEDs are active. For example, when 8 of the 10 LEDs are on, this tells us that the battery has lost about 0.4V of capacity. Moreover, the LM3914 chip was configured with a 50kΩ Potentiometer, as mentioned in the datasheet we used this potentiometer to set our LEDs to indicate whether or not our battery of 11.1V was at full charge or needing to be charged. We also have an integrated switch at the input to not draw unnecessary current whenever you don't need to view the battery voltage level at any given time.

V. WIRELESS TECHNOLOGY

This section will cover the wireless technologies that are being implemented into our project along with the ones that were researched.

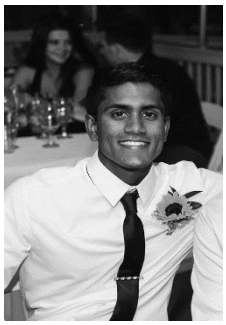
To achieve screen mirroring only two wireless technologies were researched which are Miracast and DLNA. The best option to utilize in our project was Miracast. DLNA (Digital Live Network Alliance) is also one of the powerful standards for wireless screen mirroring, the only issue is that it was introduced by Sony so it tends to produce the best output with Sony devices. Furthermore DLNA operates using a client-server model which means that the server is the device that is streaming and the client is the device that is receiving the stream via DLNA. But the main reason DLNA was not the best option for screen mirroring is because it works only with files and does not support videos or browsers to be mirrored on the client device.

Miracast is one of the most widely available technologies for wireless screen mirroring. It implements peer to peer Wi-Fi Direct standard and allows streaming of 1080p HD videos as well as 5.1 channel surround sound. Hence Miracast was the best option to use since every Android phone supports Miracast since Android version 4.0.

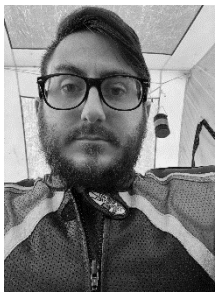
THE ENGINEERS

Kevin Ogando is a 23 year-old electrical engineer major and is seeking a career in the power industry in Orlando, Florida.

Anirudh Singh is a 23 year-old computer engineer major and is seeking a career in the software industry in Orlando, Florida.



Ameer Hakh is a 23 year old electrical engineer and loves electronics. Will work at Irvin Technologies Inc. in Winter Springs, Florida.



Nicholas Steele a 27-year old computer engineering student. Nicks career goals are to work in software specialized positions and possibly pursue his master's in artificial intelligence.

ACKNOWLEDGEMENTS

Our group would like to make the following acknowledgements to great leadership and helpful suggestions. We would like to acknowledge Dr. Samuel Richie, Dr. Lei Wei, and Professor Michael Young. Along with Irvin Technologies Inc. for allowing us to use their lab and tools.

REFERENCES

[1] Abusultan, Monther, et al. "Engineering Design Constraints for Mobile Wirelessly Communicating Maze Solving Robots." Montana State University, Jan. 2008, www.ece.montana.edu/seniordesign/archive/SP08/robot_comm/index_files/Constraints_Paper_Rev01.pdf.

[2] "LM2576/LM2576HV Series SIMPLE SWITCHER® 3A Step-Down Voltage Regulator." Jameco Electronics, Jameco Electronics, Aug. 2004, www.jameco.com/Jameco/Products/ProdDS/836123.pdf.

[3] "What Is a Bypass Capacitor?" Learning about Electronics, www.learningaboutelectronics.com/Articles/What-is-a-bypass-capacitor.html.