PROJECT 2

VISIBLE LIGHT-BASED DIGITAL COMMUNICATION USING ON/OFF KEYING

Submitted by:

OĞUZHAN ZENGİN	**2015401018**
ZELİHA ASENA KIRIK	**2015401033**

TABLE OF CONTENTS

1) INTRODUCTION

1.1) INTRODUCTION TO PROJECT

In this project, we were asked to construct a wireless transmitter/receiver pair, capable of transmitting digital information across a few centimeters using light. Specifically, it should employ on-off signaling, which is the simplest digital transmission technique, and encode the information (sentences) using ASCII codes. As a second part of the project, to compensate for possible timing errors/offsets/jitter which often occurs in communication systems, Manchester coding is also implemented.

1.2) RELATED BACKGROUND

Visible light communication (VLC) is a data communications variant which uses visible light between 400 and 800 THz (780–375 nm). VLC is a subset of optical wireless communications technologies. VLC can be used as a communications medium for ubiquitous computing, because light-producing devices (such as indoor/outdoor lamps, TVs, traffic signs, commercial displays and car headlights are used everywhere. Using visible light is also less dangerous for high-power applications because humans can perceive it and act to protect their eyes from damage. The history of visible light communications (VLC) dates back to the 1880s in Washington, D.C. when the Scottish-born scientist Alexander Graham Bell invented the photophone, which transmitted speech on modulated sunlight over several hundred meters. [1]

VLC is one of the promising candidates for future communication systems due to its features of non-licensed channels, high bandwidth, and low power consumption. Li-Fi, vehicle to vehicle communication, and robots in hospitals, underwater communication, and information displayed on signboards can be counted as potential applications of VLC. For example in vehicular communication, VLC can be used for lane change warning, pre-crash sensing, and traffic signal violation warning to avoid accidents. These applications require communication with low latency which is provided by VLC because of its high bandwidth and easier installation due to the existing presence of vehicle lights and traffic signals. VLC also has applications in areas that are sensitive to electromagnetic waves, such as aircraft and hospitals where the radio signals interfere with the waves of other machines. On the other hand, there are some challenges in the implementation of VLC such as interference with the ambient light sources, interference between VLC devices, and integration of the VLC with existing technologies such as Wi-Fi with suitable protocols [2].

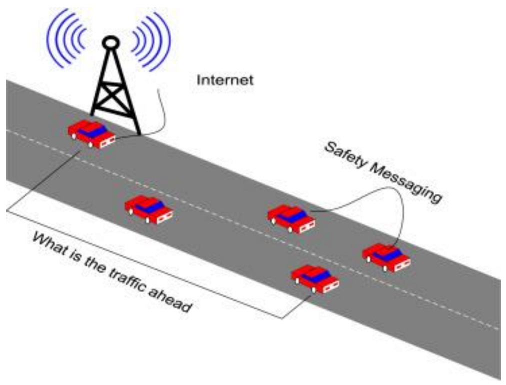

Figure 1: VLC for vehicular networks

1.3) EQUIPMENT

- Arduino Uno's (x2)
- a white LED
- a light-dependent resistor
- 100 ohm resistor (x2)
- 10k ohm resistor

2) OBJECTIVES

The objective of this project is transmitting digital data across few centimeters using light with faster (high data rate) and more robust (low bit/character errors) as much as possible.

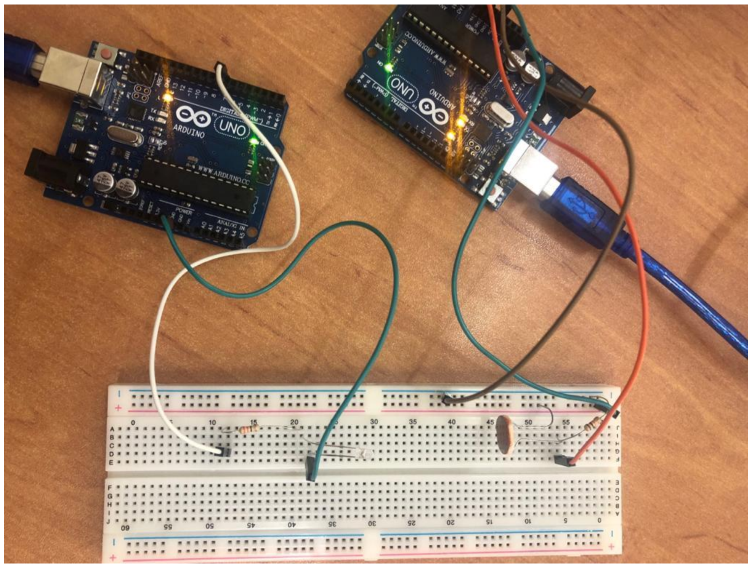
Figure 2: Implementation of Circuitry

3) METHODOLOGY

The overall schematic of the project can be seen in Figure 2. So one of the Arduinos will be used to transmit data by emitting light through LED and at the receiver side, Arduino will detect the signal. In this schematic, when there is more light coming, the resistance of photodiode will decrease and Arduino's analog input value decreases. On the other just for the convention, we wanted to see higher input at the receiver when light is on so we exchanged the position of 5V and ground on the right side of the circuit. After constructing the circuit above, we started coding.

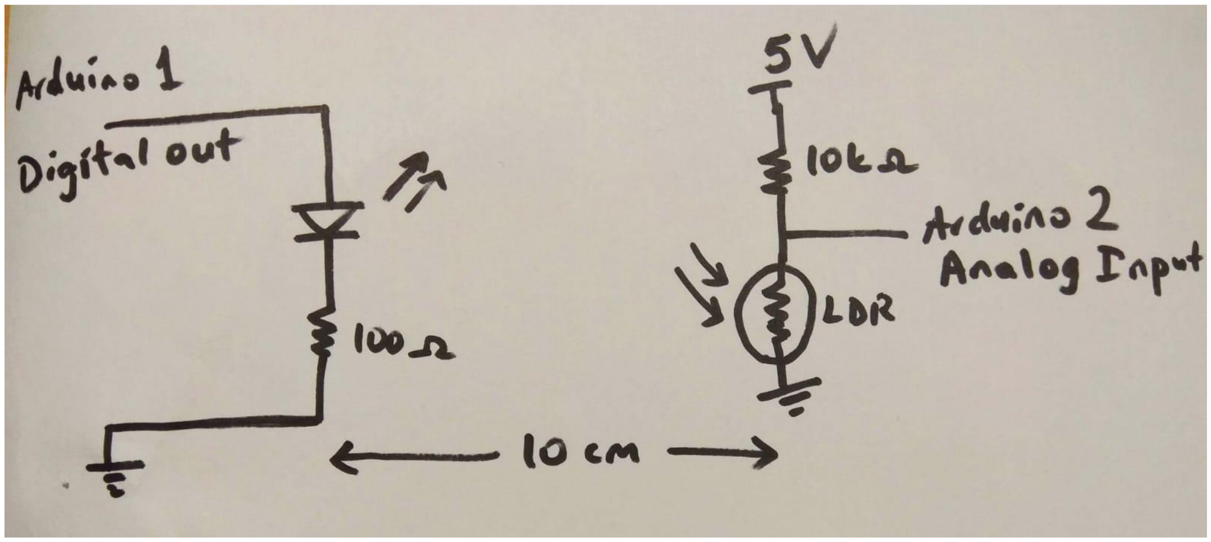

Figure 3: Schematic of Overall System

3.1) TRANSMITTER DESIGN OF ASCII CODE

We implemented this part in Arduino software. In our transmitter design, we first stored the input sentences to be sent in a char array. Then, the letters are converted to their ASCII equivalent decimal numbers and then the decimal numbers are converted into 7-bit binary arrays. Then we turn digital output “HIGH” if corresponding input bit is ‘1’ and “LOW” if corresponding input bit is ‘0’ in each binary array. This process is done letter by letter until we reach the end of the input. We tested different intervals for bits. At a minimum interval of our design, each high and low output stays active for 25ms. So, it takes 175ms to send one letter. For, synchronization purposes we are sending “010” array first and give one pilot string for the beginning {0,0,0,0,0,1,0} and {0,0,0,0,1,0,0}for the end. Additionally, due to the memory limitations of Arduino, the user should pay attention to the length of the string.

3.2) RECEIVER DESIGN OF ASCII CODE

In the receiver design, we used MATLAB. To determine the data rate of Arduino, we sent Arduino example “Blink”. From there ((1/data interval)*(number of samples in one period) we calculated Arduino’s sampling rate(8325 in our case). We chose buffer size 100, so we are reading data 100 bytes by 100 bytes. It does not have a significant effect on the project because we are checking bit by bit. Then we start communication by taking some garbage data during the connection of Arduino, after some point, “Ready” sign will be displayed on the screen. After this sign, the user can reset the transmitter Arduino and initialize the transmission. Later we start to read the data. The data value will be between 0-255. For the threshold value 120, if a bit is more than the threshold, bit ‘1’ will be saved otherwise ‘0’. First, we implemented a synchronization unit. It takes 1 buffer size information first, later when it takes the other buffer size amount of data, it checks the one by one of its values. At every bit checking, we add the bit to the end of synchronization_buffer and omit the first bit of it. We also keep the whole data in ‘total’ variable so we add the corresponding bit to total. Later we check for the rising edge and the following falling edge for synchronization purposes (whether “010” is sent). We also check the duration between the rising and falling edges to be sure that the bit sequence is really “010” but not ‘0110’ for example. In if statement for the bit checking instead of looking exactly one period between rising and falling edge, we are checking 1.4T 0.75T periods. The reason for this is since ‘1’ bit has a longer duration than ‘0’ bit in our construction due to the non-ideal behavior of photo-resistor. Then we found samp_point where the middle point of second sync bit. Then we continue to count until the samp_point+T, which means the midpoint of third sync bit. From here, we can start to read the real data. (sync_comp=1) we will count sync_ct, sflag becomes 0. Then, we start counting samples from the middle of the third bit of synchronization bit ‘0’. It should be ‘0’as it is initiated; otherwise “Error” will be displayed on the screen. Afterward, we start to read real data. samp_count will start to count after the one-bit period, it will correspond the first real information(not pilot data) bit and will be sampled by the checking the same threshold value. After it reaches to T, we will check edge_ct with half of the duration. Edge_ct becomes zero after any edge. So checking the difference provides us a synchronization mechanism. If it is bigger, the signal is a backward delay, so we adjust our sample_count to back as well. By doing this, we avoid accumulated delay. After doing these checkings, when we have 8 bits(1 byte), we convert to binary value to decimal and its corresponding ASCII value. ascii_ENDTR is the pilot to terminate the code. If the byte is equal to this end comment, the program will stop.

3.3) TRANSMITTER DESIGN OF MANCHESTER CODE

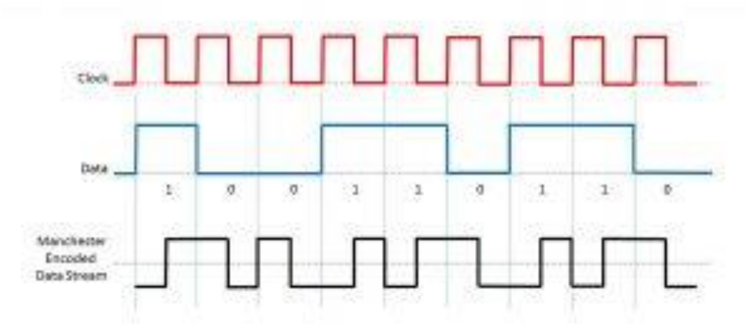

Figure 4: Manchester Code

In the transmitter side of Manchester Code, we have similar implementation. In our design for this transmitter, we first again stored the input sentence to be sent in a char array. Then, the letters are converted to their ASCII equivalent decimal numbers and then the decimal numbers are converted into 7-bit binary arrays. Then we turn digital output “HIGH” if corresponding input bit is ‘1’ and “LOW” if corresponding input bit is ‘0’ in each binary array. This process is done letter by letter until we reach the end of the input. We tested different intervals for bits. At a minimum of our design, each high and low output stays for 40ms. So, it takes 280ms to send one letter. For, synchronization purposes we are sending “10” array first and give one pilot string for the beginning {0,0,0,0,0,1,0} and {0,0,0,0,1,0,0}for the end. In order to make it Manchester coded, whenever we see ‘1’ we put “01”; when we see ‘0’ we put “10”. So the total bit number doubled. Additionally, again due to the memory limitations of Arduino, the user should pay attention to the length of the string.

3.4) RECEIVER DESIGN OF MANCHESTER   CODE

In the receiver part of the Manchester Code, we are doing the same thing with the general ASCII code except for few differences. Our threshold value is 130. We are again taking garbage data and the user should reset transmitter Arduino after seeing the “Ready” sign. Later as a synchronization bit, we are sending a “10” bit. Which means one rising edge and one falling edge. In the figure below Manchester coding can be seen. After seeing a falling edge(in the middle of the second pilot bit), the sample_counter starts counting. When it encounters an edge between 0.75T and 1.25T after sample_counter, it becomes 0 again. By checking the magnitudes of previous and current bits we can determine symbol data. After control bits “10”, sample_counter started and it checks edge between 0.75T and 1.25T. We can determine the bit by checking if the edge is a rising edge or a falling edge. By resetting the counter at the detected edge, we automatically perform a resynchronization with respect to the edge.. Then we will follow similar steps of ASCII receiver. First, save it into memory then checking every byte with STR_End. When it is STR_end, the program stops working.

4) RESULTS &CONCLUSION

For the test cases in ON/OFF signaling fastest, we were able to send bits was with 25ms periods. In ASCII coding, the smallest period was 25ms. Even though a long sentence is given, this period is in the tolerance range. In the ASCII coding case, at period 60, the algorithm was able to tolerate up to 5% and sometimes 6%. When a random period is given, we were even able to tolerate 13%. At period 25, we were able to recover some letters in 4% of randomness in the period.

In Manchester coding, the smallest period, we can obtain was 40ms. At 60 ms interval, we were able to tolerate until 16.5% constant delay.   At random delay, again we were able to tolerate until 16.7%.

In the ASCII case, we generally expect a random delay in period gives better results than constant delay. Because at a constant delay, there is a risk of accumulating delay, especially when same bit is transmitted multiple times consecutively. On the other hand, in random cases, backward and forward cases will eliminate each other's effects. From the values, it can be seen that in the same period, the performance is improved. The system can tolerate more delay when delay is random. In the Manchester coding case, random delay and constant delay gives very similar results. This is mainly because the delay does not accumulate in the Manchester coding case, due to the way the detection algorithm works. There must be a mid-bit edge in every Manchester coding symbol, so resynchronization can be performed with respect to this edge, in every symbolling period.

5) REFERENCE

[1] En.wikipedia.org. (2019). *Visible light communication*. [online] Available at: https://en.wikipedia.org/wiki/Visible_light_communication [Accessed 17 Dec. 2019]

[2] Khan, L. (2017). Visible light communication: Applications, architecture, standardization and research challenges. *Digital Communications and Networks*, 3(2), pp.78-88.