Light-Fidelity - Electromagnetic data transmission using visible light

Man-Leong Chan
School of Electronic and Computer Science
University of Southampton
mlc1g14@ecs.soton.ac.uk

***Abstract*— Lifi has amazed the industry in terms of its high-speed performance and its subtleness of its application. However the industry remains skeptical on the possibility of it being a real applicable technology in the near future. In order to investigate the technological challenges that Lifi is facing, we tried to implement the very basic version of a Lifi system.**

The goal of this project is to successfully transmit a set of ASCII characters from the transmitter to the receiver using Lifi. This project will cover some aspect of physical (L1) and data link (L2) layer of the OSI model, aiming to achieve a simple unidirectional communication, limited by time and cost.

I. INTRODUCTION

Lifi or Light-Fidelity is a high-speed network communication technology using visible light. While Lifi and WiFi both transmit data electromagnetically, Lifi is able to achieve a much higher theoretical bandwidth due to the high frequency of visible light in Lifi compared to a lower frequency of radio waves used by WiFi. While researchers at University of Oxford had been able to achieve a staggering 224 Gb/s [3], the goal of this project is to investigate the basis of transmitting data through air using visible light at a low cost.

In the physical layer, we experimented with ways to transmit simple binary signal from one end to another in order to determine the hardware that is required. In the data link layer, we explored methods to convert binary to ASCII and vise versa in real time, as well as a methodology for error handling. All together it allowed us to define a simple networking protocol suitable for low-powered unidirectional communication.

II. PHYSICAL LAYER

Both the transmitter and the receiver is based on a programmable single-board computer NXP FRDM-K64F [5] with 120MHz processor and 1 MB of flash memory. The transmitter sends data by driving a green LED using a 3.3V GPIO pin *(Fig. 1)*, while the receiver is connected to a TEPT5700 Ambient Light Sensor[6] (photodiode) which is most sensitive to the green light region.

Transmission speed is limited by the clock rate of the K64F board which can only drive the LED signal or the photodiode at a lower speed. The lack of power provided by the GPIO pins also contributed to the lower speed as the LED will not be as bright compared to a higher voltage output. This increase the photodiode's challenge to detect clear signal, therefore likely to create more signaling or interpretation error.

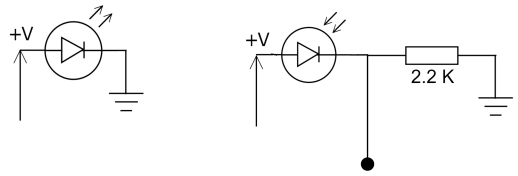

Fig. 1. Circuitry of the transmitter (left) and receiver (right)

A. Sensing

A popular Lifi implementation in the industry is to utilize all the indoor lighting such that it provides a stable single source of signal in the form of visible light. In this project however it is not viable for us to find indoor light sources capable of oscillating at a high frequency. Instead we have uses an LED as the source and photodiode as the sensor.

The LED is connected to a 3.3V GPIO digital-out, at this level of voltage the LED does not glow very bright compared to the labs background light. Therefore we have to enclose the area between the LED and the photodiode in order to minimize the interference. To further enhance the signal level, the colour of LED is carefully chose according to the peak sensitivity of our TEPT5700 Ambient Light Sensor[6] at 560 nm which is a lime-green colour.

B. Performance

While the performance of the transmitter and receiver is not part of the project scope, it is crucial for investigating a viable transmission frequency. The maximum theoretical throughput of the transmission is limited by the poorest performing hardware in the entire system, it would also define the operating speed between both ends. Therefore both transmitter and the receiver has undergone performance testing measure using a high-performance oscilloscope.

The transmitter with an LED is driven by a 120MHz processor. It has a maximum driving speed of 794.9 KHz turning an LED on and off, however the performance was dramatically reduced when it has to process data along with transmission.

	Frequency	Period
Binary Pair	794.9kHz	1.258µs
Mean	794.7kHz	1.258µs
Minimum	795.3kHz	1.257µs
Maximum	152.9kHz	1.259µs
Std-Dev	152.9Hz	242.2ps

Fig. 2. Signal frequency and period tested using an oscilloscope

III. Digital Modulation

A. ADC

Our system relies on On-Off keying to transmit data from transmitter to receiver. It uses unipolar encoding where an LED at full brightness represent *one* and off representing *zero*. Other implementation where zeros are represented half the brightness of representation of ones is unsuitable, as brightness is measured relative to the surrounding environment. If the photodiode is to determine the exact brightness transmitted from the LED, the distance between both transmitter and receiver must be kept constant[4]. Such implementation would require very accurate laboratory set up and limits the mobility of the entire system.

When the transmitter is sending unambiguous binary signals, the photodiode receiver will receive an analog value between 0-255 according to the level of brightness it detects. This requires an analog-to-digital(ADC) module to determine whether the LED has transmitted *one* or *zero*. In our implementation, we have determined a threshold such that any value above the threshold will represent *one* and any below would represent *zero*. Such threshold is not a fix value and is determined before any signal is sent. Depending on the surrounding background brightness, the threshold is increase if the background brightness is high or decrease if the brightness is low.

B. Binary Encoding

We have discussed the benefit of unipolar encoding above, however leaves ambiguity between *zero* when the LED is off and when no signal is being sent. Therefore the receiver will not be able to differentiate between '0110' and '110', as the '0' in the first string is represented by no light emitted from the LED.

In order to tackle this problem, we have employed a binary encoding to represent *one*s & *zero*s. Instead of having LED on & off representing the values directly, we encode *one*s & *zero*s with on/off patterns.

On-Off Pattern	Binary
ON-OFF-OFF	0
OFF-OFF-**ON**	1

Fig. 3. Binary encoding schema

If we use dot-and-dashes with '●' representing the LED being on and '_' LED off, we could demonstrate how the binary encoding would avoid ambiguity creating by unipolar encoding. Again with the same pair of string: '0110' & '110'

Data	Encoded data
0110	● _ _ _ ● _ _ ● ● _ _
110	_ _ ● _ _ ● ● _ _

Fig. 4. The string '0110' represented using our binary encoding schema

IV. Data-Link Layer

A. Error Handling

In many bidirectional communication, error is detected by the client and resolved by sending positive or negative acknowledgements. In a unidirectional communication however, the transmitter have no idea whether the signal is received by the receiver or if the packet content is valid. Therefore a good forward error correction (FEC) technique is important such that if the receiver happens to have receive the data, it can validate and possibly recover the content automatically. [2]

There are many factors to consider when choosing a forward error correction technique:

Nature of most error: Most error occur due to sensors thread halt, this occur when the sensors CPU get occupied and stop sensing data for a short period of time. If the halt time is short, FEC with interleaving would help recover the lost bits.

Frequency of the error: Halt happens periodically but the period is unknown. Therefore a simple parity bit technique may not be sufficient to detect, never mind correct the error. The error detection ratio is highly important in case the error rate is high.

Real-timeness of the data output: We try to achieve real-time data receiving on the sensor side and therefore error correction will need to be handled as the binary data comes in. Therefore the length of the FEC message length should be as short as possible. A short FEC message allow us to process and correct data as soon as possible instead of queuing up a long FEC message.

Throughput: The ultimate throughput is not our consideration, so we have been very generous on error-correction-bit to data ratio

Result

Binary repetition code[1] is one of the best and simple solution for FEC over a binary erasure channel. It has a

hamming distance of **n**, and an error correcting capacity of

$$\left\lfloor \frac{n-1}{2} \right\rfloor$$

In the case of this project, we have allow extra redundancy with repetition code length of 21, which is equivalent to a (21,10) hamming code. This level of redundancy allow us to tolerate sampling error that may occur. Binary repetition code does not rely on interleaving, this allow us to process the data straight away for real-time output. Therefore there is very minimal data which is stored while the receiver is listening to the incoming signals.

V. Encoding & Decoding

We have discussed the methodology of techniques we have used, now it is time to put together the workflow for both transmitter and receiver.

A. Transmitter

Encoding the signal is rather straight forward as we assume the data for transmission is fully given to the transmitter before the encoding begins.

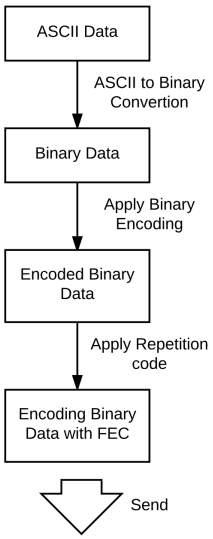

Fig. 5. Work flow for encoding ASCII character set

It starts with ASCII characters encoded into 7-bits binary data (Fig. 5). Binary encoding is then applied to every bit in the binary data, where *zero* is encoded into '100' & *one* into '001'. Lastly FEC repetition code is applied where every encoded binary data is repeated 21 times before the encoded binary data with FEC is sent over the network.

B. Receiver

Decoding receiver's information is a little more complicated compared to transmitter as data can arrive at anytime. To make the task more complicated, receiver tries to decode the information as soon as they arrive such that it can minimize the data stored.

The receiver start by decoding repetition code (Fig. 6), the benefit of such FEC code is that it can be decode

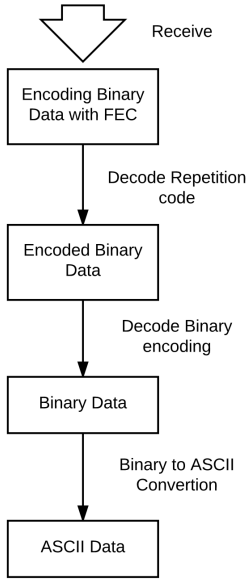

Fig. 6. Work flow for decoding incoming binary data back into ASCII

by calculating the statistical mode of the received bits according to the length of the repetition code.

The receiver than has to decode the binary data. Which is a simple pattern matching process for every set of three bits and map '100' & '001' to *zero* & *one* respectively. E.g the encoded string '001100' would produce the binary '01'. When there is an error occurring in the data, the unrecognized data would be thrown away. For example the string '010100', the pointer will point at the first 3 bits '010' which is unrecognized, the first bit '0' will then be thrown away leaving the string '10100'. The process repeats again as the 3 bit string '010' is again unrecognizable. The same happens for the next bit leaving the string '100' left. The string now is recognizable and will be decoded into '0' in binary.

Finally, translating the decoded binary data into 7-bit ASCII very straight forward.

VI. Remaining Work

The biggest challenge in unidirectional communication is clock synchronization. A synchronized clock between the sender and the receiver allows the signal to be interpolated at the correct time. Without the correct synchronization, the sender and receiver pair will soon be interpolating data at a different rate and producing error due to time drift. This is a common problem across all networking hardware and expensive time synchronization device are integrated into high-speed networking solutions.

In our implementation there is currently no time synchronization method applied to either of the device. We rely on frequent resetting of both device and the benefit

of having 2 of the same hardware on the sender and the receiver side, because both device has the same CPU and clock rate, the time drift problem could be contained for a limited amount of time.

To make matter worst, unidirectional communication does not allow our receiver to give any time drift feedback. Therefore if the receiver does detect the effect of time drift, there is no possible way of notifying the sender on updating the time delta between them. The only possible way to achieve a certain level of clock sync without using external clock-sync device or bidirectional communication is to create another unidirectional communication. This extra unidirectional communication could send the clock of the transmitter by sending the clock rate of itself at the same time, a signal that sent the highest oscillation rate driven by the transmitter's CPU. This way the receiver would be able to tell the rate of the transmitter at any given time.

VII. Conclusion

Throughout this project, we have demonstrated many techniques on how to reliably transmit simple ASCII characters 'over the air'. It shows how difficult it could be to simply transfer data even at a lower rate using visible light. As mentioned in my remaining work, this project is by no means a household Lifi product ready to be deployed. However the project does give some indication on the challenges that we face in networking technologies of this nature.

ACKNOWLEDGMENT

I would like to thank my partner Devashish Dixit on his contribution and motivation at all stages of the project, as well as Prof Kirk Martinez & Graeme Bragg for their guidance in this project.

References

[1] M Bossert. Channel coding for telecommunications. *Getcited.Org*, 1999.

[2] George C Clark and J Bibb Cain. *Error-correction coding for digital communications*. 1981.

[3] Ariel Gomez, Kai Shi, Crisanto Quintana, Masaki Sato, Grahame Faulkner, Benn C. Thomsen, and Dominic O'Brien. Beyond 100-Gb/s indoor wide field-of-view optical wireless communications. *IEEE Photonics Technology Letters*, 27(4):367–370, 2015.

[4] Institute for Astronomy University of Hawaii. Joshua E. Barnes. The Inverse-Square Law, 2003.

[5] NXP. NXP FRDM-K64F, 2014.

[6] VISHAY. TEPT5700 Ambient Light Sensor.