Visible Light Communication

A Major Qualifying Project Report,

completed in partial fulfillment of the requirements for

the degree of Bachelor of Science at

Worcester Polytechnic Institute

By

Shridhar Ambady

Megan Bredes

Calvin Nguyen

Submission Date: March 26, 2015

Advised by:

Professor Lifeng Lai

Abstract

With increasing demands for faster and more secure wireless communications, there is a pressing need for a new medium of wireless communication as the radio spectrum is already crowded. Visible light is a medium that can address both of these needs. It is a relatively new technology with great potential. This project was completed to develop a working visible light communication system and demonstrate the transmission capabilities of such a system. This year's team set a goal to surpass the previous year's team in transmission speed, range, and size. Of these goals, transmission speed and range were both achieved, while the transmission of a large audio file was deemed not possible based off the difficulties the team encountered while processing large amounts of data.

Acknowledgements

We express our deepest gratitude to Professor Lifeng Lai, our MQP advisor for devoting his time and effort to making this project possible. With his helpful feedback and suggestions on tackling problems that have arisen during the duration of the project, we are grateful to have him advise the MQP.

We also express our thanks to last year's team that worked on the original VLC project as well as the creators of the works that were cited in this report.

Table of Contents

Table of Figures

Table of Tables

Executive Summary

Visible light communication is a viable technology to accommodate the need for faster and better wireless communications in the coming years. The basic idea, is that instead of using traditional methods of communication over cables or radio frequencies, VLC systems send data by turning light on (logic 1) and off (logic 0). This report describes and evaluates the visible light communication system design the team created. As visible light communication technology is relatively new, the team worked on creating a prototype to test out this technology and demonstrate its possible capabilities.

Using last year's system design as a foundation, the team developed a successor with improved transmission specifications. The first part of the process is preparing a file or string of bytes for transmission. In order to synchronize the transmitter and receiver, the system divides the data into units called "frames", each starting with a preamble to let the receiver know that a transmission has started. The transmitter takes a file, breaks into frames, and inserts preamble sequences before each frame. Then it sends the modified file to a microcontroller unit (MCU) over the serial port. The MCU controls the gate of a transistor based on the data it receives, switching an array of LEDs on when it sees a 1 and turning it off when it sees a 0.

This light is picked up by an array of photodiodes on the receiver side. This signal is amplified and filtered to produce a clean signal as similar as possible to what was output by the transmitter MCU. This signal is then sampled by the MCU on the receiver end. Each bit sent by the transmitter is sampled 16 times, and the receiver determines whether it's a 0 or 1 based on whichever bit appears more in that 16-bit section (i.e. 14 1s and 2 0s are interpreted as a 1). This data is sent to the computer through a serial connection to be processed by a Matlab script. The script does the downsampling and converts the bits into meaningful symbols, either text or audio. At this point, the transmission is complete.

However, in our implementation, this success was only apparent on the transmission end of the system. The audio was transmitted flawlessly but the receiver was not able to convert the captured signal back into proper form due to problems in the ADC's of the microcontroller. Sampling errors accumulated to the point where they could not be filtered out by the downsampling error correction, resulting in a meaningless output.

By creating a visible light communication system, the team hopes that future students interested in this technology will be able to continue developing even better systems to study this promising new communication technology

1 Introduction

While the radio spectrum is limited, the demand for wireless data transmission keeps increasing. There is a pressing need for new kinds of wireless communication systems. Recently, visible light communication (VLC) has been proposed as an alternative means of wireless communication. The idea is to modulate LEDs transmitting electromagnetic waves in the visible light frequencies to communicate between devices within the same room.

1.1 Motivations

According to Cisco's Global IP Traffic Forecast, by 2018 there will be 21 billion networked devices, up from 12 billion in 2013. They have also predicted that the annual global internet traffic will exceed one zettabyte (10^{21} or 1 billion terabytes) by 2016, and will be well beyond that by 2018 [1]. With approximately half of these devices will be mobile, meaning they stay connected through wireless transmission ("Visual Networking Index"). Current systems use the radio frequency spectrum. Unfortunately, the radio frequency band is overcrowded and cannot keep up with the demand.

Figure 1 on the next page shows the current radio frequency allocation in the US as of 2003, which clearly shows that the spectrum is overcrowded.

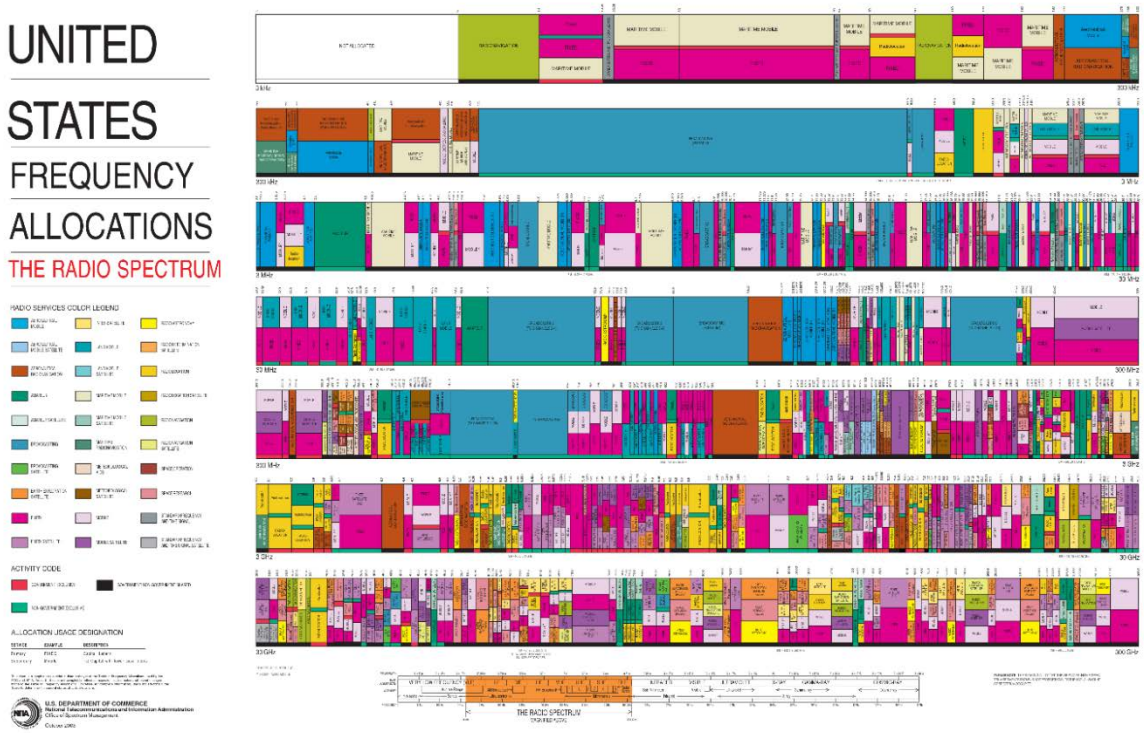

Figure 1: Frequency Allocations in the United States [2]

Not only are most of the frequencies reserved, but also the cost to reserve a free spectrum can be huge. In the US, it requires a lot of paperwork to obtain licenses, as well as studies and research by engineers to determine the feasibility and interference caused by using that spectrum. The cost for the license alone can be thousands of dollars, and it needs to be renewed yearly to ensure the frequency stays used. The fees vary wildly depending on the amount of revenue the company makes. On top of this, conducting studies necessary to show that use of the frequency in a given area that does not cause interference can create additional costs [3].

However, much of this clutter can be moved to a new system. Visible Light Communication (VLC) can be used for communication in the same room. The concept is to use a transmitter that modulates LEDs emitting light in the visible spectrum, and a receiver that captures the signal through photodiodes and processes the information. These LEDs would be blinking much faster than the human eye can detect, making it non-intrusive and unnoticeable. Rooms could have systems built into the ceiling that act as a router so devices can connect to the internet. In situations where privacy is key, visible light communication would act as a means of wireless transmission that is completely safe from being captured outside the room; data could only be intercepted from a device within the central

transmitter's line of sight. Because of this fundamental difference in how this device works versus other forms of wireless transmission, much of the costs associated with even reserving a frequency can be dismissed and the overhead becomes much lower.

No matter how great the ambition, every project needs a starting point. In 2013, a team of WPI students headed by Professor Lifeng Lai began working on a system that accomplishes the goal of transmitting data using visible light. They set up a prototype that operated at a low transmission rate of 500bits/s and was capable of sending small text messages [4]. This milestone has now motivated our group to move this system to the next step: transmitting audio.

1.2 Visible Light Communications

Visible Light Communications is essentially communication by means of optical light. It falls under the category of free-space optical communications. Transmitting data via light is achieved by having the light source flicker on and off to represent a logic high and logic low signal respectively [5]. A receiver (either photodiodes or a digital camera) will detect the light coming from the transmitter and will interpret the signal. When the receiver detects light, it is represented as a logic high and when it detects no light at all from the transmitter, it is represented as a logic low. By turning the light on and off, the transmitter can transmit 0s and 1s. This is the simplest method that visual light can be used for digital communication. Varying levels of light between on and off could allow for the transfer of more than one bit of information.

Any light technically could be used to transfer data but what matters the most is the brightness and the frequency of the light at which it modulates. The data rate of the transmission will depend on how fast the lights can turn on and off. LEDs are a popular choice for VLC communication as they can be switched on and off at a very high speed. Fluorescent lights used indoors can also be used as they flicker at a speed that is fast enough that the human eye cannot see. There is one issue with fluorescent lights however. While they could be used for communications, they can only do so at relatively low frequencies, due to the fact that fluorescent bulbs cannot be turned on and off at high speed. The resulting transmission rate would be approximately 10 kbps. This rate is not high enough to support the transmission of data such as video or audio, while LEDs, have a much faster switching speed, as they are capable of providing up to 500 Mbps and possibly even more.

Currently, one major VLC system is operating within a few countries. Named the RONJA (*Reasonable Optical Near Joint Access)* system, it can transmit at speeds of 10 Mbps while transmitting light to the receiver at a distance of nearly 1 mile apart [6]. It makes use of a high brightness LED and shines it through loupe lens to increase the brightness. This allows for a longer transmitting distance. The RONJA system is a free technology project and was developed by Twibright Labs. The purpose of this system is to network neighboring houses with cross-street Ethernet access, extend the internet connection to houses close by that are not covered by ISPs by the last mile, or just provide a link layer for fast neighborhood mesh networks.

Figure 2: RONJA system attempting to transmit through heavy fog

With VLC technology improving and becoming more prominent in the world, Li-Fi, a subset of VLC is becoming more of a reality. Li-Fi is built upon VLC and is a high-speed bidirectional wireless network similar to Wi-Fi. This is achieved by having a transmitter and receiver built in each device where it transmit light and receives incoming light from another device. However, most VLC systems today are unidirectional. As time progresses, more bidirectional VLC systems will emerge.

Recently, a company named Axrtek launched a bidirectional RGB LED VLC system. Named the MOMO, it can transmit at a high bandwidth speed of 300 Mbit/s at a range of up to 25 feet [7].

1.2.1 History

The first account of transmitting information via visible light communication was back in the 1880s where Alexander Graham Bell invented the photophone. It was an analog phone system, which could

transmit voice via modulated light [8]. The brightness of the light used in the photophone is altered by changing the position of the mirror, which changes accordingly to the sound waves from the user of the phone. On the receiver side, a selenium cell with a parabolic mirror was used. The resistance of the cell would change inversely proportional to the amount of sunlight that hits the cell. The selenium cell would replace the carbon microphone and alter the current flowing through a regular telephone.

More recently, in 2003, a team at the Nakagawa Laboratory located at Keio University in Japan used LEDs to transmit data via visible light [9]. This is the first account of transmitting digital information via LEDs.

In 2010, a collaboration between researchers from Siemens and from Heinrich Hertz Institute in Berlin were able to transmit at a speed of 500 Mbit/s over a distance of 5 meters and transfer at 100 Mbit/s over an even longer distance with 5 white LEDs manufactured by Ostar [10].

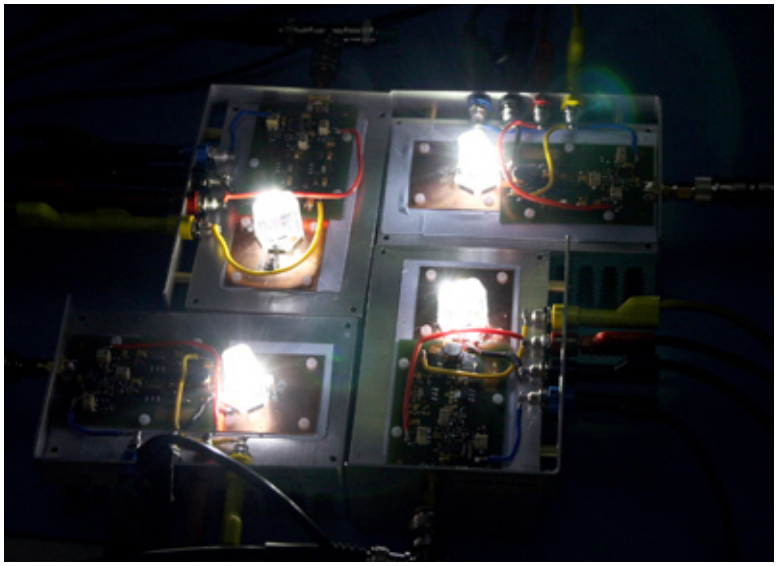

Figure 3: 500 Mbps White LEDs [9]

In July 2011, at TED Global, there was a demonstration of the D-light project, a VLC project led by Harald Haas, a professor at the University of Edinburgh [11]. The demonstration showed a HD video being transmitted from a standard LED lamp. The data rate of the VLC system was approximately 10 Mbps which is roughly the data rate of a DVD playing back [12], [13]. As time passes by, VLC is expected to grow immensely and is expected to be a possible method of providing the Internet of Things.

The previous year's team was able to design a working visible light communication transmitter and receiver. They were originally able to achieve a transmission frequency of 1 MHz but due to ADC sampling issues, there was a bottleneck for the system since the receiver could only receive up to 1 KHz. However, another issue existed within the receiver MCU meaning that the team had to set the transmitting frequency to 500 Hz just for the system to work. The team was able to transmit an 11-character message in a room with ample ambient lighting successfully. Their transmitter and receiver were 20 – 30 cm apart from each other when operating at a data rate of 500 bits per second. The data that was transmitted needed a computer to decode the data as it was encoded as a file and then decoded by using MATLAB. They were hoping to be able to transmit audio or video later on but were unable due to constraints by their current VLC configuration and the amount of time they had to work on the project. The team gave their system mobility by powering it with eight AAA 1.5V batteries (12V in total) for both the receiver and transmitter (6V for each system).

Last Year's Accomplishments	
Transmission Distance	20-30 cm
Transmission Frequency	1 kHz
Transmission Data rate	500 bps
Transmission	11 characters
Transmitter Voltage Requirements	6V
Receiver Voltage Requirements	6V

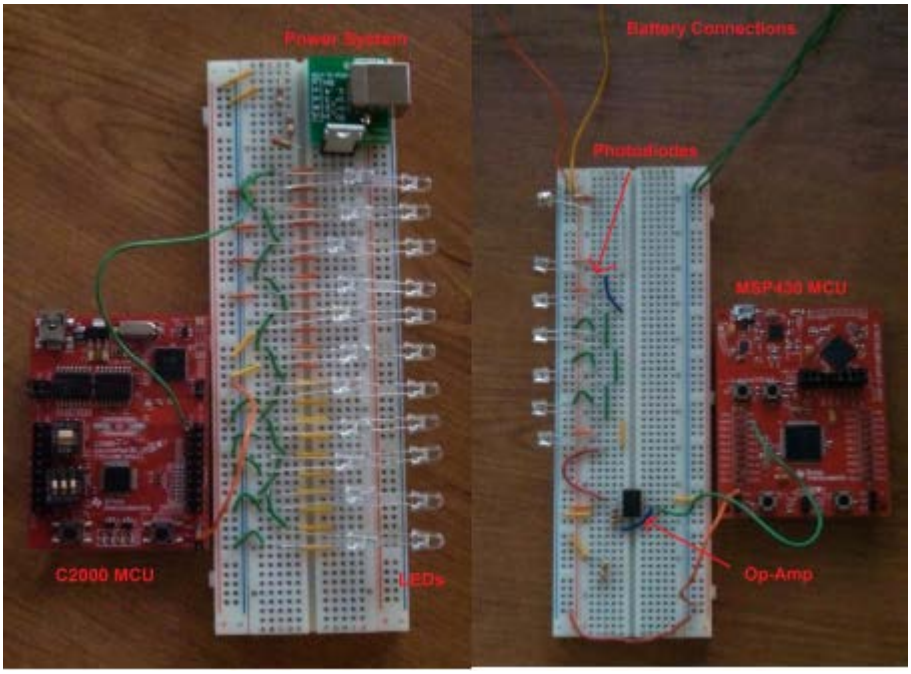

Figure 4: VLC System design from 2013

Our group will be looking back at what the first team of students achieved and how they were able to do so; we will also review the mistakes and issues that they encountered so that we can learn and create an even more powerful VLC transmitting more information such as audio. For the system to reach this point, it needs to be able to transmit data at a faster rate over a greater distance.

1.3 Possible Applications

Visible Light Communication (VLC) Systems have many possible applications. The most basic of these applications is to provide a wireless connection to all of the devices in a specific location. Every room has a number of lights in it. If basic lights were replaced with LEDs and a VLC system, then a room is being supplied with both light and internet access. For this reason, VLC technology has the potential to become quite popular for indoor local networks. While it may not necessarily replace the current wireless local area networks that exist in many buildings, it will find its purpose in devices that operate under wireless personal area networks or near field communication.

A number of the reasons people are interested in VLC systems are related to the unique qualities that come along with the fact that the system functions using visual light. One such unique quality of VLC Systems is the fact that they function in a completely different spectrum than radio frequencies. Radio Frequencies are currently highly used for everything from regular Wi-Fi to Cell Phone communication.

When the absence of interference is crucial, using radio frequencies could be dangerous. A VLC System could make delicate procedures, such as medical operations, safer by allowing them to be completely separate and safe from interference from very noise radio frequencies.

Another unique quality of VLC Systems is that by using visual light to transmit data, the data cannot be accessed by any device not in the same room as the system. Unlike Wi-Fi systems, VLC systems are contained within the room they exist in because visual light is not capable of traveling through walls the same way radio frequencies are. None of the data transmitted over a VLC System can be accessed by anyone not in the same room as the system. This unique feature has caused VLC Systems to be of great interest in groups that are concerned about keeping transmissions secure, such as personnel in the defense sector.

Another possible application for a VLC system is indoor geolocation. Currently GPS systems are mostly incapable of functioning within buildings. Since a VLC System consists of a number of different lights, it would be capable of determining the position of devices within a room through triangulation. This could be useful in situations where the location of a person inside a building that has become dangerous in some way needs to be determined. As long as the VLC System remained undamaged and the visibility in the room is for the most part clear, it would be possible to use it to locate a device that was with the person. This could make rescuing people from buildings that are on fire easier for firefighters.

VLC systems could also be embedded in traffic lights and the dashboards of vehicles. From these locations, they would be able to communicate real time traffic information to the person driving the vehicle. This would increase the usefulness of traffic lights. It would be similar to the signs over highways that alert drivers to upcoming traffic conditions, only the traffic information would be sent directly to a display screen in the car. Adding VLC Systems to Traffic Lights has the potential to make the Traffic System more efficient and more useful to the people who are using it.

1.3.1 Drawbacks of VLC

While there are a large number of advantages to the use of a VLC system, there are also a number of drawbacks. Of these drawbacks the most obvious is the fact that it will only work in places where there are electronic lights. Conventional Wi-Fi can function a decent distance from a house, allowing internet access while outside. A VLC system would not be able to achieve this due to the need for a lighting

system. It will generally work best in confined spaces where it is easy to ensure the whole room is being covered in the light.

Another disadvantage to VLC systems is the issue of blocking the light required to receive the signal. It would be very easy to unintentionally block the receiver of the system. This would cause the signal to be interrupted, potentially by things like a person walking through a room. While this disadvantage could be mitigated my using multiple light sources, it would still be very easy to inadvertently interrupt the connection by blocking the light sources.

Another potential problem would be seen mostly with outdoor systems, such as a traffic light system previously described. There could be a number of issues with the system functioning well depending on the weather outside. On cloudy days, it may work while on sunny days it most likely would not. On days when it is raining, snowing, or foggy the signal might be prevented from being transmitted well enough that it could be used. Even indoor systems with many windows might experience this issue.

1.4 Goals and Features

The overall goal of this project was to be able to transmitting an MP3 file across a distance of 1 meter at a speed of at least 150 kbps while aiming for possibly 250 kbps. To satisfy our goal, we needed to make sure that when we do transmit the MP3 file, it would be received correctly with little to no errors to prove that it has been transmitted successfully. These goals were part of an improvement on the design that was used last year, which was only capable of transmitting small text messages over very short distances.

This Year's Goals Vs. Last Year's Accomplishments		
	Last Year	This Year
Transmission Distance	20-30 cm	1 meter
Transmission Frequency	1 kHz	1 MHz
Transmission Data rate	500 bps	150 kbps
Transmission	11 Characters - Text	Text/MP3 File

The most basic functionality that this project should be capable of performing is the transmission of a prepared MP3 file. All of the file processing would take place on a computer connected to the

microcontroller that would be in charge of transmission. The file would then be sent to the microcontroller, which would transmit the file using the LEDs. The receiver would take in the information from the flashing LEDs. Signal processing would be performed on the microcontroller. The microcontroller would be connected to a computer, which would handle decoding the MP3 file. The file would then be played through the speakers of the computer as proof of the transmission.

Beyond the basic functionality described above, an additional feature would be added if there was more time. To make the presentation of the completed project more presentable, a microphone would be attached to the computer on the transmitter side. This microphone would be used to make a MP3 file for transmission. The purpose behind this addition was to show people that the system is capable of being used in a real time scenario.

This project was planned to be completed over the course of three terms. Each term has specifically defined goals. For A-term, our goals were to design and complete the analog portion of this project. The analog portion of the project would be considered complete when a transmitter and receiver circuit have been put together and a slow square wave could be observed on the receiver end while the transmitter is functioning. The functionality of the receiver would be observed by adding an LED to the receiver circuit that would blink based on the output of the photodiodes. Additionally, higher speed transmissions would be tested and observed using an oscilloscope. The system would be deemed functional when both of these types of transmissions can be observed.

For B-term, our goal was to create a series of digital signal processing code that would be reasonably effective and decoding the messages we are sending. This code would then be tested first to see if it is capable of receiving text messages. Once we were capable of sending text messages over the desired distance of 1 meter, the system would be tested with MP3 files and if given more time, streaming audio would possibly be implemented.

The goals for C-term was to thoroughly test the design that has been implemented and then present the final product. This would involve testing in a variety of different lighting conditions to see how the conditions would affect the ability to receive transmission. It would also involve sending different quality/size audio files to see how high quality of an audio file could be received without any loss of quality.

2 Design Approach

When designing and creating a VLC System, a number of steps are required. Using last year's design as a guide, the team selected similar components for this year's design. This process of component selection would be repeated as needed for every additional component that was to be selected. Once the initial components were selected, the circuit for both the receiver and transmitter were designed and tested in simulations. After that, the team would order the components, then build, and test the circuits. Once the circuits were shown to function correctly when observed with an oscilloscope, they were connected to their respective microcontroller, which connects to the PC, and the digital portion of this project would be addressed. Overall, the components for the final design cost approximately fifty dollars.

2.1 Component Selection

Once the team has a rough idea of what the VLC system would be like, components for the circuit were tested before a final selection is made. The selection process is comprised of a value analysis to place emphasis on certain desirable features of each component so that the best component of its group for the circuit could be determined. As the circuit was tested, it was expanded upon to fix problems and allow for greater functionality. A few of the components that were added also needed to be selected through analysis.

2.1.1 Analog Parts

The analog components consist of several but all equally important parts for the analog portion of the system. This includes the LEDs, the transistors, the photodiodes, op-amps, and batteries. The process used to select these components is described in the following sections.

LEDs

The purpose of the LEDs in the VLC system is to provide light to be used to transmit data. LEDs accomplish this by turning on which represents a logic 1 and turning off which represents a logic 0.

In order to be successful in attaining our goals, we needed to make sure that the LEDs we selected were bright and can switch at a high frequency. Without LEDs capable of those features, the transmitting

distance and data rate would be small. This would cause the transmission of audio to be difficult, especially with ambient light present.

In order to choose the most optimal LEDs, the team considered the important factors shown in *Table 1* below. The team organized the importance of each feature in a table to aid in the selection of the best LED. For the VLC project, there are two goals: to transmit CD quality audio from the transmitter to the receiver and have it transmitted across one meter. With those two goals in mind, the brightness and frequency speed are both important to us when choosing a LED.

Table 1: Important Selection Requirements for LEDs

Category	Importance	Desirable	Undesirable
Brightness	1	> 10000 mcd	< 1000 mcd
Frequency Speed	1	> 1 MHz	< 100 KHz
Price (Unit Cost)	3	Less than $0.25	More than $1
Color	2		

A high level of brightness is preferable so that data can be transmitted reliably under ambient light and across larger distances. The frequency speed is also crucial. In order to transmit data quickly enough, the LEDs have to turn on and off quickly.

A typical song from a CD (16-bit resolution) has a frequency of 44.1 KHz. By multiplying the sampling frequency by 16 bits, we get 705,600 bits per second or 705.6 kbps. Having a high frequency LED is very important if transmitting uncompressed music via light. A transmission rate of at least 1 Mbps would be needed to accomplish that.

Fortunately, MP3 files are compressed audio files. This results in a lower transmission rate requirement. A 3 Mb MP3 audio file is approximately 25,165,824 bits long. Transmitting a file that big with a transmission rate of 1 Mbps would take approximately 25 seconds. While a slower transmission rate can be used, the amount of time to transmit the data would take longer. This allows for a larger range of components to be considered since the high-speed transmission is not an absolute requirement of the system.

Table 2: LED Comparisons

Part Number	Brightness	Viewing Angle	Frequency Supported	Price	Color	Max Current
YSL-R542G5C-A14	10000 - 13000 mcd	10°	> 1 MHz	$ 1.89	Pure Green	20 mA Peak: 30mA
09855-04	200 mcd	Unlisted	Unlisted	$ 0.12	Green Plastic	20 mA
LW514	35000 mcd	15°	> 1MHz	$ 0.66	White	30mA Peak:100mA

Brightness tests were conducted by using the chosen photodiode and switching between different LEDs. They were supplied the same amount of power to see which one would induce the highest voltage in the photodiode. To test the max frequency that the LEDs could switch at, a frequency generator and photodiodes were used. The frequency generator would generate a square wave for the tests and the oscilloscope would be hooked up to the photodiodes to make sure the LEDs were flashing at the specified frequency.

From our tests, the LEDs easily modulated at high frequencies (a few MHz) while still maintaining nearly the same level of brightness. The only concern was figuring out how reliable the LEDs were when modulating at high frequencies at nearly full power. Simple testing showed that LEDs running near their peak max current at high frequencies are not a problem.

For reference, all data organized in tables dealing with different models for a specific component are highlighted in green, yellow, and red. Green means that It was the best choice based off of specifications and pricing while yellow means it is a fair choice while red means it was a unwise choice.

In *Table 2*, it was concluded that the white LEDs were the best. Last year's design also used these same LEDs. The white LEDs supported more current or are brighter than the bright green LEDs. With our chosen LEDs, our next task was to determine the appropriate number of LEDs for our transmitter circuit. We decided to use half of what last year team has used which was 20 white LEDs. With less LEDs, more current would be provided to each LED. We plan to supply 10 LEDs with approximately 300 mA of current. That is equivalent to 30 mA running through each LED using Kirchhoff's Current Law. If

necessary, we can increase the amount of current flowing through the LEDs to increase the transmitting distance. Even if the current exceeded the max continuous current rating, the LEDs should still be able to handle as the LEDs are flickering instead of continuously staying on. With that, we expect LEDs to have a higher maximum current when modulating as the strain is less compared to staying completely on.

Pricing for the LEDs was also important but not as important as the frequency speed and brightness levels are. Cheap LEDs are a plus as the team intends to buy multiple LEDs and connect them in parallel. This would increase the overall viewing angle of our transmitter increasing the chances of the data being detected and transmitted. The transmitting range would also increase. Lastly, color is important. Color was not as important as the rest but choosing the right color could affect how well the transmitter would perform with respect to the receiver. Transmitting a white light could result in issues, as the ambient lighting is also the same color.

Photodiodes

When selecting the photodiodes, a number of features needs to be taken into account. These features are listed in *Table 3*.

Table 3: Important Selection Requirements for the Photodiode

Category	Importance	Desirable	Undesirable
Response Time	1	< 10 ns	> 100 ns
Wavelength Range	3	Difference of 700 nm or more	Difference of 300 nm or less
Price	2	Less than $0.50	More than $1
Field of Vision	4	> 30°	< 10°

Each of the features was rated for its importance. The most important feature was the response time. If the photodiode was incapable of detecting the flashing of the LED fast enough, then it would be incapable of meeting our design requirement. A response time of less than 10 ns is desirable as a shorter response time means that the photodiodes can react faster to each bit that is transmitted. With a shorter response time, the photodiodes can support faster transmission rates. With a transmission speed of 1 MHz, the receiver would need to be accurate and be able to account for each bit. A longer

response time would lead to a higher chance of inaccuracies when receiving data. When the LEDs are transmitting at a frequency of 1 MHz, a response time of 10 ns or less would not have a huge effect on the signal being received.

The second most important feature was price. Pricing is important because the intended design would utilize a large number of photodiodes strung together in parallel. This design feature was chosen due to the very low current and small viewing angle of each photodiode. Connecting multiple photodiodes together would alleviate this problem so cheap photodiodes are strongly desirable. The third important feature is the range of wavelengths that the photodiode can detect. The larger the range, the easier the light of the selected LEDs would be able to pick up. Field of vision was also chosen as another important feature. A viewing angle that was too narrow might be incapable of identifying light from the LEDs. Knowing what the desired features were allowed a scale of what was suitable and undesirable to be produced. This scale was then used to categorize all of the photodiodes that were researched.

Table 4: Photodiode Comparison

Part Number	Response Time (ns)	Wavelength Range (nm)	Price	Field of Vision	Surface Mount	Photocurrent
SFH 203	5	400-1100	$ 0.42	40°	N	50uA
SFH 213	5	400-1100	$ 0.54	20°	N	135uA
TEFD4300	100	350-1120	$ 0.70	40°	N	17uA
OP993	5	600-1100	$ 0.74	118°	N	15uA
SFH2701	2	400-1050	$ 0.88	120°	Y	1.4uA
BPV10NF	2.5	790-1050	$ 0.97	40°	N	60uA
HSDL-4400	7.5	770-1100	$ 0.48	110°	Y	1.6uA
QSD2030F	5	700-1100	$ 0.52	40°	N	25uA
OPF482	1	500-1100	$ 13.79	Small	N	Unlisted
PD204-6B	6	840-1100	$ 0.43	Unlisted	N	3uA

As seen in Table 4, the SFH 203 photodiode turned out to be most suitable photodiode based on specifications and pricing. It should be noted that one consideration was added and that was whether the component was a surface mount or not. Surface mount photodiodes would have been highly difficult to work with and anything that was only available as a surface mount component was immediately rejected.

Despite being the most suitable choice initially, some basic testing ended up showing that the SFH 213 was the better photodiode for this system. Its significantly higher photocurrent resulted in a clearer response on the receiver output. This knowledge led to the use of 5 SFH 213 photodiodes were used in the VLC system.

Op-Amp

With the photodiodes producing a low amount of current and voltage when exposed to light, an operational amplifier is needed to amplify the signal. With op-amps, a low power signal could be converted into a higher power signal suitable for the analog to digital converter.

When selecting a suitable op-amp for our VLC system, the minimum input voltage and the amount of amplification that the op amp can provide must be considered. The team chose last year's op amp, AD848JN, as a starting place. Because of its proven ability to work well, the team has stuck with the decision throughout the duration of the project.

High-Pass Filter

With an op-amp amplifying the signal coming from the photodiodes, we needed to consider that the noise along with the signal would also be amplified. To resolve this issue, employing an active high-pass filter to remove the noise before being amplified would create a clean signal. The amplified signal would then be a clean signal, which would be interpreted by the ADC easily, leading to accurate data. Capacitors and resistors were used to design the high-pass filter. Through testing, we determined that a 100pF capacitor connected to a 300kΩ resistor would do the job.

BJT/Transistor

The purpose of the transistor in our transmitter analog system would be used as a switch controlled by the MCU. The transistor would connect an external power source to the LEDs instead of having the MCU drive the LEDs directly. That way, more current/power would flow through the LEDs than what the MCU could output. This would result in brighter LEDs, achieving farther transmitting distance. The MCU would

control the transistor and when the transistor receives a logic high, or 3.3V signal at the base, the transistor would turn on meaning it would allow current to go through from the external power source to the LEDs to ground. When it is off, no current would travel.

When choosing the transistor, we needed to make sure that it could switch on and off properly with a logic voltage of 3.3V and that it supported high enough frequencies (at least 1 MHz). Last year's team used a logic level gate MOSFET so we decided to first use a MOSFET. The IRF520 MOSFET was first used but later we found out that the MOSFET actually needed 10V to properly turn on completely (unlimited current passing through). The MCU only provides 3.3V so the MOSFET failed to work properly with our system but did allow some current to pass through. From this point on, the team had to either buy the MOSFET that last year's team used and endure the issues that last year's team had with their MOSFET (overheating issues) or try using BJTs. With BJTs, the cutoff voltage is 0.7V, which was more than enough to work with.

For BJTs, there was the 2N3904 and then there was the 2N2222A. We have tested out the 2N3904 transistor and while it was proven to work, we realized that this BJT was not suitable for our purposes as we plan to provide over 300mA of current and possibly more to the LEDs. We saw that the 2N2222A BJT was the most suitable transistor for the job. It is similar to the 2N3904 but has a higher maximum continuous current rating, which is 600 mA and has similar switching frequencies as the 2N3904. The frequency is supposedly around 150 MHz, which is more than enough for what we need for our project.

Batteries

Batteries were used to provide additional power to the system. The transmitter used an 8x AA battery holder, which is capable of supplying 11-12V to the 10 LEDs. This additional power was needed for the transmitter to ensure that the LEDs received enough current to transmit brightly. A benefit of using batteries is to allow the transmitter to move around without relying on a cumbersome power supply unit.

2.1.2 Digital Parts

The digital parts of our VLC system are considered as the brains while the analog portion can be thought of as the body. Both are equally important and choosing the optimal parts is important in having the system perform at its best.

MCU

With light being transmitted by the LEDs and being received by the photodiodes, the raw analog signal itself is not of much use. A DAC is needed to take the binary values that the data stream is represented in and turn it into a signal that can control the LEDs' flickering rate. On the other side of the system, an ADC is needed to convert the photocurrent back into a binary format that can be digitally processed. All of this conversion and processing can be found on microcontrollers and development boards. The microcontroller not only does some conversion and processing, but is also the interface between the analog section and the computer, which further analyzes the data using Matlab.

Cost Analysis

The transmitter/receiver microcontroller is arguably one of the biggest bottlenecks in the system. While the analog circuitry may introduce quite a bit of noise and be difficult to tune properly, a fast digital board can get around this by digitally processing the data and outputting accurate results to the computer for Matlab to finish processing. The problem is that while there are many boards that are more than fast enough and have the necessary features, they come at too high of a cost. The boards that are within the price range for the system we would like to create are less capable than the ideal setup.

Before assessing which board to use, we identified key requirements for the MCU, and gave them a weight. However, the transmitter and receiver have different purposes and so their criteria vary between each other.

Transmitter

The transmitter has a couple of key roles:

- Stream in data from the computer
- Convert data from 1s and 0s to a logic high and logic low voltage, respectively
- Control switching of the LED array

In order to best satisfy these purposes, we identified what aspects of a microcontroller we would be most interested in, and assigned them numerical weights. The ranking was organized in the Table 5.

Table 5: Important Selection Requirements for Transmitter MCU

Category	Importance	Desirable	Undesirable
Interface	1	As many different types of I/O ports as possible	Very limited or not having USB the very least
Cost	2	Less than $20	More than $40
Clock Speed	3	> 5 MHz	< 1 MHz
Ease of Use	4	Libraries, Documentation, Help Forums	Rarely used in the real environment

Initially, we felt clock speed was the most important. Our goal was to transmit audio. At a sampling rate of 44.1 kHz, and 16 bit of resolution, this translates to 705.6 kbps for lossless audio. If we were to use a compressed audio format instead, we could reduce the transmission rate to about 256 kbps, the nominal rate for the MP3 format. Because these speeds are comparable to dial-up internet, we figured that there would be quite a bit of processing necessary beforehand and a high clock speed is needed. We soon discovered that even $10 MCUs had enough clock speed and that other criteria would have more differentiation between them and be more important in the consideration.

The available interfaces were very important in choosing a board. Because memory is very limited on microcontrollers, the data had to be streamed from a computer. For this reason, it is important to have a proper interface that can transfer the data of at least 250 kbps. Simply using digital out pins would not suffice. For this reason, we weighted the available interfaces as the most important feature.

The next most important factor to consider was the cost. The end goal of VLC research is to create a system that can be widely used for mobile communication, which means that costs should be cut wherever possible to make it more affordable for customers. While cost could not be used to cut out necessary features, it could be used to pick between two boards that pass the bare minimum.

The MCU also needed to be relatively easy to use. With such a short time span in which to create the prototype, we could not afford to spend weeks memorizing register names and APIs. We needed to have a board with clean documentation and easy to use built in functions that expedite the process.

Receiver

The receiver had an entirely separate set of requirements based on its purpose:

- Sample amplified filtered photocurrent
- Perform intermediate processing
- Stream data out to computer for further analysis

The demands placed on the receiver cover a greater field of needs and are much more intensive than those placed on the transmitter are. We came up with the following rankings below.

Table 6: Important Selection Requirements for Receiver MCU

Category	Importance	Desirable	Undesirable
Interface	5	As many different types of I/O ports as possible	Very limited or not having USB the very least
Cost	4	Less than $20	More than $40
Clock Speed	2	> 5 MHz	< 1 MHz
Ease of Use	6	Libraries, Documentation, Help Forums	Rarely used in the real environment
Sample Rate/Reliability	1	> 1 Megasamples	< 500 Kilosamples
Memory	3	> 1 Mb	< 512 kB

The primary purpose of the receiver is to take the photocurrent from the analog photodiode circuitry and convert it into digital bits that can then be processed quickly using algorithms in Matlab. This means

that the ADC must have a fast enough sampling speed. We initially need to oversample the signal by a factor of 12 so that we can greatly reduce conversion errors during the transmission and receiving process. When streaming MP3 files, which have a data rate under 250 kbps, we would have to sample at 2.5 Msps at the minimum. Anything less would mean unreliably recreating the audio signal, and the system simply would not meet the goal of transmitting audio properly.

Simply sampling and passing the signal onto the computer is not useful because there are a couple calculations that can be done directly on the MCU to reduce any slowdown caused by an interface bottleneck. For this reason, the clock rate becomes more important than it was with the transmitter. Any calculations that need to be done on a given sample must be completed before the next sample is taken. Otherwise, the system would be taking in data faster than it can stream to the computer, and it would fill up the buffer quickly. To assist in making sure the buffer does not fill up, we wanted enough memory on the MCUs such that there could be some lenience in data throughput. While it would not be practical to get several megabytes of memory to store the entire audio file before sending it to the PC, a few hundred kilobytes would definitely help in performing calculations and keeping track of samples.

Cost was not as important a factor as it was with the transmitter, primarily because we were willing to pay extra for a good receiver MCU. With the transmitter, we were more likely to find plenty of low cost options that fulfilled our other needs at the minimum. With the receiver, the bare minimum sampling rate was difficult to find amongst cheaper boards. If it ended up being more expensive than we preferred to get a good sampling rate and clock speed here, we did not let it weigh down the decision too much.

After this came interface and ease of use. While interfacing is still very important for the receiver (if it cannot reasonably transmit to a computer, then the option is not viable), the sampling rate and clock speed were just that much more important. Ease of use is at the bottom because it would be nice to be able to just jump into using the microcontroller, but not completely necessary.

Picking MCU

When looking for appropriate MCUs that matched the requirements, we did not specifically try to find a transmitter and then look for an appropriate receiver; we instead looked for reasonable MCUs and then later looked at which would be the best to use as either a transmitter or receiver.

Table 7: MCU Comparison

MCU	Sample rate	Memory	Cost	Clock rate	Ease of use	Interface
C2000 Piccolo	4.6 Msps	64 KB	$17.05	60 MHz	Unreliable with sampling rate over 800 ksps.	USB, UART, I2C, SPI
STM32F401 RE	2.5 Msps	512 KB	$10.12	84 MHz	Libraries are not well documented. Varying opinions on difficulty.	USB, I2C, SPI, USART
Intel Galileo 2	2 Msps	8 MB	$60-80	400 MHz	Can be fairly complex, very feature rich and powerful	USB, USART, SPI, I2C, Ethernet
LM3S8962	500 Ksps	256 KB	$72	50 MHz	Used before in previous course, well documented and supported	USB, UART, SSI, I2C, Ethernet, CAN
Raspberry Pi	Needs Peripherals	512 MB	$25	700 MHz	Is literally a general purpose computer, can run Linux. Lots of coding overhead. Documentation for hobbyists, but not engineers	USB, I2C, UART, SPI, I2S Audio, Ethernet
Teensy 3.1	3 Msps	256 KB	$19.80	72 MHz	Can run Arduino code, has several useful libraries for audio processing. Little documentation or support	USB, I2C, UART

In terms of sample rate, it would appear that the C2000 Piccolo has the advantage. However, last year's team found that while 4.6 Msps is theoretically achievable, it cannot be reliably achieved. After 800 Ksps, the MCU can be set to continuous sampling mode. This tells the ADC that it no longer has to hold and wait between samples and samples as fast as it can. This method lets it go much faster, but it is not precise. It can hit above 4Msps, but can also be just 1 Msps. The Intel Galileo and TI LM3S have sampling rates below the necessary 2.5 Msps so those are almost immediately out of the picture for receivers. The Raspberry Pi does not have a good ADC on board, but an external peripheral can be attached. While

this would increase the cost and make it a less viable solution, it does not detract from it. The STM and Teensy both sample reliably at the minimum sampling rate needed.

For memory, the Raspberry Pi is the clear winner with 512 MB. Because it is a general-purpose computer, it has a lot more memory available to run an operating system. After that in terms of most memory, is the Galileo, which possesses quite a bit of computing power and is meant to be somewhat of a digital powerhouse. The rest of the MCUs have comparable memory units, though the C2000 has the lowest at just 64KB.

The Galileo and LM3S are both on the pricier end of the spectrum, though still affordable at around $70 each. The others are around $20, except for the STM32F401RE MCU, which is the cheapest at just barely above $10. This splits the board into three pricing categories. While the STM is not at too much of an advantage being only $10 cheaper than the next category, the Galileo and LM3S are at a clear disadvantage, sitting at over triple the cost of the previous category.

In terms of clock rates, the breakdown is very similar to memory. The Galileo and Raspberry Pi have very high clock speeds for a microcontroller, though it should be noted that the Raspberry Pi's speed is almost double that of the Galileo. The other four MCUs have similar clock speeds under 100MHz. The LM3S is the slowest and the STM is the fastest of those. These clock speeds are all generally fast enough to handle transmission to the computer and other tasks.

Ease of use, while not weighted as much as the other categories that are more technical and more easily quantifiable, remained very important. We had a very limited time span in which to complete the project, especially considering how ambitious the goals of this project were. Using a new board can involve weeks of poring over the datasheet and being acclimated to the development environment. Some boards, like the MSP430, are coded primarily through directly bit masking registers. This can mean spending a very long time figuring out simply how to initialize the board itself. Other MCUs come with very extensive libraries. While these libraries offer several useful functions that bypass manipulating register bits, it can still take a while to familiarize one's self with these functions. The C2000 is similar to the MSP430, which we were all familiar with from a previous course, and so it was automatically considered easier to use. The LM3S had also been used in a separate course and so while it would have been quite difficult to figure out how to operate it and take advantage of its many capabilities, our previous instruction in its usage gave it an advantage. The Teensy can run both C and Arduino code. Arduino code is quite high leveled and requires little understanding to code. The Teensy also had

specific audio libraries that would have helped us process and decode the incoming signals directly on the board, but the Teensy is actually entirely developed by one person, and continues to be supported by a very small team of engineers. Because of this, the documentation is limited and support essentially means asking questions on the Teensy's forums, which has a small user base. The Raspberry Pi is relatively simple to use once set up, but the problem is that there is a lot of overhead. Setting up the Raspberry Pi requires installing Linux on it and configuring the environment. Furthermore, the Raspberry Pi is meant to be more of a hobbyist developer's kit instead of being marketed at professional engineers. Documentation for it is overly obtuse and easy to understand, but does not give specifics like a lot of the electrical characteristics, or timing diagrams. The Galileo is very feature rich and is well documented with powerful functionality. Unfortunately, it would take too long to be familiarized with the board. Finally, the STM was very easy to initially pick up and be able to program simple tasks, but its functions are somewhat overly simple and it is difficult to find out how to configure finer details. The documentation includes user guides and example programs, but not much on the API. However, it does have a very large and active community with guides on how to do tasks that are more complex.

Last on the table is the interface. All of the boards support USB, UART/USART, and I2C connectivity. All of them except for the Teensy can also connect through a Serial Parallel Interface. The higher end boards, like the Galileo, have Ethernet ports. Ethernet connections are a double-edged sword in this situation. It is by far the fastest method of communication, but can be unnecessarily difficult to program and process. Fortunately, the boards that have Ethernet ports offer good support for networking functions.

In the end, we chose the STM32 F401RE because it was a good middle ground in many categories. It had a high enough clock rate, an average amount of memory, standard interfacing ports, met the minimum sample rate, was rated quite well by reviewers, and was the cheapest option. Coding for it was somewhat odd due to it being an online compiler and having little documentation on the software functions. The coding process certainly would have been better on the C2000 or the Teensy, but they do not offer the same physical features as the STM. For interfacing, the STM32 can also have an Arduino Ethernet shield plugged into it, which has its own set of libraries and documentation.

2.2 Analog Design

Once the initial components had been selected, the circuit design process had begun. In every communication system, a transmitter and a receiver are always involved. The transmitter and receiver end of this system both went through several iterations before the current final design was produced.

2.2.1 Transmitter

The purpose of a transmitter is to send data to the receiver so that the other side can process and interpret the data. In our analog design of the transmitter, we used LEDs to transmit light, which would be used to transmit data. As mentioned earlier in the paper, the LEDs would modulate on and off to transmit 1s and 0s. The source of the data that would be sent to the LEDs would be the MCU of the transmitter circuit. During the preliminary rounds of building the analog transmitter circuit, it consisted of nothing more than just 20 LEDs and a MCU. While this design worked, it failed to meet one of our goals, which was to transmit at a distance of 1 meter apart to the receiver. The amount of power being sent to the LEDs from the MCU was miniscule. The current from the MCU to the LEDs is not adequate as it was approximately 23 mA of total current coming from the MCU to about 10 LEDs directly with zero resistance. This is approximately comes out to around 2-3 mA for each LED in our system which results in little light produced. In order to alleviate this issue, another power source was needed.

We decided that some kind of relay or switch would be needed to handle the switching of the LEDs. We also decided to use the batteries from last year's team to be used as our external power source for the LEDs. The batteries used in our second design of the transmitter were four AAA batteries providing an overall voltage of 5-6V. We also had a low resistance between the batteries and the LEDs to not short circuit the LEDs. For the LEDs themselves, we decided to cut back the amount of LEDs from 20 to 10. The reason for doing so is to increase the brightness output for each LED as more current can be supplied to each LED. 20 LEDs was how many LEDs last year's team used. For the switch, we originally used the IRF520 MOSFET but then we realized that our data signal voltage from the MCU did not satisfy the MOSFET's minimum requirements. We changed the transistor to a much more suitable transistor in our third build, which was the 2N2222A BJT.

In our third and current design as shown in Figure 5 below, we decided to change a few things. For the external power source, we decided that four AAA batteries was just not enough for the 10 LEDs so we decided to make use of eight AA batteries. The benefit of using AA batteries is that the power output

would last much longer due to having a higher capacity for each battery. Furthermore, more batteries imply a higher voltage, which would lead to an increase in LED brightness as the amount of power for each LED increases. The voltage of our previous battery source is now effectively double in our current build. The resistors between the batteries and the LEDs also were changed as it used to be a higher resistance due to fear of damaging the LEDs with high current but later testing showed that the LEDs appeared to have no issues with a very low resistance. Therefore, the team ended up using two 10Ω resistors in parallel to have the current from the batteries travel through a 5Ω resistance pathway to the 10 LEDs. From our results, the current supplied from the batteries is approximately 360 mA. That would mean about 36 mA is supplied to each LED. The batteries voltage measured to be approximately 11V when the transmitter is off and when under load, the voltages drop down to 10.4V. The final power consumption for each white LED while being supplied with approximately 30 mA is estimated to be around 100 mW so for 10 LEDs, which would be 1.0 W overall.

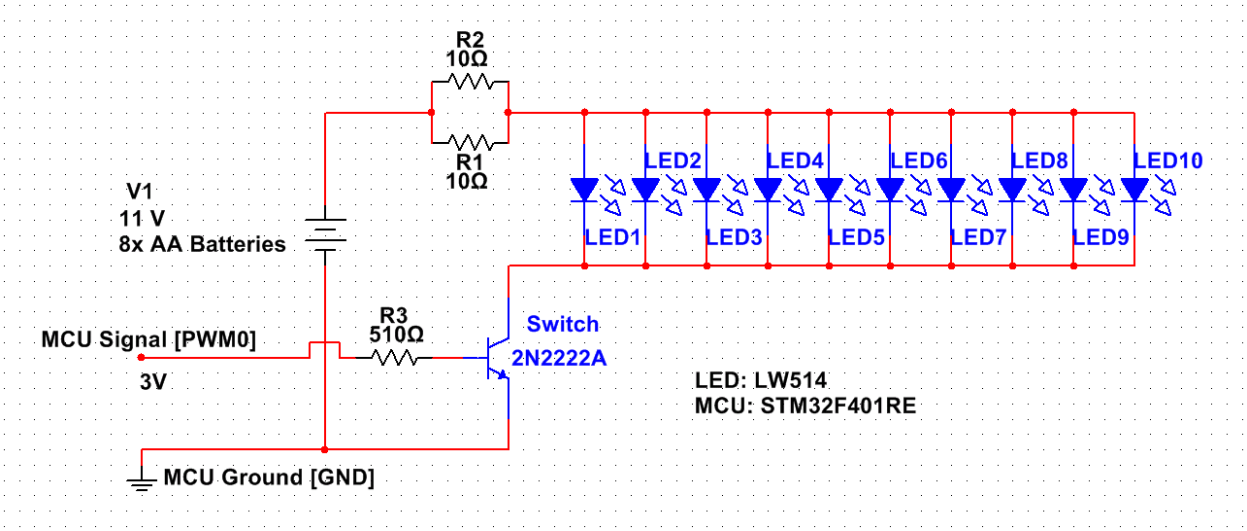

Figure 5: Transmitter Circuit Diagram

The other major change to our current design is the transistor used. Originally, a MOSFET was used but was deemed to be unsuitable. This time, we decided to switch to using a BJT. During the testing of transistors, we briefly decided to use the 2N3906 BJT; however, as mentioned in previous sections, it was not adequate. We finally decided to use the 2N2222A BJT, which is very similar to the 2N3906 BJT but is made for slightly higher power usage circuits such as ours. The 2N2222A transistor turns itself on (the resistance between the collector and emitter essentially become 0Ω) when a 0.7 V or higher signal

is at coming from the MCU PWM pin. This cutoff voltage is the main reason why we switched over from the IRF520 MOSFET to the 2N2222A BJT as the IRF520 needed 10 V to turn on completely (to lower the resistance between the drain and source to 0Ω). When the BJT turns on, the collector and emitter connect and the power from the batteries reach the LEDs. With the 2N2222A supporting high frequencies, we can turn the switch on and off quickly (up to 150 MHz) so there are no issues with the speed capability. A 510Ω resistor is used to limit the amount of current travelling from the MCU PWM data pin to the base pin of the BJT. The amount of current flowing is measured to be approximately 4.7mA. The collector current is approximately 360mA (may be lower depending on the amount of power left in the batteries). The beta of our BJT can be calculated by dividing the collector current by the base current. $\beta = \frac{I_c}{I_b}$. The beta for our configuration was approximately 76.6. The optimal beta for a 2N2222A transistor is 100 but can largely vary depending on how much base current is flowing as well as the temperature of the transistor.

With those changes in our final design of our transmitter, the LEDs can now run at a very bright level effectively while also modulating on and off quickly. These design changes helped us meet our goal of achieving a transmitting distance of 1 meter.

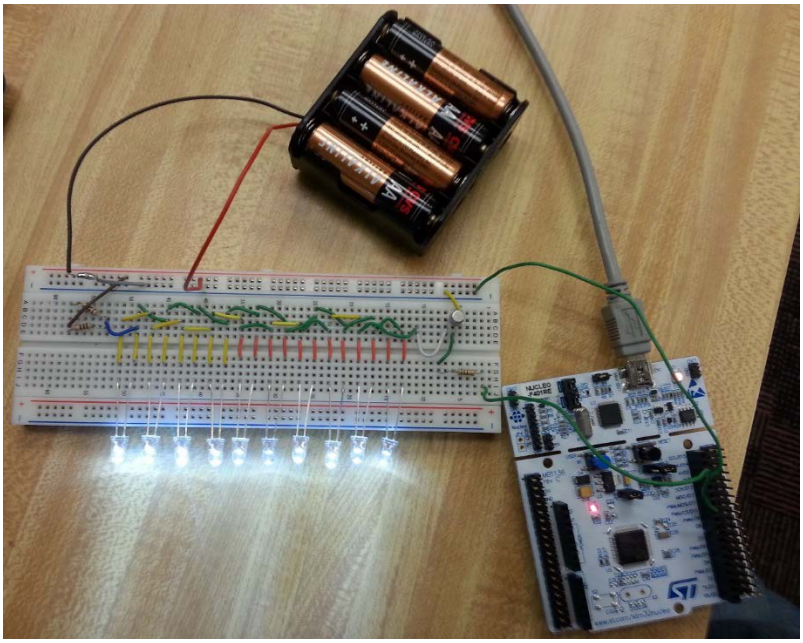

Figure 6: Transmitter - Final Design

2.2.2 Receiver

The idea behind this circuit was to get a reasonably strong photocurrent that would only intake the signal from the transmitter LEDs and output that signal to the receiver MCU. To get a strong photocurrent at up to a foot away, we needed a sizeable array of photodiodes. Matching the transmitter, the array had 10 photodiodes to capture the light. With the LEDs shining at 6 inches away, each photodiode produced 1.5 mA of photocurrent. 10 photodiodes produced 15 mA of photocurrent. Unlike the transmitter, which uses a resistor to pull a sizeable current from the voltage source through the BJT, the receiver needed a resistor to create a voltage difference with the photodiodes acting as a current source. This resulted in a square wave going from 150 mV to 300 mV, which was smaller than expected.

Initially, the idea was to amplify the signal, and then we could set a voltage cutoff based on the output and there would not be a need for much analog processing beyond that. We set up a simple non-inverting amplifier circuit to achieve that. With this circuit topology, the gain equation for a closed loop op-amp is $A_{CL} = 1 + \frac{R_2}{R_1}$. We decided upon resistor values of R_1 = 1 kΩ and R_2 = 10 kΩ to give a gain of 10.

This circuit did accomplish the goal of amplifying the input voltage to get a clearer difference between a logic 0 and logic 1, but the problem was that when the output was tied to the ADC input on the MCU, the voltage going in was at a range of about 2.5V to 5V. Not only was the AC component of the signal amplified, but the DC was amplified as well.

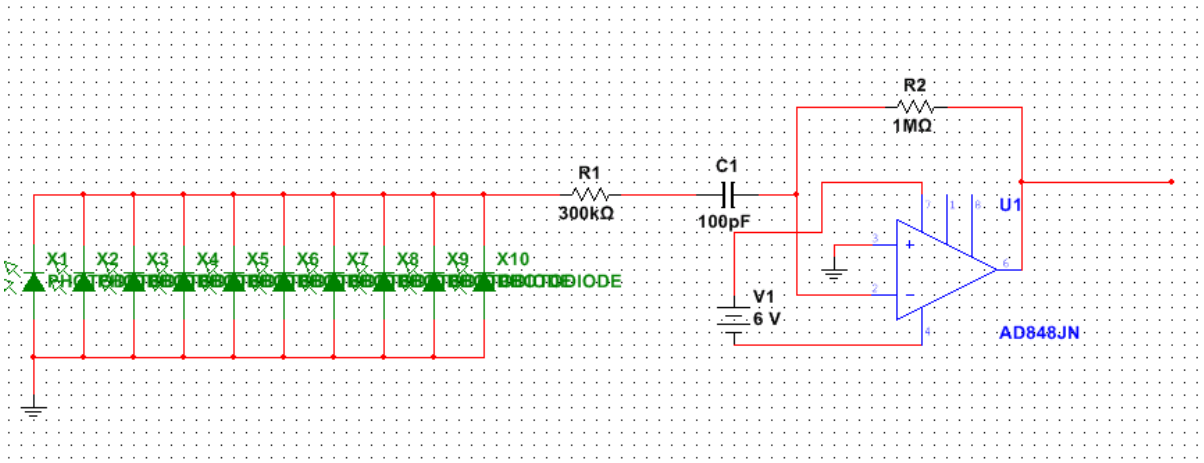

Figure 7: Old Receiver Circuit Diagram

To deal with this small voltage difference as well as the DC offset, we put this signal through an active high-pass filter. This would accomplish multiple things for our photodiode's output. For one, it would amplify the photocurrent, making it less likely for the MCU to make a quantization error. Secondly, it would remove the DC offset and allow for more amplification while not running into the problem of hitting the MCU's input ceiling of 3.3V. It is not very useful if the signal goes from 3V-7V if the MCU treats everything above 3.3V as the same. Removing the offset makes the active range more compatible with the MCU. Third, a high pass filter would filter out all ambient light. Generally, ambient light in any indoor environment would be locked to 60Hz. This noise was actually initially quite apparent in the output and had a significant effect. A high-pass filter got rid of the noise caused by ambient as well as any general white noise.

The transfer function of this filter type, a basic inverting active high-pass, can be given by the following equation: $H(j\omega) = \frac{-R_2}{R_1}\frac{j\omega * R_1 C}{1 + j * R_1 C}$, with $1/R_1 * C$ being the cutoff frequency. The feedback resistor, in relation to the series resister, determines the amplification of the system. The capacitor, combined with the series resistor, determines the cutoff frequency of the system. With the series resistor set to a value that would create a suitable input voltage (300 kΩ in this case), the capacitor and feedback resistor were chosen to match the desired amplification and cutoff frequency. We wanted to get a gain factor of ~3, and a cutoff frequency between 250 Hz and 500 Hz for the initial tests, where data would be transmitted at 1 kHz but without the 60 Hz ambient light frequencies. Thus, our feedback resistor was set to 1 MΩ and the capacitor was set to 15 nF. Theoretically, the gain should be approximately -3.33, and the frequency cutoff should be about 250 Hz.

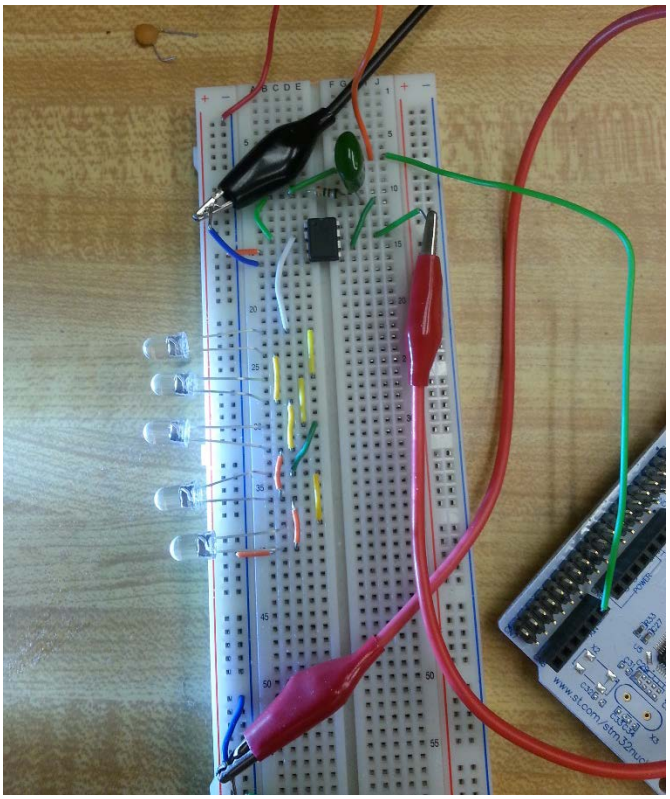

Figure 8: Receiver Final Design

However, some problems were experienced with the previous design. There were a series of inconsistencies in the output of the previous circuit. At times, it would produce strange output signals as seen in Figure 12.

Through a series of tests, it was found that this problem was mostly eliminated by changing how power was supplied to the op-amp. Introducing a duel polarity power supply set up removed the inconsistencies in the output. This is likely because a power supply can more reliably supply power than the batteries could. Additionally, a capacitor was added to the circuit output to remove the DC bias from the output. This made it easier to interpret for the MCU. The final design on the receiver circuit is shown in Figure 9 where the photodiodes shown are SFH 213 photodiodes.

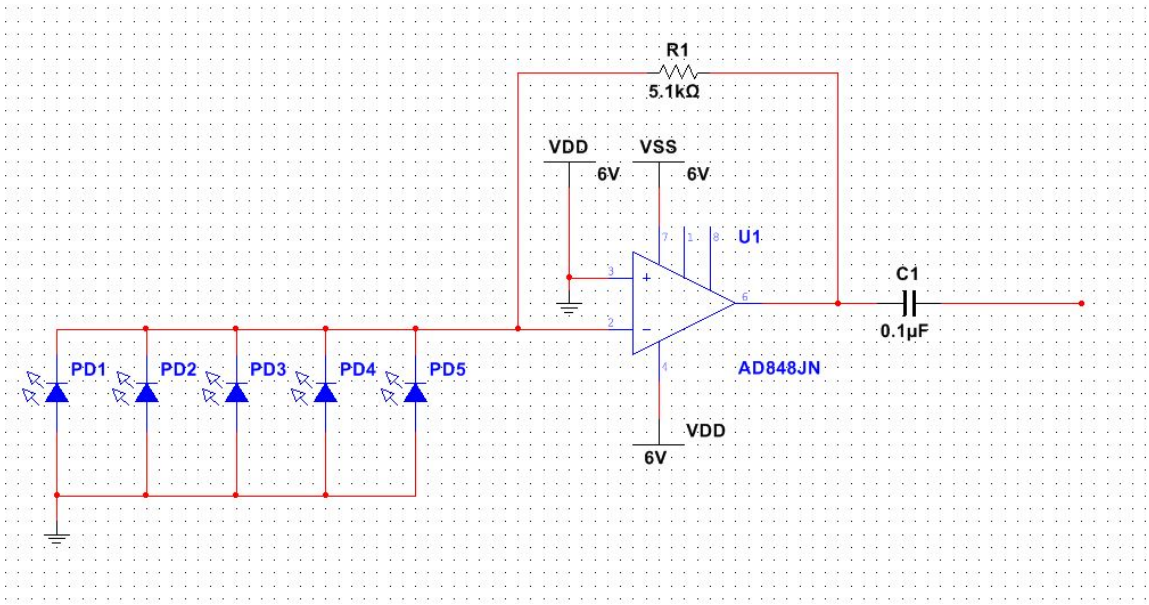

Figure 9: Receiver Circuit - Final Design

2.2.3 Design Verification

Once each component was shown to be capable of functioning correctly on its own, the system was combined. A function generator was set to generate a square wave with a moderate frequency on the transmitter end. On the receiver end, the output of the photodiodes was measured using the oscilloscope to determine if the appropriate signal was being received.

During our testing, we ran into small issues with the produced received signal coming from the photodiodes. The signal was very noisy and what happened was that the filter was configured incorrectly. The capacitor was located in the wrong spot, which caused a terrible passband and transition band. After fixing the issue, the produced signal was now clean.

This was first done in a circuit that did not include an Op-amp as shown in Figure 10.

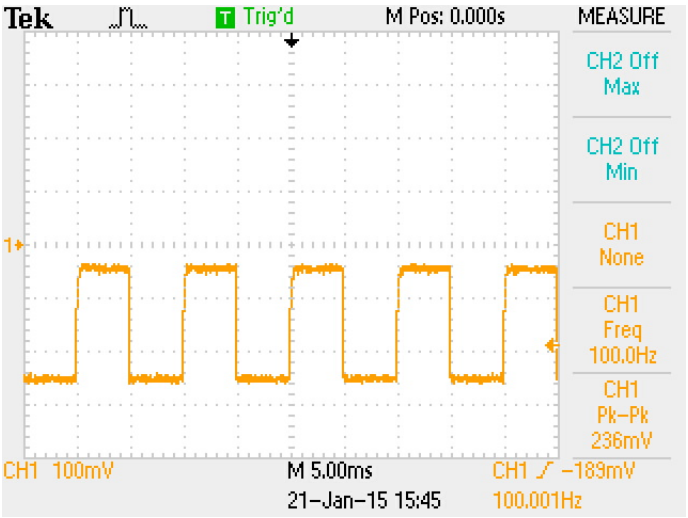

Figure 10: Photodiode Output before Op-Amp

As can be seen in the figure, the voltage difference between a logic 1 and logic 0 was very small, with the peak-to-peak value being only 236 mV. Because of this, an Op-amp was selected as described in the previous section. This op-amp is powered by a set of batteries; this allows the output of the photodiodes to be amplified so that the difference between the voltage of a logic 1 and logic 0 is clearer. As shown in Figure 11, the peak-to-peak value is now 1.4 V. This means that it is now easier to define a threshold voltage for what a logic 1 and logic 0 is.

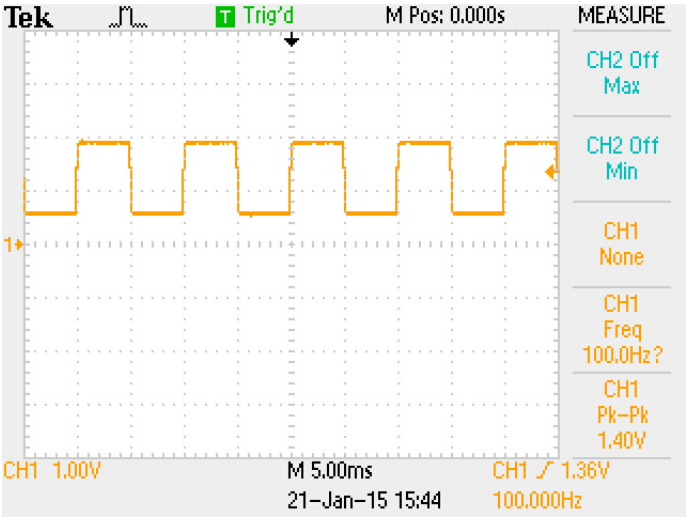

Figure 11: Photodiode Output after Op-Amp

Now that the threshold voltage can clearly be determined, the analog system can be declared successfully implemented, and the digital system could now be addressed. These results were however obtained with the original, unreliable version of the circuit. Occasionally the system would produce an output like the one shown in the figure below.

Figure 12: Bad Output

As mentioned in the previous section the design was revised. The new design ended up with a very similar output to what is shown above; however, the voltage was centered around zero now and had a range of 5V to -5V.

Figure 13: Good Output

2.3 Digital Design

For this project, the digital design focused on the code required to receive and transmit the desired data. This involved creating code for the MCU, and for the computers in Matlab and C#. The team used the online mbed compiler and made use of the mbed libraries available online to write the transmitter and receiver programs for the MCU.

2.3.1 Transmitter

For the transmitter code, there were two different styles of code that needed to be implemented. Computer code needed to be used to load a file from the computer and send it out to serially to the MCU using the program TeraTerm. Once on the MCU, the program (written in C language) transmits the series of zeros and ones obtained from the computer.

Computer Code (C#)

The computer's role is loading data onto the MCU so that it can output it to the LEDs. The original file that is to be transmitted is store on the computer's hard drive, so that needs to be streamed to the MCU. Not only does it need to be streamed, but also needs to be sent frame by frame, with a preamble at the beginning of every frame for synchronizing on the receiver's end. When we were doing initial testing, we wrote the code in Matlab. The Matlab code did not need to open a file; it just needed to be able to take a text input from the user and would send the text. Because the text was only so many characters long and under the number of bytes per frame, we only had to append the text after the preamble and send it out through a serial port.

This served well for the purposes of testing the receiver and seeing if it could properly sense the light from the LEDs and convert that to text. The next step was improving the transmitter so it could go beyond simple text messages and send audio files. While at first we were going to stick with Matlab code, we realized that Matlab's serial functions are not too great and the program would end up being much less portable. We decided to write the code in C#, which would be easier to make a visual design for, and it would be a more portable program. While Matlab has the advantage of powerful signal processing functions, the functions we would need are simple and easy to implement in C#.

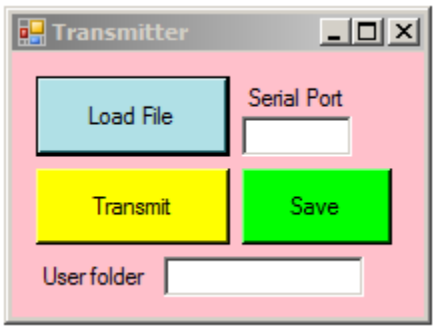

Figure 14: C# Transmitter Program GUI

The program works first by loading an mp3 file to be transmitted. Clicking the "Load File" button brings up a window to select the file. Doing so stores the entire mp3 in memory under a single string. Since every file is just a series of bytes, these bytes can be stored as a series of chars, which is a string. No information is lost and no unnecessary information is otherwise added. It keeps all the other properties and identification data in the header of the file so that when the receiver assembles the decoded information and outputs it to a file, it still has all the same headers necessary for an mp3 file.

Hitting the "Transmit" button establishes a serial connection with the COM port input by the user in the "Serial Port" box at a baud rate of 600, no parity, 8 data bits, and one stop bit. Before this, the program takes the string holding the file and breaks it up into 512 byte sections. It appends each one of these sections to the preamble string to create a series of frames, each one with 7 preamble bytes and 512 data bytes. The choosing of the preamble is discussed below. Once the data has been broken into frames and organized, it is sent out through the serial connection until the end of the file is reached.

For testing purposes, we first used a simple preamble of "AaBbCcDd". We then researched methods of generating a preamble that would work best for our purposes. We wanted to have a preamble that was unlikely to show up by random chance. One of the methods of creating a preamble is called Manchester coding, name for the University of Manchester where it was created. Manchester coding involves XORing a sequence of data with the system clock. Outside of the context of preamble generation, it is used to ease clock synchronization between two devices because Manchester encoding contains bit transitions and optimizes reliability. For the creation of a preamble, we had Matlab generate a random sequence of 1s and 0s, as well as a pseudo-clock signal of alternating 1s and 0s, and then XORed them. The resulting 40 byte array was our preamble.

While this preamble worked well, we later discovered that the IEEE 802.11 Wireless LAN standard has specifications for a preamble. The specified preamble is simply the byte 0xAA seven times. The probability of this sequence appearing in noise is $(2^{-8})^7 = 1.39 \times 10^{-17}$. In other words, the probability that this sequence will appear in noise is about once for every 72 Petabytes of noise assuming 1s and 0s appear equally, which they do not because of the preliminary filtering. This preamble is very simple, standard, and safe to use. The byte 0xAA simply alternates between 0 and 1 (10101010). Because Matlab does not recognize hexadecimal in any way, shape, or form, it is only possible to set the preamble to an ASCII character limited to between 0x00 and 0x7F. The only other character that has an alternating 0 and 1 pattern (01010101) is 'U', or 0x55.

There were difficulties in synchronizing the serial connection between the C# program and the MCU, but we found that a serial terminal program named TeraTerm that could achieve the same results with the "Send File…" function. The design changed to having the C# program take a file and break it into preambled frames, and then send the file out using TeraTerm, which seems to work quite smoothly.

MCU Code

The purpose of the transmitter MCU is relatively simple. It has to generate a logic high voltage signal from the designated pins when the program reads a '1' from the selected bit of a byte and vice versa. The signal is off or no power is sent to the pin when it sees a binary value of 0. Before taking any data from the PC, the MCU must first initialize the buffer array and set the PC baud rate to match the set baud rate from the PC. The MCU will take incoming data from the PC, store it in a buffer array (memory) for the first 3 bytes, and then signal the interrupts to start running. The purpose for doing this is so that interrupts do not occur before data is stored the buffer. The MCU will then constantly recheck if any data is coming from the serial port and insert into the buffer array while determining if each bit in a byte is a 1 or 0. While this occurs, an interrupt is called every 208 µs. During each interrupt, the program reads a bit, starting from the most significant bit (MSB) and determines whether to turn on the transmitting LEDs. This process is bit-masking where it masks or hide the unimportant bits at that moment and looks at one particular bit. Before the interrupt ends, two counters run. One counter is to increment the number of times that the program should shift the bit for the next interrupt. That counter will increment up to 7 times and then start back at 0. At that point, the other counter kicks in and

increments by 1 which corresponds to moving to the next element in the buffer array. This whole process repeats continuously and when the counter reaches the end of the buffer array, it will start back at element zero again.

In our earlier designs for the MCU transmitter code, we relied on using only the hardware timers and ran them continuously. The interrupt used in our current transmitter MCU program inherited some functions in our old program. In our old program, it did bit masking and would constantly keep incrementing until it reached the end of the whole buffer array. It would then look for more data on the serial port to store into the buffer array and start the process again. The issue with this design was that it was not reliable enough for transmitting data with the receiver resulting in major inaccuracies.

2.3.2 Receiver

The receiver, like the transmitter, required two different types of code. C code was written for the MCU so that it could send the data it received from the transmitter to the computer through a serial port. Matlab code was written to read in information from the MCU over a serial port, and then perform appropriate signal processing (match filtering, downsampling), so that the data that was read in could be translated into its original form.

MCU Code

The receiver MCU's job was to sample the output of the photocurrent after analog filtering and amplification, quantize that value, and then output it to the computer so the digital receiver designed in Matlab could process it further and interpret it as text or audio.

Sampling on the Nucleo F401RE was very simple by using the mbed API. Rather than bit masking registers and attempting to set up the ADC manually, we were able to use a couple very simple functions to reliably sample at regular intervals up to 1MHz.

The next step is quantization, which involved taking the 12-bit value from the ADC and setting it to either a 0 or 1. The cutoff was empirically determined and set to a reasonable value based on a distance of 6 inches for testing, which came out to 800 mV converted to 12-bit hex form. Anything above this

voltage would be set to 0 and anything less than or equal to it would be set to 1, to account for the inverting effect of the filter.

An 8-bit character stores bit values ready for serial output for the receiver to take. Whenever the MCU quantizes a sample, it creates a value 0x00 or 0x01 and bit shifts it to the left an appropriate number of spaces, and then ORs it with the serial value. After 8 samples, the MCU sends it out through the serial connection at a specified baud rate, and clears the serial buffer. The table below shows the process with 9 samples.

Table 8: ADC Translating Voltage into Binary

Sample	Sampled voltage	Quantized value	Bit shifted value	Serial output buffer
1	249 mV	00000001	00000001	00000001
2	963 mV	00000000	00000000	00000001
3	1282 mV	00000000	00000000	00000001
4	693 mV	00000001	00001000	00001001
5	1360 mV	00000000	00000000	00001001
6	618 mV	00000001	00100000	00101001
7	324 mV	00000001	01000000	01101001
8	768 mV	00000001	10000000	11101001
9	1492 mV	00000000	00000000	00000000

At first, we were taking 8 samples, storing them into the 8-bit temporary value, and then sending that byte out to the computer. While the MCU was sending the byte out to the computer, it was not sampling and missed 8 samples. In other words, this method could only ever get every other byte. We took the sampling code, put it into its own interrupt, and gave the system a larger buffer to work with. The interrupt uses the bit shifting method just as before, but instead of sending it out to the computer, it stores the byte into an array. The main function ends with an infinite loop that sends data from that array to the computer. There is a slight delay between the enabling of the interrupt and the start of the

loop so that the array has a couple bytes with samples in it before it starts being sent out. This avoids there being any race condition between the interrupt and the main loop.

We finally saw that our interrupts were not interrupting the MCU properly as the timings were off. After investigating, we saw that the ADC was delaying the interrupts causing major inaccuracies in our received data over time. For the most part, data is being sent and received correctly but the inaccurate timing causes the data to shift overtime. This can be seen in Figure 19.

To fix this, we had to change the way that the ADC works so the interrupts would not be delayed.

Matlab Code

A few different files were used to achieve the necessary behavior of the Matlab code. As with the transmitter, the serial connection needed to be established by setting up the serial communication in Matlab. This process was similar to the process described in the transmitter section. The serial port was set up the same, as it had been set up for the transmitter, except, the InputBufferSize variable was increased to ensure that there would be enough space to store the data that is being transferred to the computer from the MCU instead of the OutputBufferSize variable. The code looks for a certain beginning sequence called the preamble. This sequence of ones and zeros has been selected due to the unlikeliness of its occurrence in a string of text or mp3 file.

Beyond that, another sequence of ones and zeros has been selected to appear as the header of each packet of data that is sent. Each time the system reads in the header sequence after it has read in the preamble, it will then know that it is at the beginning of a new packet. At the end of each packet, there is a tail sequence that is similarly unique. When this sequence has been identified, the system knows that the end of the total transmission and that it should not be expecting any more packets.

To make all this happen two functions were created, and some processing was done on the inputs. The first time that the system ran, the binary sequences in Matlab were being automatically converted into decimal values. This meant that the functions being used later in the code would not be receiving their expected inputs. This problem was resolved using the de2bi function to convert each number read in back into its binary format. This binary value would then have each bit stored in an array. The function match_filter.m was created to determine where the transmission started, and then where each packet started. It functions by convolving the known preamble or header sequence with chunks of data as they

come in. The preamble and header sequences will be extended so that each one and zero is now 12 ones or zeros. This is necessary because transmission will be received as 12 of each bit. Once the convolution has been calculated, the function searches for the highest value resulting from the convolution. That index value of that value is then obtained and it is known that the transmission starts at that index. From here, the function downsample.m is used to convert the data from the expanded form it had been received in down to a form that can be translated into ASCII characters or that can be converted back to an mp3 file. When transmitting a sentence, the results of this function will then run through the function bin2text.m. This function will produce the sentence that was originally transmitted without the preamble, header, or tail sequences.

Once the binary code of the mp3 file has been fully transmitted and received, it will be reconstructed back into the original mp3 file, which will then be played back on the destination computer to make sure that the data transmission went smoothly.

3 Results and Conclusions

After many months of testing and project designing, the team came up with a system that exceeded last year's VLC system in terms of transmission speed and distance as well being able to send audio files. The project itself consists of three milestones correlating to three academic terms at WPI. The goal of the first stage was to finish the analog portion of the system. The second stage aimed at having the system transmit text properly and the last stage is being able to successfully transmit and receive an audio file. Results from the team's testing as well as demonstrations of the system transmitting and receiving are documented in the following sub sections.

3.1 Analog Results

After designing and testing the receiver, the team finally got a clean signal from the transmitter as shown in figure 15. All analog results recorded were done during testing under normal ambient indoor light conditions with the receiver and transmitter at a distance of 1 foot apart. The difference in output from the receiver circuitry at 1 inch and 1 foot was negligible.

Two files were used for testing purposes, both of which had been modified to be broken into frames of 7 preamble bytes followed by 512 data bytes by using the C# program. One was a text file with a string of 'U' characters. The purpose of this was to alternate sending 1s and 0s to make it easier for us to see how accurate the timing was. The second file was a short audio clip sampled from an old Iron Man cartoon. This was to test the system's full capabilities of transmitting audio. The files were then downloaded to the MCU by using TeraTerm's "Send File" function.

Figure 15, on the next page, shows the results of testing the transmitter at a serial rate of 1200 bits/sec. The oscilloscope reported the frequency as 602-610 Hz. Because each cycle is actually two bits, one high and one low, this means the rate was about 1204-1220 Hz, slightly above the set rate. However, the oscilloscope is not perfect in its frequency calculation. Using the oscilloscope's cursor function revealed that every logic cycle was almost exactly 833 µs, which was what the interrupt timing was set to.

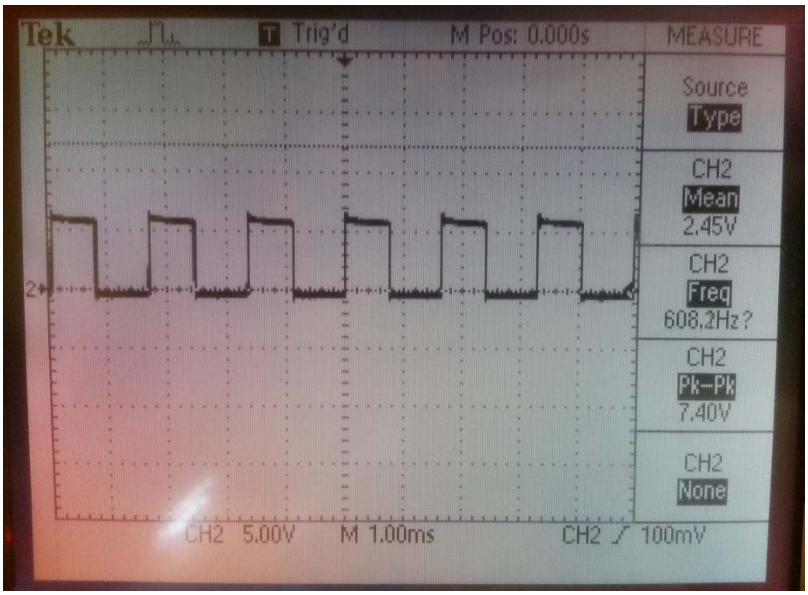

Figure 15: Op-amp output from 1200 bits/sec 'U' Signal at 1 foot

We then sent the audio file to get the results below. What we noticed is that when there were several 1s in a row, there was a bit of a drop in the logical voltage because of the capacitor used to cut out the DC portion of the signal. However, even with an entire byte of 1s, i.e. 0xFF, our system can still work, as the drop was not significant enough to pass the threshold.

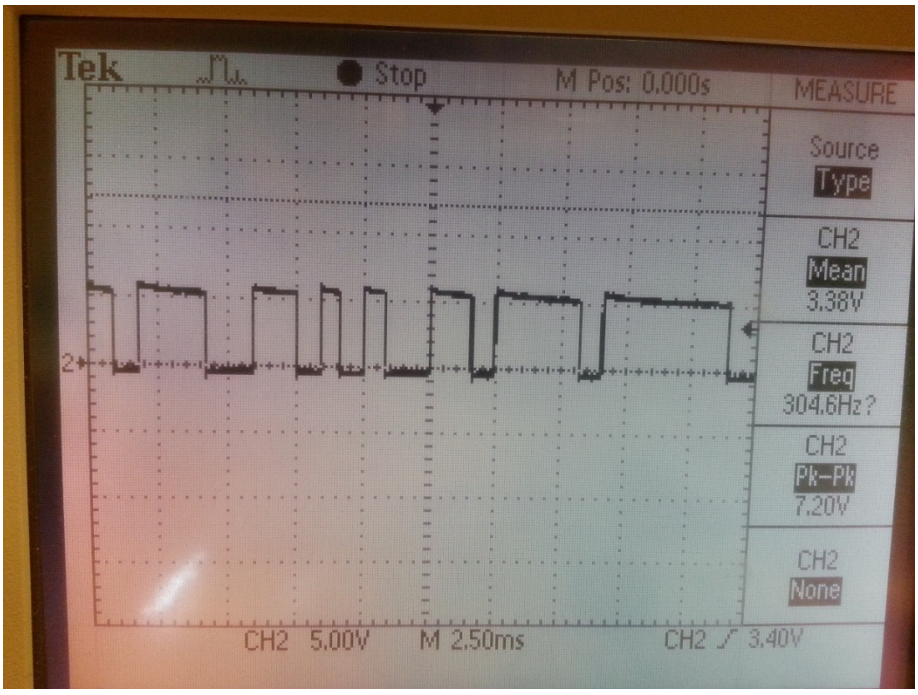

Figure 16: Op-amp output from 1200 bits/sec audio signal at 1 foot

1200 bits/sec was a bit low for what we wanted to transmit, and so we decided to increase the transmission rate. The next rate would be 2400 bits/sec, but we found that rate to be unstable on the Nucleo board, so we moved up to a baud rate of 4800. The picture below shows the results of that test using the 'U' signal. We were able to get a reliable signal, although it did seem as if the rising time and falling time cut into the on time of the logic highs. As can be seen below, when the signal switches from a 1 to a 0, it falls below 0V very quickly before coming up to a logic low, and because of this the high is noticeably shorter. The signal also appears slightly noisier because of the tighter timing constraints, but not nearly enough to affect the results at all. Any error would come from sampling during the edges, but this would be cut out through downsampling. The noise was significantly diminished by placing covering the top of the photodiodes with a piece of plastic to cut out the ambient noise injection.

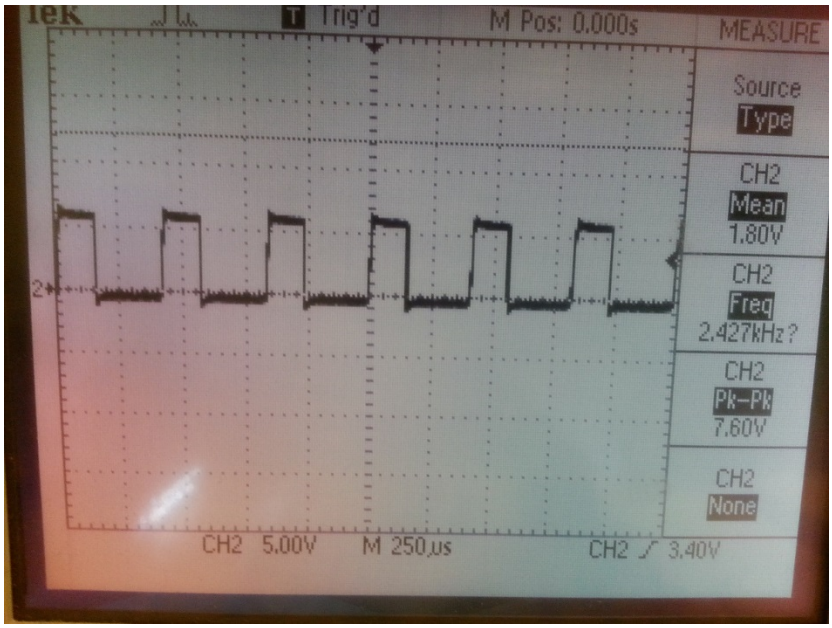

Figure 17: Op-amp output from 4800 bits/sec 'U' signal at 1 foot

3.2 Digital results

Using the same test set up used in the analog test, two different types of digital tests were performed. The initial test was done using a short message of text with a preamble. The second test was done using the C# program to transmit a 150 KB MP3 file. This file was broken up into a number of smaller packets for transmission. Each of these packets had a header sequence at the beginning of it and a known number of bytes after that. This allowed us to identify the beginning and end of each packet.

3.2.1 Transmitting Text

Some basic testing was done initially using a hard coded message of “Hello World” on the transmitter end of the MCU. Using a preamble of “He” and a tail of “ld” the following result was shown in Matlab.

```
msg_txt =

llo World
```

Figure 18: Matlab Text Output - Initial Test

From here, more tests that are complex were attempted using larger quantities of data such as audio files. Some changes were made to some of the Matlab files to allow for frame-by-frame reading in of data and to remove the tail from the actual transmission.

3.2.2 Transmitting an MP3 File

After successfully transmitting a small string of text, the next goal was to transmit an audio file. However, when this was attempted it failed. A number of tests were performed to determine the source of the problem.

First, a baseline needed to be established. By transmitting only the letter ‘U’ and then running the Matlab receiver code, the following output was obtained.

50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 19: Matlab Output 'U'

The first test that was performed was to check and see if the output of the op-amp was what it was expected to be. This test was performed by setting the transmitter to repeatedly send the letter ‘U’, which alternates between one and zero every bit.

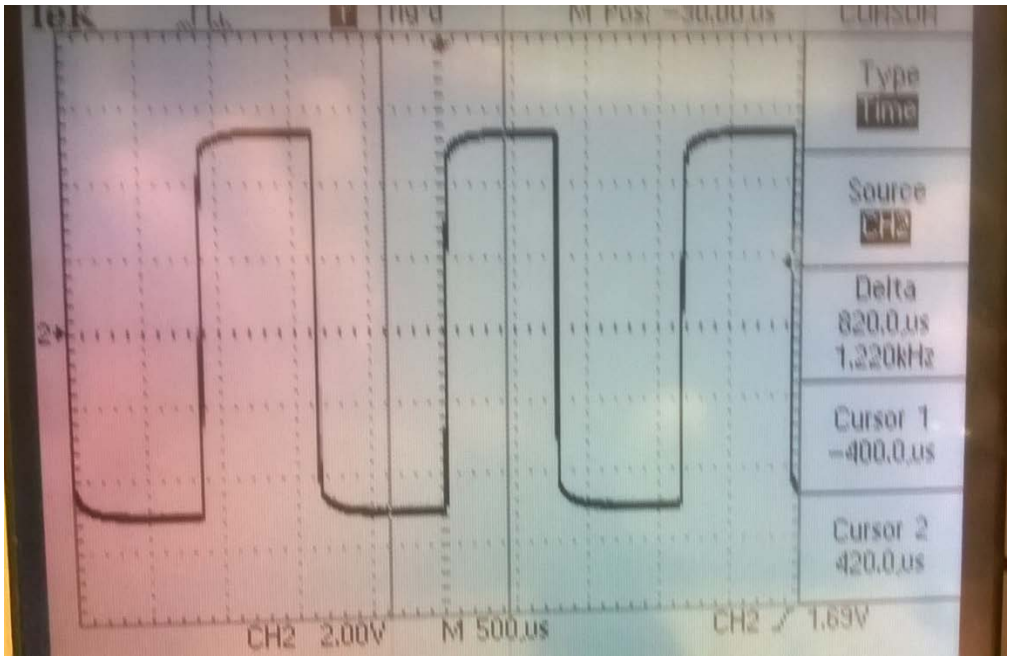

Figure 20: Transmitting 'U'

While performing this test, the rise and fall time of the signal were also noted. The rise time was found to be 30 us, and the fall time was found to be 6 us. Both of these values were deemed to be too small to be influencing the interpretation of the signal that was being received. This would likely only result in a small occasional error at the beginning or end of the signal not the odd bit flipping behavior that was observed in Figure 19.

After determining that the problem was not the signal being seen at the output of the op-amp it was decided that the issue had to be something related to the receiver MCU. The next test that was performed involved filling the MCU buffer with the letter ‘U’ and removing the interrupt call to the ADC. This was done to avoid any potential problems caused by the interrupt taking too long and interfering with the serial communication between the MCU and the computer. First, the test was done to see if the letter ‘U’ was being read on the serial port. TeraTerm was configured to view what was being printed to the serial port, and the output below shows that as expected, the letter ‘U’ was being printed repeatedly to the serial port.

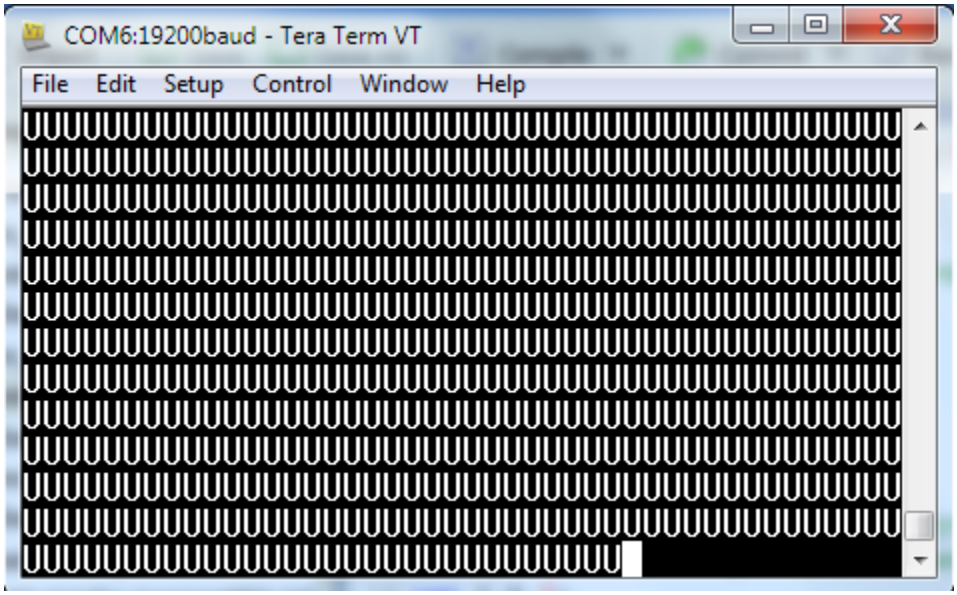

Figure 21: TeraTerm Output 'U' without Interrupts

From here, the Matlab code was run to check and see if the letter had any flipped bits showing up when it was being sent. As seen below, that was not the case. The bits were being sent exactly as expected.

bin3

16x2048 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	1	1	1	1
7	0	0	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	1	1
9	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	1	1	1	1	1	1
11	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	1	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0
16	1	1	1	1	1	1	1	1	1	1	1

Figure 22: Matlab Output 'U'

This proved that serial transmission was able to happen correctly while the interrupts were not occurring. From here, the next step was to add the interrupt back into the code. The interrupt was implemented in full, except for the portion of it related to reading from the ADC. Once again, TeraTerm was used to prove that the letter ‘U’ was being properly received, and once again, this was the case.

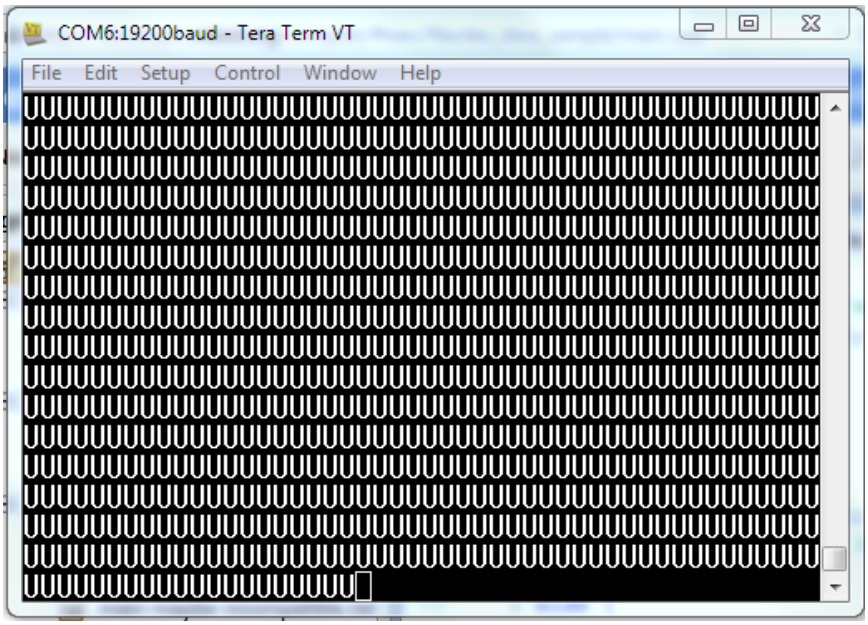

Figure 23: TeraTerm Output 'U' with Interrupts

This demonstrates that the issue was not the interrupts taking too long and preventing the serial communication from continuing accurately. From here, the next step was to see if the interrupts were running properly. A variable within the interrupt was set so that it would flip between a 1 and a 0 every time the interrupt was called. Then the value was output to a pin on the MCU and that output was measured using the oscilloscope.

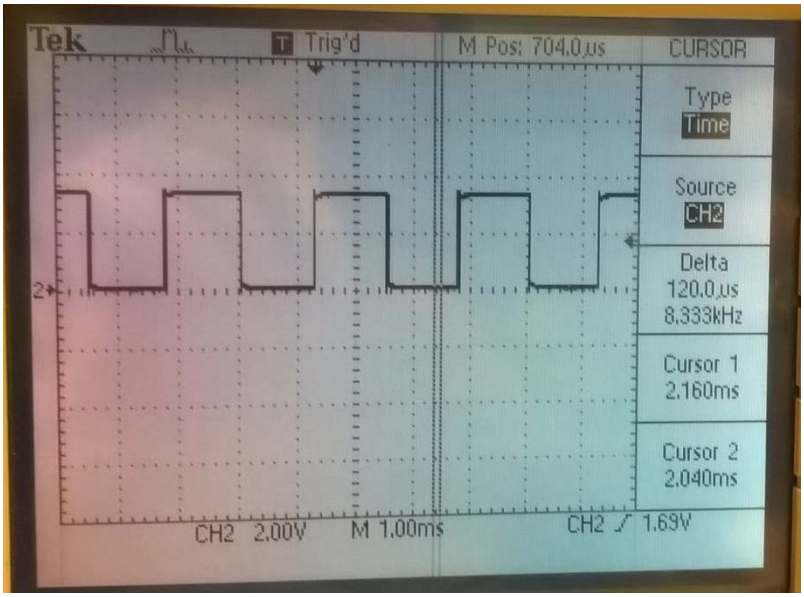

Figure 24: MCU output Test

As seen in Figure 24, the output behaved as expected proving that the interrupts were functioning correctly. This led us to believe that the ADC was causing the problem. Modifying the previous diagnostic test, we had the variable flip to whatever we were intending on sending to the computer. If the sample from the ADC was above 1600 mV, it would be set to a 1, and if it was below it would be a 0. Ideally, we would have and output similar to Figure 15. Instead, we got the output shown below.

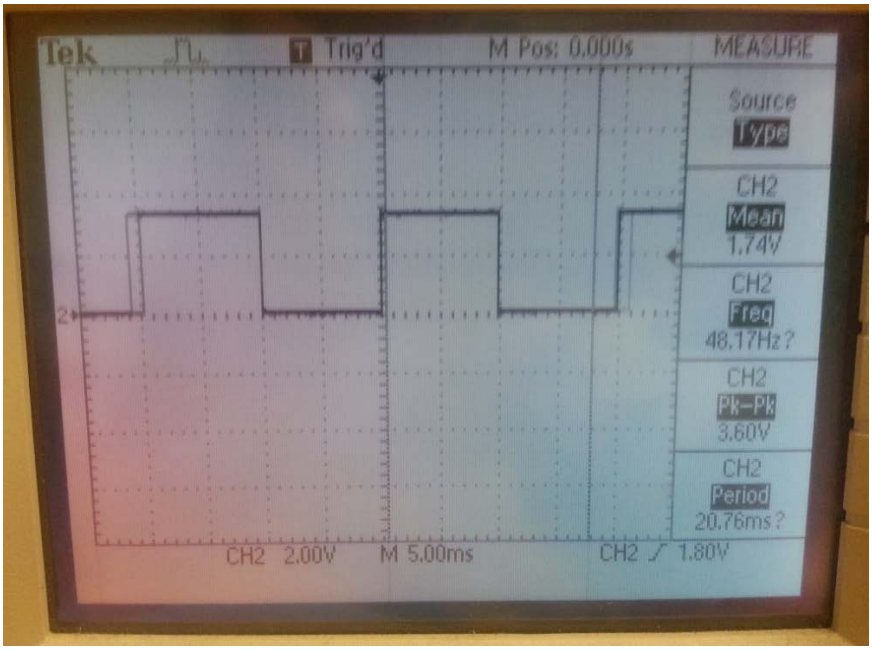

Figure 25: Ghosting on the Wave Form

It is somewhat difficult to capture in a picture, but there is ghosting of different wave shapes. On the leftmost and rightmost waves, there appears to be two rising edges. This ghosting is caused by the wave width changing and the triggering, resulting in them being overlaid inaccurately. While it appears as though only the first or last samples are being dropped or flipped, that is simply because of the way the oscilloscope does triggering and displays them. In actuality, bits in the middle of a logic cycle are flipped. This definitively proves that the issue is with the ADC and nothing else. Directly before the ADC, at the analog output into the MCU, we see a clean square wave that matches exactly what we intended to send from the transmitter. Immediately after, we see an irregular shape. Cutting out the ADC and relying only on the MCU's other features and the Matlab code produces exactly what we want. Research into the problem revealed that the ADC on the MCU's that were being used could only be set to continuously sample using provided bit masks. Due to a lack of documentation easily available about the registers of the MCU, it became unfeasible to alter the way the ADC was functioning without drastically extending the length of the project.

3.3 Overall Conclusions

The completion of this project largely depended on many days of researching and testing. This project was used as a continuation and general improvement of the VLC project completed last year. Utilizing the knowledge the previous team gained through their experiences, a number of improvements could be made. However, as with last year, many of the problems experienced in the system were related to limitations of the MCU, specifically the ADC.

Designing and building the transmitter for the system was relatively easy. It was a relatively straightforward update of the previous design. The receiver design however presented a bit more of a challenge in the way of unreliable components, and high power drain from the batteries. The basic format of these circuit features was overall very similar to the previous year's design. The main difference came in component selection, particularly the photodiodes and the MCU's.

At the start of this project, the team had set a goal to surpass last year's VLC system in terms of transmission speed and distance. Additionally, there was the added goal of transmitting a larger quantity of data, in this case an audio file. While the final system is not able to transmit audio properly, the project was still a success in terms of achieving a higher transmission speed and a larger transmission range. The system attained a transmitting distance of 2/3 of a meter, with the potential to be increased to an even greater distance with ease, at a transmission rate of just 1.2 kbps while operating in an ambient light setting.

Overall, the VLC system prototype that was designed was a good continuation of research into the field of VLC technologies. Hopefully, the research that was completed for this project will aid others in delving deeper into the possibilities of VLC technologies.

3.4 Future Recommendations

With this project ending, future students at Worcester Polytechnic Institute will have the opportunity to develop an even better system on the foundation this team has laid. The VLC system created currently transmits small text messages at a relatively high speed. Future projects will have the option of continuing from where we have left off, using this technology to transmit data in forms such as live streams of audio or video files.

Another problem that future teams are likely to encounter is in the way their system's ADC functions. As mentioned in previous sections of this paper, the functionality of the ADC is what eventually caused us to realize that it was not feasible to transmit an audio file successfully. As mentioned previously, the problem is solvable, but given the amount of time that the team had left when the issue was discovered, there was not enough time. WWhen choosing a microcontroller for the transmitter and especially the receiver is to choose the board that has good documentation andgives you the ability to accessthe registers directly for more flexibility when programming.

In the coming years, we can expect that VLC technology will vastly improve and become ubiquitous worldwide because of its advantages. It is important that future students investigate this promising new technology, as VLC systems will play a major role in the world of wireless communications since the Internet of Things becomes closer to reality.

References

[1] http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html#~forecast

[2] http://www.ntia.doc.gov/files/ntia/publications/2003-allochrt.pdf

[3] FCC 2014

[4] https://www.wpi.edu/Pubs/E-project/Available/E-project-032814-001416/unrestricted/MQP_Report_Final_Draft_3_27_14.pdf

[5] http://www.theengineer.co.uk/in-depth/the-big-story/light-reading-visible-light-communications/1007419.article

[6] http://ronja.twibright.com/

[7] http://axrtek.com/momo/

[8] http://www.bluehaze.com.au/modlight/GrothArticle1.htm

[9] S. Louvros, D. Fuschelberger, N. Sklavosm M. Hübner, D. Goehringer, and P. Kitsos, "VLC Technology for LTE Indoor Planning," *System-Level Design Methodologies for Telecommunication*. NY: Springer, 2014, pp. 23.

[10] http://www.siemens.com/innovation/en/news/2010/500-megabits-second-with-white-led-light.htm

[11] http://www.ted.com/talks/harald_haas_wireless_data_from_every_light_bulb

[12] http://www.techthefuture.com/technology/using-visible-light-frequencies-for-wireless-data-transfer/

[13] http://www.ieee802.org/15/pub/TG7.html - IEEE 802.15.7

Appendix A: Parts List

Component	Model Number	Quantity
Microcontroller Unit	STM32F401RE	2
Op-Amp	AD848JN	1
Transistor	2N2222A	1
LED	LW514	10
Photodiode	SFH 213	5

Appendix B: Matlab Code

receiver.m

```
%Receiver.m
%Known Constants
frame_size = 4096;
header = 'UUUUUUU';
%expected out put is llo Wo
flag = 1;
%in_buf = 2^ 18; %this is for testing
timeout = 30;

%Open Serial Port
s = serial('COM6');
%serial port settings
set(s,'BaudRate', 19200, 'InputBufferSize', frame_size, 'Timeout', timeout,
'Terminator', 'CR');
fopen(s);

%Read in data from serial port
out = fread(s, frame_size, 'uchar')

%Close Serial Port
fclose(s);
delete(s)
clear s

%Processing
%out is decimal values
bin = de2bi(out,8);
bin = fliplr(bin);
%bin is a multidimensional array aka [255 255] => [1 1 1 1 1 1 1 1; 1 1 1 1 1
1 1 1]
bin2 = reshape(bin',1,numel(bin));
%downsample
down_data = downsample(bin2);
%Run data through a Match Filter
header = text2bin(header);
[msg_bin, index] = match_filter(down_data,header,frame_size);
msg_bin = bin2text(msg_bin)
%convert to text
```

match_filter.m

```
function [match_data, index] = match_filter(received_data, header,
frame_size)

%Match Filter Function:
%   This function will take binary data received from the MCU and will
%   compare it to the known preamble or header to determine
%   where the start of the transmission is

%Find where in the array, the message should be converted from
%performs correlation on the received data and the preamble
corr = xcorr(received_data, header);
%finds the peak of the correlation
maxCorr = max(corr);
%determine which index has a value that is equal to maxCorr
index = find(corr == maxCorr);
frames = length(index);
%calculates the index that is the start of the actual data
for n = 1:frames
    index(n) = index(n) - length(received_data) + length(header) + 1;
end
%uses the index calculated above to get the appropriate starting point
%match_data = zeros(length(index)*frame); %used in none frame by frame
%transmission
match_data = received_data(index(1):index(1)+frame_size-1);
%added for frame by frame transmission
for n = 2:frames
    %match_data((n-1)*frame+1:n*frame) =
received_data(index(n):index(n)+frame-1);
    match_data = horzcat(match_data,
received_data(index(n):index(n)+frame_size-1));
end

end
```

downsample.m

```
function down_data = downsample(data)

%Downsampling Function:
%   This function will take binary data received from the MCU and will
%   compare 10 bits to determine if they represent a 1 or a 0
%   it will output an array of the data that is the binary representation
%   of the message that was sent to the receiver
%   if half of the set of 10 bits is a 1, it will default to a 1

%Changed to deal with a varaible sample_rate, set the sample rate below
sample_rate = 12;

len2 = length(data);
%makes an empty array of zeros
down_data = zeros(1,ceil(len2/sample_rate));
%creates the result array which should be the same as the one sent from the
%Tx side
counter = 0;
binary_val = 0;
%loop to traverse the match_data array and downsample it
for i = 1:1:len2
    %if the counter is less than the number of samples taken
    if data(i) == 1
        binary_val = binary_val+1;
    end
    if counter == (sample_rate - 1)
        if binary_val > sample_rate
            down_data(i/sample_rate) = 1;
        else
            down_data(i/sample_rate) = 0;
        end
        counter = 0;
        binary_val = 0;
    else
        counter = counter + 1;
    end
end


%down_data is the output of this function.
```

Appendix C: C# Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;

namespace Transmitter
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        byte[] file = null; //initialize file output

        byte[] preamble = { 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55 };
//preamble, 'U' in hexadecimal

        //Load File
        private void button1_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFile = new OpenFileDialog(); //open Windows
Explorer file loader
            if (openFile.ShowDialog() == DialogResult.OK) //when user clicks
OK
            {
                System.IO.Stream fileStream =
File.OpenRead(openFile.FileName); //stream file
                System.IO.BinaryReader br = new BinaryReader(fileStream);
//set to read

                FileInfo f = new FileInfo(openFile.FileName); //load file
properties
                file = br.ReadBytes((int)f.Length); //read all bytes of file
            }
        }

        //Transmit
        private void button2_Click(object sender, EventArgs e)
        {
            int offset = 0;
            int i = 0;
            int max = (file.Length / 512); //number of frames to create

            System.IO.Ports.SerialPort comm = new
SerialPort(String.Format("COM{0}", txtPort.Text), 600, Parity.None, 8,
```

```
StopBits.One); //setup COM port

            for (i = 0; i < max; i++) //for each frame
            {
                comm.Write(preamble, 0, 7); //write the 7 preamble bytes
                comm.Write(file, offset, 512); //write the next 512 bytes of
the file from the offset
                offset += 512; //jump forward one frame in the original file
            }
        }

        //Save
        private void btnSave_Click(object sender, EventArgs e)
        {
            int offset = 0;
            int i = 0;
            int max = (file.Length / 512);

            System.IO.BinaryWriter comm = new
System.IO.BinaryWriter(File.Open(String.Format(@"C:\Users\{0}\Desktop\file.mp
3", txtUser.Text), FileMode.Append)); //setup file output

            //same as above
            for (i = 0; i < max; i++)
            {
                comm.Write(preamble, 0, 7);
                comm.Write(file, offset, 512);
                offset += 512;
            }
        }
    }
}
```

Appendix D: MCU C code

Transmitter Code

```
#include "mbed.h"

#define END 519
//max number of bits in block array **change to how long the message (change
accordingly with c# program)
//we are sending through matlab [512 bytes per frame + 7 or more bytes for
preamble] (4152 bits)**
//519 bytes

Serial pc(SERIAL_TX, SERIAL_RX);
Ticker interrupt; //call interrupt into program
DigitalOut my_pwm(D8); // IO used by pwm_io function

void interpret();

char buffer[519]; //buffer array to hold the data frame coming from the PC
int i=0;
int k=0;

int main() {
    pc.baud(1200);
    int j = 0;
    for(int k=0; k <END; k++) {
        buffer[k] = 0; //initialize block array for incoming data from C#
program for first time
    }

    //fill up buffer a little bit before sending it out to the LEDs to avoid
race condition
    buffer[0] = pc.getc();
    buffer[1] = pc.getc();
    buffer[2] = pc.getc();

    interrupt.attach_us(&interpret, 832); //interrupt to send out bit to LEDs
every 832us (1/1200)

    while(1) {
        if (pc.readable()){ //detects if pc is sending data to mcu
            while (j < END) { //Load Serial data into memory
                buffer[j] = pc.getc();
                j++;
            }
            j=0;
        }
    }
}

//interrupt calls this function
void interpret() {
```

```
    if (buffer[i] & ( 0x80 >> k )) //check only one specific bit
        my_pwm = 1; //LED is on
    else
        my_pwm = 0;  //LED is off
    k++;
    if(k > 7){ //if end of byte is reached
        i++;
        k = 0; //reset bit counter
        if(i == END) { //if end of buffer is reached
            i=0; //reset buffer counter
        }
    }
}
```

Receiver Code

```
#include "mbed.h"

#define MAX 512
AnalogIn analog_value(A0);
Serial pc(SERIAL_TX, SERIAL_RX);
Ticker interrupt;

// Calculate the corresponding acquisition measure for a given value in mV
#define MV(x) (x)/3300

void sample();

float meas;
int i = 0; //bit shift counter
char byte = 0x00; //temp sample byte
int k = 0; //interrupt buffer counter
int j = 0;
char byte_array[512]; //serial buffer


int main() {
    pc.baud(19200); //serial out at 9.6 kbps

    //initialize serial buffer
    for(j=0;j<MAX;j++){
        byte_array[j]=0;
    }

    my_pwm=0;

    interrupt.attach_us(&sample, 52); //enable sample interrupt every 52us
(1/19200)
    wait_us(1600); //let a few bytes of the buffer fill up before sending it
out

    while(1) {
        for(j=0;j<MAX;j++){ //automatically loops back around
        pc.putc(byte_array[j]); //send next element of serial buffer
        }
```

```
    }
}

void sample() {
    meas = analog_value.read(); // Converts and read the analog input value
    if (meas > MV(1600)) { // If the value is greater than 80 mV toggle the
LED
        //op-amp inverts signal
    } else {
        byte |= 0x01 << i; //bit mask
        }
    i++;
    if (i == 8) { //if temp byte fills up
        byte_array[k] = byte; //store temp byte in serial buffer
        byte = 0x00; //reset temp byte
        i = 0; //loop bit counter back around
        k++;
        if (k == MAX) { //if reached end of serial buffer
            k = 0; //loop buffer counter back around
        }
    }
}
```