LiGET: Transfering files via Li-Fi

Benjamin Creem, Brett Grossman, and Mira Yun
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115, USA
{creemb, grossmanb1, yunm}@wit.edu

***Abstract*—With the radio spectrum growing ever more crowded and demands for faster, more secure wireless connections continuing to skyrocket, the prospect of communication via the light spectrum grows. This technology, known as Light Fidelity (Li-Fi), has been generally considered state of the art and not sufficiently accessible either as a commercial product or for educational use. This paper will demonstrate the viability of undertaking an *Arduino* based Li-Fi file transfer system as undergraduate work.**

Keywords— *Li-Fi; Arduino; File Transfer system;*

I. INTRODUCTION

Light Fidelity (Li-Fi) is a wireless communications technology that substitutes usage of the radio spectrum with that of visible light [1]. This is a powerful, relatively new technology that draws a great deal of interest due to its potential for speed, security and avoidance of spectrum crowding. However, Li-Fi generally only sees experimental work and is highly unlikely to be found in the undergraduate classroom setting. With the growing prospect of commercial use of Li-Fi, integration of basic lessons in its usage and capabilities can be a great asset to undergraduates.

In this paper, we propose *LiGET*, a one-way Li-Fi based file transfer system, to introduce Li-Fi technology and applications for undergraduate education. At around $60, the build for our project was relatively cheap and had assembly that can be easily accomplished with proper instructions. With off-the-shelf and low cost hardware, it would be feasible to re-create our project in a classroom environment in order to gain crucial experience working with Li-Fi.

The rest of this paper is organized as follows. Section II provides Li-Fi introduction for undergraduate education. Section III describes the proposed system design and implementation details. Finally, Section IV summarizes the paper and outlines ideas for future work.

II. LI-FI FOR UNDERGRADUATES

Li-Fi works by use of LEDs to transmit data over the visible light spectrum [2]. By using visible light spectrum, transmission speed can be significantly increased. This can be accomplished by using the presence of light, and conversely the lack thereof, to represent 1s and 0s. The LEDs blinks at extremely fast speeds in order to send data at desired rates.

It is commonly known that light's movement through air has the fastest velocity of anything harnessable by today's technology. By transmitting data on this spectrum, the travel time of data between any given devices is significantly decreased. Another incentive for using Li-Fi is the consistently growing problem of spectrum crowding. As more and more wireless devices enter the market, the radio spectrum becomes increasingly crowded, increasing problems of interference and making it difficult to find spectrum bands to license [3]. Moving wireless communication to the visible light spectrum circumvents this problem altogether, opening up a massive spectrum of unused frequency for commercial use. Lastly, the use of Li-Fi inherently provides a high level of security due to the predictable and easily retractable paths that light travels through. Without being able to physically place a receiver in the path between the access point and the user device, there would be no way to intercept transmitted data [4]. These factors greatly enhance the potential for commercialized Li-Fi usage in the near future.

According to this industrial trend, our undergraduate students must be explored and experienced with that state-of-the-art technology. Experience in setting up a Li-Fi system and working with its protocols can give a massive advantage to any student looking for an edge. The demand for this technology will only grow, and we have shown that it is not too advanced to be brought into the classroom.

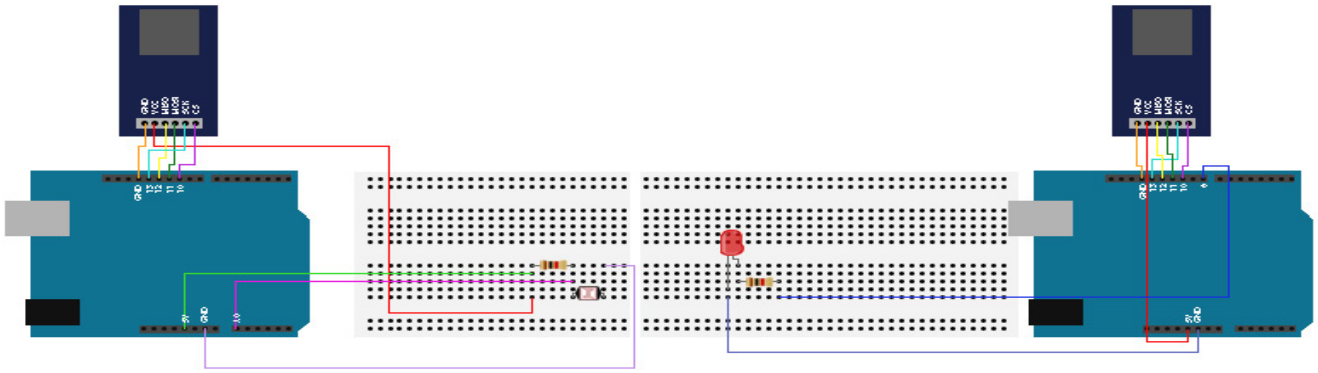

Fig. 1. *LiGET* System Architecture

III. LiGET

LiGET requires two *Arduino Uno Rev3*s, two *Arduino* SD card modules, one light dependent resistor (LDR), one clear light emitting diode (LED), two breadboards, a 100ohm resistor, a 1000ohm resistor, and 17 wires as shown in Figure 1. Other *Arduino* variations may work as well, however the only tested build uses *Arduino Uno*s. One *Arduino* needs to be setup as the client, or receiver, and the other *Arduino* needs to be setup as the server, or transmitter.

The server is easier to setup due to it requiring less wiring than the client. To setup the LED, we use two wires, the 100ohm resistor, and a GND pin and pin 8 on the *Arduino*. The *Arduino* pin number can be any of the digital pins, except for 11, 12, and 13, because those will be used for the SD card module. One wire goes from the GND pin on the Arduino to an empty spot on the breadboard. The positive pin on the led goes to a space on the breadboard in the same row that the wire from the GND is. The negative pin on the LED goes to a different row. One end of the 100ohm resistor comes from the same row as the negative pin on the LED, and the other goes to a new row. The end of another wire then goes from that new row with the resistor in it to digital pin 8 on the *Arduino*.

The client is slightly more difficult to set up. To set up the light dependent resistor, we use the second breadboard, three wires, the 1000ohm resistor, a GND pin, the 5V pin, and pin A0. It should be noted that this setup is on the second Arduino. One wire is connected to pin A0, another wire is connected to GND, and the last wire is connected to 5V. The LDR needs to be placed on the breadboard, with both of its ends in different rows. The other end of the wire connected to GND goes to one of the same rows that the LDR is connected to. The other end of the wire connected to A0 goes to the row that has the opposite end of the LDR than the one that is connected to the GND pin. The wire that is connected to the 5V pin goes to a new row on the breadboard. One end of the 1000ohm resistor goes to that row, and the other goes to the same row that the wire connected to A0 is in.

The last wiring step is to setup the SD card modules. These will be mostly the same on both *Arduinos*. The SD card module can be put into the breadboard so that each pin is on its own unique row. The first pin on the module is the GND pin. This pin just needs a wire that goes from another point in that row to a GND pin on the *Arduino*. The VCC pin on the Module needs a wire that goes from the 5V on the *Arduino* to an empty spot on the breadboard that is in the same row as the VCC pin on the SD card. On the receiver, the SD card needs the 5V pin which is also used for the LDR. All this means is that an additional wire needs to go from the same row that the wire that is connected to the 5V pin on the *Arduino* to the VCC pin on the receiver *Arduino*. Another wire goes from the MISO pin on the module to digital pin 12 on the *Arduino*. One more wire goes from the MOSI pin on the module to pin digital pin 11 on the *Arduino*. Another wire goes from the SCK pin on the module to digital pin 13 on the *Arduino*. One last wire goes from the CS pin on the module to digital pin 10 on the *Arduino*. The CS pin is the only pin that the user can choose, and must be defined in code. We used pin 10, but any digital pin that isn't being used by the other parts of the SD card module or the LED can be used.

The implementation of sending a file is done by reading bytes from a file, and then sending those bytes by turning them into its 0s and 1s and turning the LED on if we want to send a 1 and turning the LED off if we want to send a 0. Getting values from the light dependent resistor is done using an analog read. The analog read returns values from 0 if it detects high light, to 1023 if it is in complete darkness. Before a file can be sent, the receiver need to calibrate based on the ambient light wherever it is. It calibrates by doing an analog read 100 times and calculating the average. We then subtract of a

```
byte getValue()
{
  unsigned long interval = INTERVAL;
  int d[dsize]; //array to store incoming 8 bit values
  bool startBitReceived = false;
  while(!startBitReceived){
        sensorValue = analogRead(sensorPin); //Takes 100 microseconds
        if(sensorValue < threshold){ //If we detected start bit
          timer0 = 0;
          startBitReceived = true;
        }
      }
      //Just delay here until we get to the middle of the start bit
      while(timer0 < HALF){}
      timer0 = 0;
      //Get bits based on timer0
      for(unsigned long i = 0; i < dsize; i++){
        while(!moveToNextBit){
          if(INTERVAL - timer0 < 10){
            d[i] = analogRead(sensorPin); //Takes 100 microseconds
            timer0 = 0;
            moveToNextBit = true;
            interval = interval - 100;//Adjust interval because of how long analog read takes
          }
        }
        moveToNextBit = false;
      }
      byte recVal = convertToDecimal(d);
      //Wait on stop bit
      timer0 = 0;
      while(timer0 < INTERVAL)
      {}
      return recVal;
}
```

Fig. 2. Method for Receiving Bytes

portion of this average, and that becomes the threshold for a 0 or a 1. If the analog read returns a value that is lower than the threshold, it is a 1. If the analog read returns a value that is greater than or equal to the threshold it returns a 0.

The receiver then uses a software, universal asynchronous receiver-transmitter (*UART),* to receive values from the transmitter as shown in Figure 2. There needs to be a predefined bit time in both the receiver and transmitter before any transmission can be sent. We call this predefined bit time INTERVAL, because it determines the delay between analog reads to get values for each byte. We defined our bit time to be 30,000 microseconds. We found that any value lower than that would occasionally cause errors in the transmission. Once the receiver is started, initialized the SD card and determined the light threshold, it enters a polling state where it is constantly looking for the start bit. When it receives the start bit, it waits for an interval equal to 1.5 times the bit time. This is enough of a delay to get to the middle of the transmission of the first bit. It then does an analog read to get the first bit of the byte. It then waits one more bit time to get the second bit of the byte, and so on until all 8 bits have been received. It then waits one last time for the stop bit.

This method can be used to send individual bytes. To send a file, the receiver needs to know when to stop receiving bytes to write to the file. To do this, the first four bytes of the transmission are the file size. After that the transmitter sends 1 byte for the size in bytes of the name of the file. The receiver then uses that to receive the name of the file, create the file, and then uses the size of the file in bytes that it read in earlier to know when to stop receiving bytes and write to the file. Once the receiver has finished receiving bytes, it closes the file on the SD card.

A. Fining the bit time

We wanted to know at what bit time the receiver would be able to keep up with the transmitter and avoid errors. The receiver has much more overhead than the transmitter, so if the bit time is too low, the receiver may fall behind. If the receiver detects a start bit too late, it might wait too long and miss the next transmitted bit. We tested by sending a small file of size 43 bytes 10 times for each listed bit time. If it failed at all, we increased the error rate by 10%.

TABLE 1. BIT TIME ERROR PERCENTAGE

Bit Time (microseconds)	Error
50000	0%
40000	0%
30000	0%
29000	20%
28000	60%
27000	90%
26000	100%

There is a dramatic increase in error as the bit time decreases as shown in Table 1. As a result, the recommended bit time for this setup is 30000 microseconds. That is the bit time we used in our final implementation and it has worked consistently without any errors.

IV. CONCLUSION

Our project was primarily meant to bring the emerging technology of Li-Fi down to a level that undergraduate students could work with in a classroom setting. If students do not need to go through the work of figuring out how each method and the software *UART* needs to work themselves, this project can be completed within a few hours' maximum. It could exist as a hands-on lab exercise for wireless based college courses.

The project is expandable. At the moment we have only implemented single directional communication. If we were to have bidirectional communication, we could add error detection and handling. We could also start using infrared LEDs so that the project does not use the visible light spectrum. To improve the speed, we could switch to doing a binary read instead of an analog read, however that would require additional circuitry that we were not prepared to setup at the time of the project. If we were to move away from *Arduino* to something else that may increase the speed as well. The *Arduino* is great for small projects like this, however when timing needs to be done down to the microsecond the *Arduino* is not fast enough. A device with a faster processor may work better in the future. What may also improve speed is solving the *UART* problem with hardware instead of software. Typically, *UART* is a hardware problem, and if we could have device that reads in infrared values and returns bytes to our code it could drastically improve the speed of this implementation of Li-Fi.

REFERENCES

[1] H. Haas, L. Yin, Y. Wang and C. Chen, "What is LiFi?," in *Journal of Lightwave Technology*, vol. 34, no. 6, pp. 1533-1544, 15 March15, 2016

[2] M. Leba, S. Riurean and A. Lonica, "LiFi — The path to a new way of communication," *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, Lisbon, 2017, pp. 1-6.

[3] R. Mahendran, "Integrated LiFi(Light Fidelity) for smart communication through illumination," *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, Ramanathapuram, 2016, pp. 53-56.

[4] S. Kulkarni, A. Darekar and P. Joshi, "A survey on Li-Fi technology," *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, 2016, pp. 1624-1625.