SkyFi

The Wireless Networking Paradigm

Dhruva Kaushal, *Electrical and Computer Engineering, Carnegie Mellon University*
Syed Raziq Mohideen, *Electrical and Computer Engineering, Carnegie Mellon University*
Sribhuvan Sajja, *Electrical and Computer Engineering, Carnegie Mellon University*

***Abstract*—SkyFi is a system which uses wireless network using visible light communication (VLC) at the physical layer to transfer data. Four nodes are connected to a hub router capable of receiving multiple transmissions simultaneously using Overlapping Pulse Position Modulation. The nodes are running a multiplayer version of Pong, and the hub acts as a game server. We aim to achieve at least 1.6Kbps of system goodput for responsive gameplay, and we use SkyFi to demonstrate the reliability, performance, and robustness of VLC for similar use cases.**

***Index Terms*—FPGA, Gaming, Modulation, Networking, Visible Light Communication, Wireless Communication**

I. Introduction

Wireless data transfer is a key component of communication between the numerous devices that surround us today. With the advent of IoT and smart-devices, the wireless networks that support them also need to scale up proportionally. Current wireless networks such as Wi-Fi and cellular data are based on data transmission using radio waves. When the number of devices increases, the fixed bandwidth makes data transmission slower as radio waves are a small part of the electromagnetic spectrum. These networks are also susceptible to security threats since radio waves can pass through walls. In places like hospitals, airports, and military bases, radio waves are intentionally interfered with so that data transfer is restricted. With our capstone project, we aim to address some of the above key technical challenges using visible light since recent research predicts visible light communication to have high transmission speeds, orders of magnitude more bandwidth than radio waves, suitability in areas sensitive to electromagnetic interference, and increased security and directionality.

SkyFi is a system that will demonstrate the benefits of visible light communication. It supports full-duplex communications between nodes and a central hub/router. Multiplayer gaming is an intuitive use-case, as it requires low latency, sufficient data rates, and simultaneous transmission and reception. As VLC is still a nascent technology, there is ongoing research on how to most effectively exploit the available spectrum. Based on current understanding, we are aiming for a reliable 1.6Kbps system throughput, with full support for reception of concurrent transmissions at the hub. For a smooth gaming experience, we need a refresh rate of at least 24Hz, taking into account all communication between nodes and the server.

II. Design & System Description

Refer to 15 for the complete system level block diagram. The overall design can be broken down into three sections.

A. The Physical Layer

In the data-transfer and communication system, the physical layer is first and lowest layer in a network. It is the fundamental layer which underlies the higher level functions in the network. It consists of the electronic circuit transmission technology of a network. For our project, this represents the circuitry which would transmit the signal, in the form of bits, from the transmitting FPGA to the receiving FPGA. The goal is to create two circuits: a transmitter and a receiver. The transmitter will take a digital signal from the GPIO pin of the FPGA and broadcast it to any listening receiver circuits by modulating an LED. This idea of modulating an LED forms the backbone of the physical layer in visible light communication. The receiver circuit is responsible for picking up any broadcasted signal from a transmitter circuit and reconstructing the transmitted signal successfully. This receiver would be connected to the GPIO pin of the receiver circuit which would interpret the received signal.

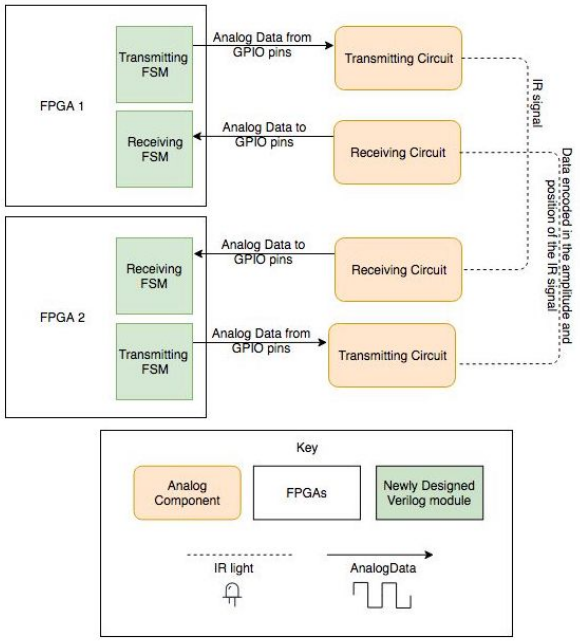

Fig. 1. Block Diagram depicting interaction and signals between components

Figure 1 depicts the interactions between the various components of the physical layer which enable a full duplex communication between two distinct nodes. The transmission

FSM of the transmitter FPGA drives an analog signal which represents the data we want to broadcast to receivers. The analog signal flickers the LED in the receiver circuit, thus broadcasting simultaneously to all receivers in the field of transmission of the LED. The photodiode in the receiver circuit which is in the broadcasting field will generate a current corresponding to the wavelength and intensity of the incident light. This in then reconstructed and sent to the receiver FPGA for interpretation.

B. The Data Link Layer

The data link layer is the bridge between the physical and application layers of the networked system. It packages bits from the application into frames which are sent out as pulses to be emitted by the LEDs, and it receives other nodes' pulses through the photodiode and interprets them as packets of data. This is referred to as modulation and demodulation and it is where a great deal of research is being done regarding visible light communication.

Our modulation technique is inspired by a Dartmouth project named DarkLight [1]. It is called Overlapping Pulse Position Modulation (OPPM) and it is an advanced version of Pulse Position Modulation. Bits are transmitted as n-bit symbols, and each symbol is represented as a single pulse of length t_{pulse} within a period of time t_{symbol}. Each symbol period is divided into 2^n time slots of length L. Data is encoded by the time slot in which the pulse is located (hence, pulse position). Unlike regular PPM, OPPM is only concerned with the rising edge of the pulse, allowing for $t_{\text{pulse}} > L$ (hence, overlapping). In our design, we will be using 8-bit symbols and a time slot width of 16.22 μs, giving us a throughput of 1.926 Kbps. Refer to Section III-B for information regarding how these values were determined.

On top of the modulation technique is any additional encoding/decoding required. It is possible that interference or ambient noise may cause pulses to be misinterpreted, so we would like higher-level control of potential errors. One possible technique is Reed-Solomon (RS) error correction. RS codes are widely used in both data storage and data transmission. They are suitable for VLC because they support short packets, have hard decision decoding support, are low in complexity, and are suited for transmission purposes. Typical RS codes have a code rate of 50-90% [2].

Figure 2 summarizes the high-level design of the Data Link Layer. Most of the modules will be written from scratch, but there are existing implementations of common Forward Error Correction (FEC) techniques which we can adapt to our board and data sizes.

C. The Application Layer

The application layer is the layer closest to the end user. We wanted an application which demonstrates the working of our network while also showcasing its real-world usefulness. Gaming is one such use case that involves real-time duplex communications between multiple nodes and exercises many useful aspects of networking. We chose the arcade game, Pong,

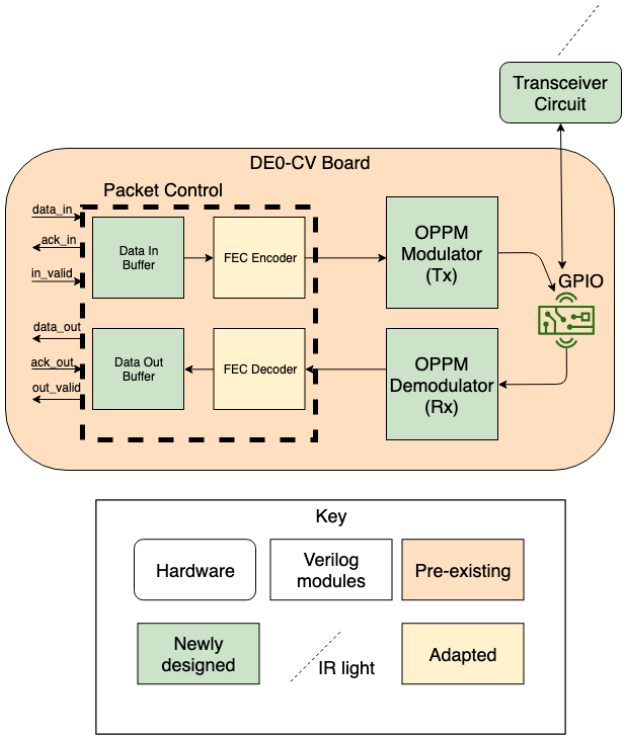

Fig. 2. Data Link Block Diagram

to run on the application layer of SkyFi. Pong is a two-player game, where each player controls a paddle on opposite sides of the screen. A ball is set into motion in between the paddles. The objective of each player is to use their paddle to bounce the ball to hit the wall behind the opponents paddle. Apart from being easy to understand and play, Pong is a very lightweight application which requires simple graphics and user input.

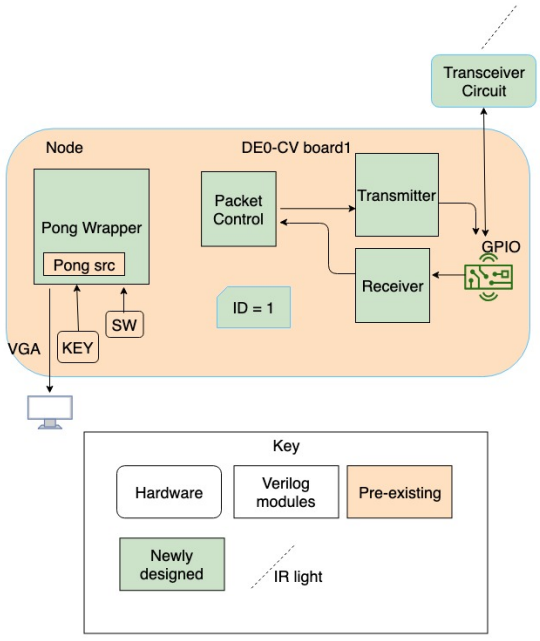

Fig. 3. Node Block Diagram

Figure 3 shows the block diagram for the application. This will be replicated on each node. The Pong wrapper module will contain the bulk of the logic implementing Pong. This module is responsible for interfacing with user input and displays the state of the game on a monitor through a VGA cable.

The Pong wrapper will interface with the packet control to send and receive data. The packet control module has two functions: One, it will look at the data and control signals received from Pong wrapper and encode packets containing the destination and forward it to the Transmitter module. Two, it will decode the packets received from the Receiver module and forward the data to Pong wrapper.

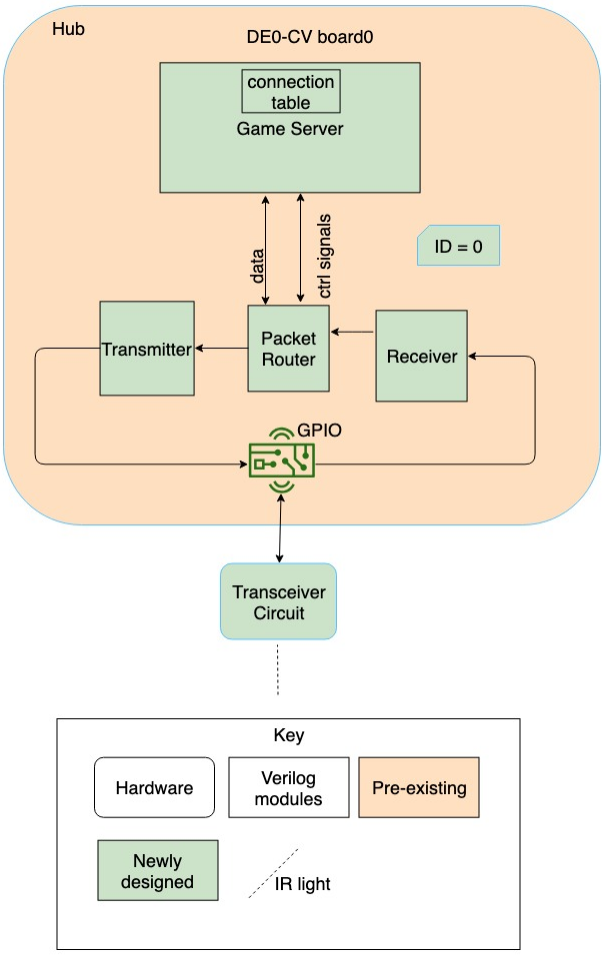

Fig. 4. Hub Block Diagram

Figure 4 shows the block diagram for the application running on the hub board. The game server module will contain information on all player matchings as well as ball and paddle information for each player. To setup a new game, the nodes must communicate with the server to synchronize their state. When the game starts, each player will send their paddle location to the server. The server will then transmit the opponents paddle location and ball location to ensure that state is maintained.

III. Design Trade Offs

A. Application Constraints

Our application drives our design choices. We want multi-player Pong to run smoothly, so the data transmission must be capable enough. To maintain a smooth gaming experience, the game state will need to update at a rate of at least 24Hz for each node. This means that the game server will need to send data about the ball location and opponent paddle position to a node at least once every 0.042s to ensure smoothness. The X and Y coordinates of the ball and the paddle location of each player is encoded in 11 bits. So, we will need to transfer 44 bits of data between the server and nodes every 0.042s. This requires a network speed of at least 1.047Kbps.

B. Modulation Parameters

In Overlapping Pulse Position Modulation, the bits per symbol n, symbol width t_{symbol}, and time slot width L are all variables that can determine the data rate or throughput T. They are related by Equation 1:

$$T = \frac{n}{t_{\text{symbol}}} = \frac{n}{2^n L} \tag{1}$$

In our design space exploration, we set n and L as independent variables and plotted the resulting throughput in Figure 5:

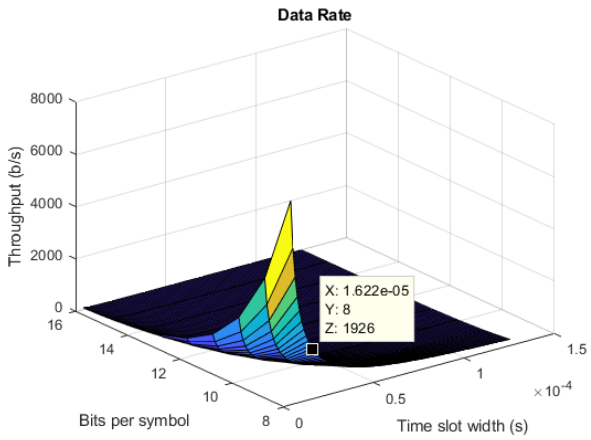

Fig. 5. Data Rate

In the domains we specified (8-20 bits, 4-125 μs) we find that minimizing n and L increases T. Naively, one would assume that the best performance is achieved with $n = 1$ and L as low as the hardware can support (which is equivalent to On Off Keying). However, the need to support multiple transmissions with a single receiver complicates matters.

In order to distinguish between multiple transmitters, the receiver must be able to accurately determine when the rising edge occurs. Depending on the sampling rate, there is an offset in the measured rising edge from the start of the intended time slot. In [1] the distribution of offsets tends to follow a Gaussian distribution. For a 1 MS/s sampling rate, the offset range $\theta = 1\mu$s covers at least 90% of this distribution.

Once you are able to determine the time slots in which pulses are occurring, you can then demodulate the signal

by converting the pulse positions into n-bit symbols. This becomes more complicated when there are multiple unsynchronized transmitters (which is our case because each node is a separate FPGA). It is *likely* but not **guaranteed** that different transmitters' clocks (and thus pulses) are unaligned. Thus, we must consider the probability of two transmitters sending a pulse within the same time period (which is the offset range θ). Such probability p is given by Equation 2:

$$p = 1 - (\frac{L - N\theta}{L})^{N-1} \tag{2}$$

This equation, proven in [1], demonstrates the effects of slot width, number of devices, and offset range have on the probability. In SkyFi, $N = 4$ and $\theta = 1\mu$s so Figure 6 demonstrates the relationship between L and p.

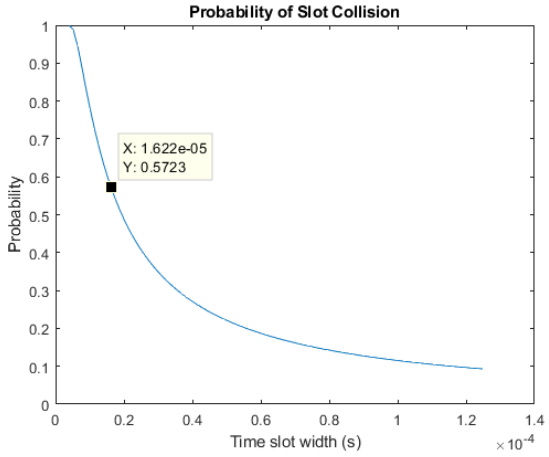

Fig. 6. Collision Probability

Although DarkLight's single-link prototype uses a slot width of 3.2μs to achieve a 1.6Kbps throughput, when multiple transmitters are in the vicinity, there is a high probability of slot collision. Thus, there is a trade-off between data rate and reliability.

To determine the values appropriate for SkyFi, we estimated the "goodput" (throughput of actual data) in Equation 3:

$$G_{\text{system}} = NTR(1 - p) \tag{3}$$

T is a function of n and L, and p is a function of L. R is the ratio of data bits to total bits and is a factor dependent on the error correction and was set to 0.6 as a conservative estimate. G was plotted as a function of n and L in Figure 7:

In the specified domains, we find a local maximum at $n =$ 8bits and $L = 16.22\mu$s, resulting in $T = 1.977$Kbps. This meets our application requirements.

C. Equipment for the Physical Circuit

In order to support the above mentioned data-rate, the physical circuit needs to have parts which are able to operate at that frequency. Specifically, the IR LED, photodiode and the op-amp should be able to operate as expected.

As our IR LED, we chose the TSHF5410. The key rationale for choosing the TSHF5410 was that it has a cut off frequency

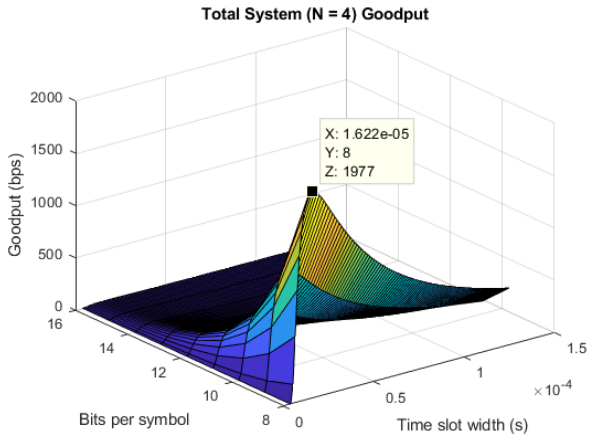

Fig. 7. System Goodput

of 12MHz which is far larger than the frequency we intend to operate at. Additionally, it was cheap and easily available in the 18-220 lab.

Initially, we planned to order an off-the-shelf transimpedance amplifier to use in the receiver circuit. However, we found that we get more control over the voltage levels by assembling this ourselves. Most of the equipment we need is available in the CMU labs. The only component we will order is the operational amplifier. We chose the LM 7171 op-amp. The choice of the this op-amp was dictated by the gain-bandwidth product value. We know that op-amps have a finite bandwidth. A fairly practical approximation of the frequency response is the integral of the gain. The gain of an op-amp is inversely proportional to its frequency. As an example, an op-amp with a gain-bandwidth product of 600MHz has a gain of 6 at a frequency of 100MHz. Another factor that we needed to consider was the slew rate of the op-amp. The slew rate is defined as the maximum rate of change of the output voltage in response to the input voltage. Since we are operating at high frequencies, we need the slew rate to be as high as possible. In an ideal op-amp, the bandwidth and slew rates are both infinite. For our purposes, the LM7171 which has a gain-bandwidth product of 20MHz as well as a slew rate of 4100V/s, works perfectly and is strictly higher than needed.

The QSD2030F photodiode in the circuit acts as a current source that peaks for the 880nm wavelength which corresponds to the IR part of the electromagnetic spectrum. This was another factor that determined our choice of the IR LED as the transmitter. The QSD2030F has a rise and fall time of 5ns which means it can theoretically operate at frequencies as high as 50MHz which is much higher than the frequency we intend to operate at. It also has a forward voltage of 1.3V which is almost equal to the forward voltage of our IR LED.

Note that having the physical circuit support frequencies higher than what we expect to be operating at allows us to make further speed optimizations later on (as a stretch goal) without having to change the design of the physical circuit later on.

D. FPGAs

FPGAs serve a dual purpose for us. Firstly, they allow us to have a data processing module and a data transfer module on the same board. Since the processing capacity of an FPGA is high and it can run both modules in parallel, we can safely put a high load on the data transfer module which we know will not constrict our speed in the data processing module. Secondly, they allow us to parallelize the sending and receiving processes. The base clock speed of the FPGAs we have is 50MHz. We do not use a microcontroller as they cannot easily support parallelization of multiple processes on the same device. In order to counter this we would need to select a microcontroller with a proportionally higher clock speed. This option thus turned out to be expensive and unfeasible since we needed five nodes. All of the above factors led to the decision to use FPGAs as our nodes.

IV. Implementation Plan

A. The Physical Layer

The way we plan to implement this is by constructing the physical circuits onto a protoboard and wiring them to the FPGA. Each node will have both a transmitter and a receiver circuit.

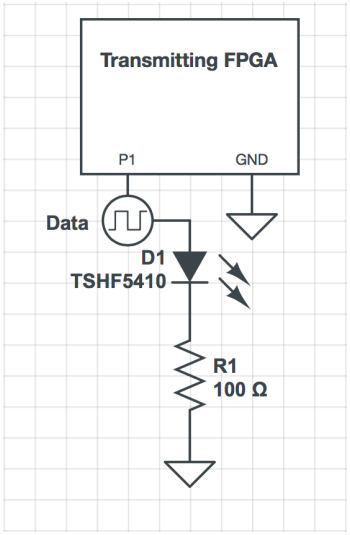

Fig. 8. The Transmitting Circuit Diagram

The transmitting circuit is a simple IR modulation circuit which flickers an IR LED. The on voltage of the selected IR LED is 1.4V. This means that for the bit 0 for which the FPGA will supple a 0V, the LED will be off. Conversely, for the bit 1 for which the FPGA supplies 5V (configurable), the LED will be on. The circuit is thus able to broadcast the data as a stream to any listening receiver circuits inside its field of transmission. The intensity of the transmitted light is proportional to the voltage applied across the circuit. The only device that requires a power source in this circuit is the FPGA. The rest of the circuit is driven by the power supplied by the FPGA.

The receiving circuit is responsible for picking up any signal being broadcast in the field of the photodiode and

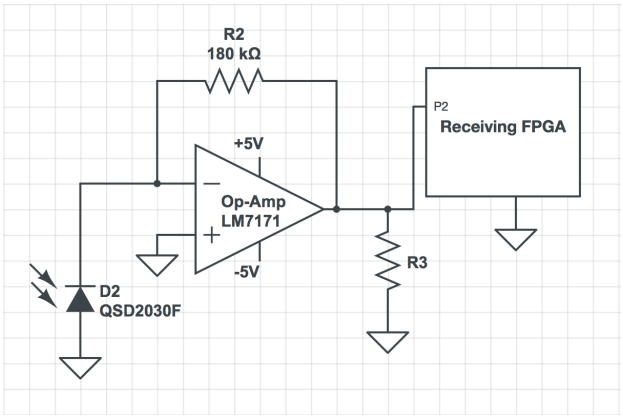

Fig. 9. The Receiving Circuit Diagram

successfully reconstruct the originally sent signal. The way the above circuit does this is by using a trans-impedance amplifying circuit with the photodiode acting as a current source. The current source produces a potential difference which gets amplified at the output of the op-amp. The GPIO pin of the receiving FPGA is connected in parallel with a small resistor at the output node of the op-amp. Hence, it is able to read the amplified signal. The photodiode produces a current proportional to the intensity of light incident on it. Since the photodiode is largely in phase with the transmitting IR LED, as long as we choose components with the correct bandwidth, we know for certain that when the IR LED is transmitting a 0 bit, the photodiode will not produce any current and thus the voltage will be 0 at the output of the op-amp. Similarly, when the IR LED is transmitting a 1 bit, it has a voltage of 5V across the circuit. The photodiode will produce a proportional current and according to our calculations, the output node of the op-amp will have a value of 3.3V which will be interpreted as a 1 bit by the FPGA.

B. The Data Link Layer

The data link layer will be implemented on the FPGA. All FPGA development (including at the Application Layer) will be done using Synopsys VCS to develop and Intel Quartus to synthesize our hardware designs. Each module (modulation/demodulation, encoding/decoding, etc.) can be developed in isolation and tested before being integrated as part of higher-level constructs.

The OPPM Modulator will latch onto n bits at the start of each symbol period. It will contain internal counters to count the number of FPGA clock ticks per time slot and the number of time slots per symbol period. At the correct time slot (determined by the value of the bits), the Modulator will send a pulse of length t_{pulse} through the GPIO pins.

The Demodulator is a bit more involved, because of the potential to demodulate multiple transmissions at once. Darklight provides a viable algorithm (Figure 10) to determine the source of an arbitrary pulse [1]. Due to a lack of inter-loop dependencies, the imperative algorithm can be converted to hardware with relative ease, whether manually or with tools such as Vivado HLS.

input : 1) t, start time of the rise edge to be classified;
2) $T[i]$, start time of the most recent rise edge of stream i;
3) L: time slot width;
4) $[\underline{\theta}, \bar{\theta}]$, rise edge offset range.
output: s, the stream number that the new rise edge is classified to.

```
n = 0                                          /* # of streams */
cnt = 0
k = 0
for i ← 1 to n do
    m_i = round((t − T[i]) / F_s)              /* closest integer */
    q_i = t − m_i × F_s
    if (q_i > θ_) AND (q_i < θ̄) then
        cnt = cnt + 1
        k = i
    end
end
if cnt == 0                                    /* new stream */
then
    n = n + 1
    T[n] = t
    s = n
end
else if cnt > 1 then  s = −1                   /* collision */

else  T[k] = t;   s = k;                       /* existing stream */
```

Fig. 10. Rise Edge Identification Algorithm

The encoding and decoding stages are also straightforward to implement, and efficient open-source hardware implementations exist. We will tweak these to the desired code rate. The data buffers exist because RS codes operate on blocks of data at a time.

C. The Application Layer

We identified a preexisting Pong implementation [3] that can be adapted to suit our needs. This implementation is made to be executed on a single Altera DE2-115 board with both players controlling their paddles seated next to each other. We will need to port this implementation to be synthesizable on a DE0-CV board and enable a networked multiplayer experience.

Pong Wrapper and Game server will communicate with each other using the following packet type. The data in each packet is decoded based on the packet type.

TABLE I
TYPES OF PACKETS

Packet type	Content
BALL_LOCX	Packet contains ball's X co-ordinate
BALL_LOCY	Packet contains ball's Y co-ordinate
INIT	Packet contains info to setup a new game
PAD_LOC	Packet contains paddle location
ACK	Acknowledgement packet
VALID	Sent when two players have valid setup conditions

For two players to set up a match, they must communicate to the game server that they wish to play with each other. Figure 11 shows the protocol to set up the connection table. The server permits communication between two nodes only after their connection table entries correspond to each other.

After setting up the connections, the two nodes will continuously send their player's paddle position to the game server.

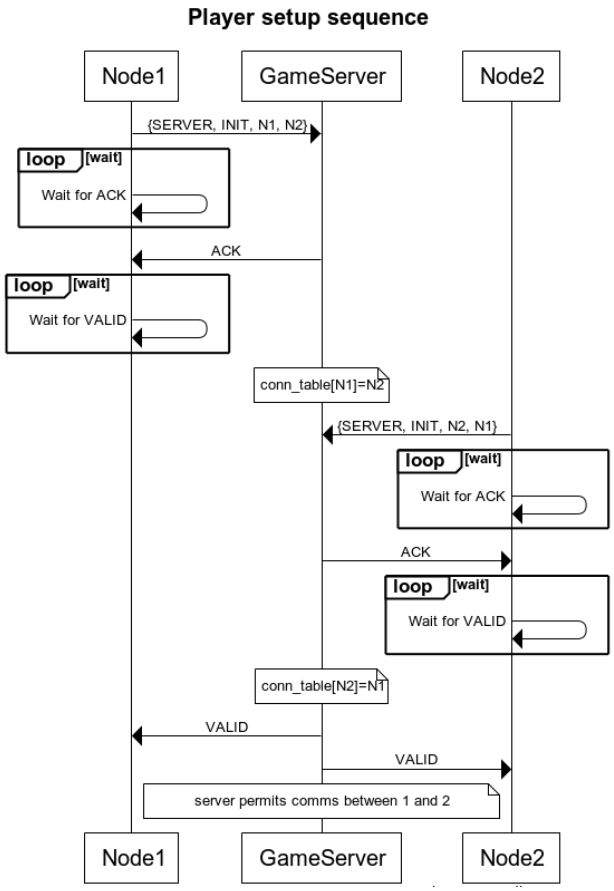

Fig. 11. Protocol to setup a game

The game server is responsible for sending the opponent's paddle location and the X and Y coordinates of the ball to each player. Figure 12 shows the protocol for a node to synchronize game play with the server.

Development of the application layer can be broken down into stages where each stage can be worked on sequentially or in parallel. The first stage of development will require implementing and testing the Pong wrapper module. The test bench for this module will involve sending ball and opponent paddle position to Pong wrapper and receiving the players paddle position.

The next stage in development will require implementing the packet control module which provides an interface between Pong wrapper and the transmitter and receiver modules. Packet control should receive data from Pong wrapper and generate correctly structured packets to send. It should also provide capabilities to receive packets and decode the data contained and provide inputs to Pong wrapper.

The Game server will need to have logic to store the game state and ball positions for up to two simultaneous games between any two player nodes. It will need to interface indirectly with the Packet control using the protocols defined above. The last stage of development will require integrating the different modules to function as expected on different boards. Refer to the validation section for more detailed information on how each of these stages can be tested.

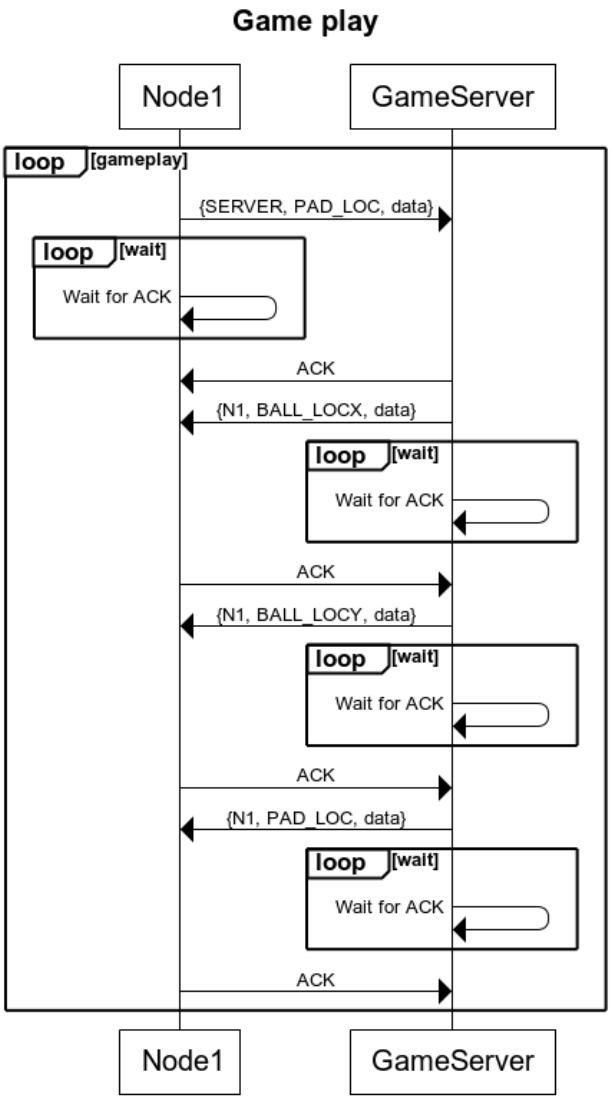

Fig. 12. Protocol for game play

V. Validation Plan

A. The Physical Layer

In the physical transmission of our data, our goal is to accurately transmit a sequence of bits as an analog signal between the two FPGAs. Thus it would be intuitive to perform a quantitative and qualitative comparison of the transmitted wave and the received wave. This includes factors such as the difference in amplitude, phase and waveform of the two signals. In addition, we also need to pay close attention to the rise time as a very high rise time can limit the speed of communication. The transmitter and receiver circuits can be individually tested by sending a simple square wave on one end and validating whether an FPGA can reproduce the square wave on the input GPIO pins using Signal Tap or visually verifying the square wave using an oscilloscope.

Another validation procedure is to implement tests for one-way end-to-end communication between two FPGAs. We plan to code a short pseudo-random number generator on the source and destination FPGAs and initialize it with the same seed. This will result in the sequence of numbers produced on both FPGAs being identical. We will send alternate numbers between the FPGAs and validate whether the number received from the circuit is the number expected from the sequence. The percentage of number of correctly sent bits from a source node to a destination node through the hub will give a quantitative measure of the correctness of our entire system. We built the circuit using a low speed op-amp and other materials available at our disposal to observe the viability of our validation procedures and the results were promising. In the following plots, the yellow graph depicts a 1kHz signal we sent from one FPGAs GPIO pins and the green graph depicts the signal we received at the receiver FPGAs GPIO pins.

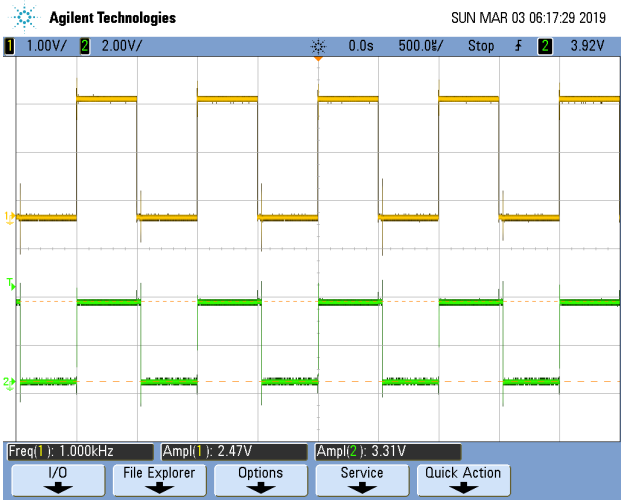

Fig. 13. The Transmitter FPGA's signal (yellow) and the Receiver FPGA's signal (green)

Additionally, we checked the rise time and phase difference in the two graphs at a 1kHz and measured it to be of the order of 3μs. This measurement was taken by recording the difference in time between the 0 voltage and the 90% amplitude voltage. Although we expect it to change as we operate at higher frequencies this proves that our validation procedure are sound.

B. The Data Link Layer

Using Synopsys VCS, we will implement testbenches for our data-link-layer-level modules. These include both manual inspection of waveforms and automated testing via SystemVerilog testbench constructs like assertions and properties. All modules will be unit-tested before being composed together into larger integration tests. And finally, SkyFi will be tested as a whole, incorporating all layers.

The OPPM and FEC modules can be tested in simulation without any dependence on the physical or application layers. These modules can then be put together for integration testing. Absent any errors, we expect an encoded bitstream to be perfectly decoded, and we expect a modulated signal to be perfectly demodulated. In other words, in an environment of

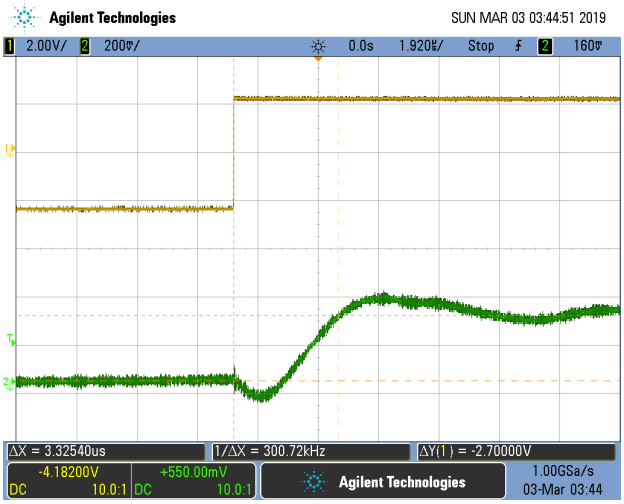

Fig. 14. Rise Time Measurement with a square wave input

simulated perfect transmissions, the data link layer should never signal errant packets or misinterpret pulses. Once correctness has been established, resilience can be tested.

For OPPM, we know that two rise edges with an offset greater than θ must be classified as coming from two different sources. Tests can be written to check for this. We expect a perfect classification of rise edges separated by sufficiently-supported offsets. Then, on the FEC side, there are a subset of errors that can be detected and/or corrected. Naturally, we want all such errors to be handled appropriately and for the appropriate flags to be raised to the Application Layer.

Later in the project, the Data Link and Physical Layers can be tested together, for both single-link (one node to hub) and multi-link network configurations.

C. The Application Layer

We will need to perform unit testing on each of the modules listed in the application layer implementation plan. The test bench for Pong wrapper will require sending ball and opponent paddle position to Pong wrapper and receiving the players paddle position. Tests will need to make sure that the screen correctly updates with the new location of the ball and paddle. The test bench for this packet control will need to test both functions of the module. This includes sending packets to Packet control and checking the produced outputs which feed into Pong wrapper and checking the generated packets when providing test values for ball paddle location. To validate the game server module, test vectors will need to be generated to simulate the protocol to setup a game. The developer will need to write tests to ensure that the connection table is correctly updated and generates the correct packets to send to the payer nodes. The game server will need to be tested to ensure that it sends accurate and synchronized ball and paddle locations to both player nodes. The application layer will also require integration tests to ensure that all of the individual blocks work together. Integration tests will mostly involve setting up arbitrary games and performing visual testing on game play. If errors are present, the interfacing between different nodes will need to be examined for inconsistencies in data or control signals.

VI. Project Management

A. Schedule

A detailed version of the schedule is attached as Figure 16.

B. Work Distribution

In general, we distributed the work by assigning each of us to a layer. Dhruva is primarily responsible for the Physical Layer, Raziq is responsible for the Data Link Layer, and Bhuvan is responsible for the Application Layer. As needed, we may help each other with tasks. For example, once the Physical Layer is implemented and tested, Dhruva plans to code modules in the hub for CRC & error-correction, write testbenches for these modules and run speed & correctness validation procedures on the entire system.

VII. Bill Of Materials

Table II lists all the components the project is expected to require for the final demo:

TABLE II
DETAILS OF ALL COMPONENTS REQUIRED

Component	Price	Quantity	Source
Terasic DE0-CV FPGAs	0	5	CMU Labs
IR LED: TSHF5410	0	10	CMU Labs
Photodiode: QSD2030F	0	10	CMU Labs
Op-Amp: LM7171	$28.87	10	Digi-Key
Resistors	0	-	CMU Labs
Jumper Wires	0	-	CMU Labs
Protoboards	0	10	CMU Labs
VGA cables	0	4	CMU Libraries
Monitors	0	4	CMU Libraries

References

[1] Z. Tian, K. Wright, and X. Zhou, "The DarkLight Rises: Visible Light Communication in the Dark," in MobiCom '16 Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking, pp. 2–15, October 2016.

[2] S. Rajagopal, R. D. Roberts and S. Lim, "IEEE 802.15.7 visible light communication: modulation schemes and dimming support," IEEE Communications Magazine, vol. 50, no. 3, pp. 72–82, March 2012.

[3] D. Chhun, [Online]. Available: https://github.com/Dennis-Chhun/Pong-Game. Base implementation of Pong to be used

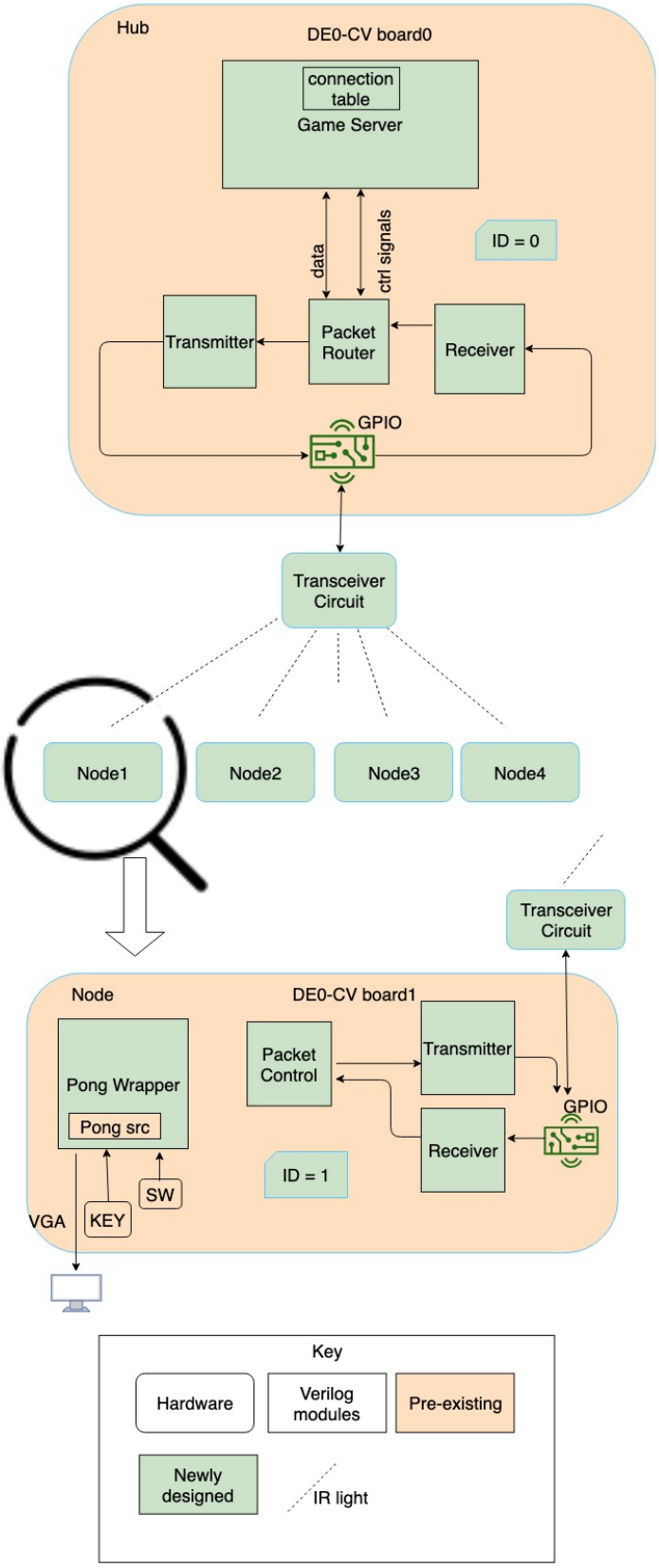

Fig. 15. Complete System Level Block Diagram

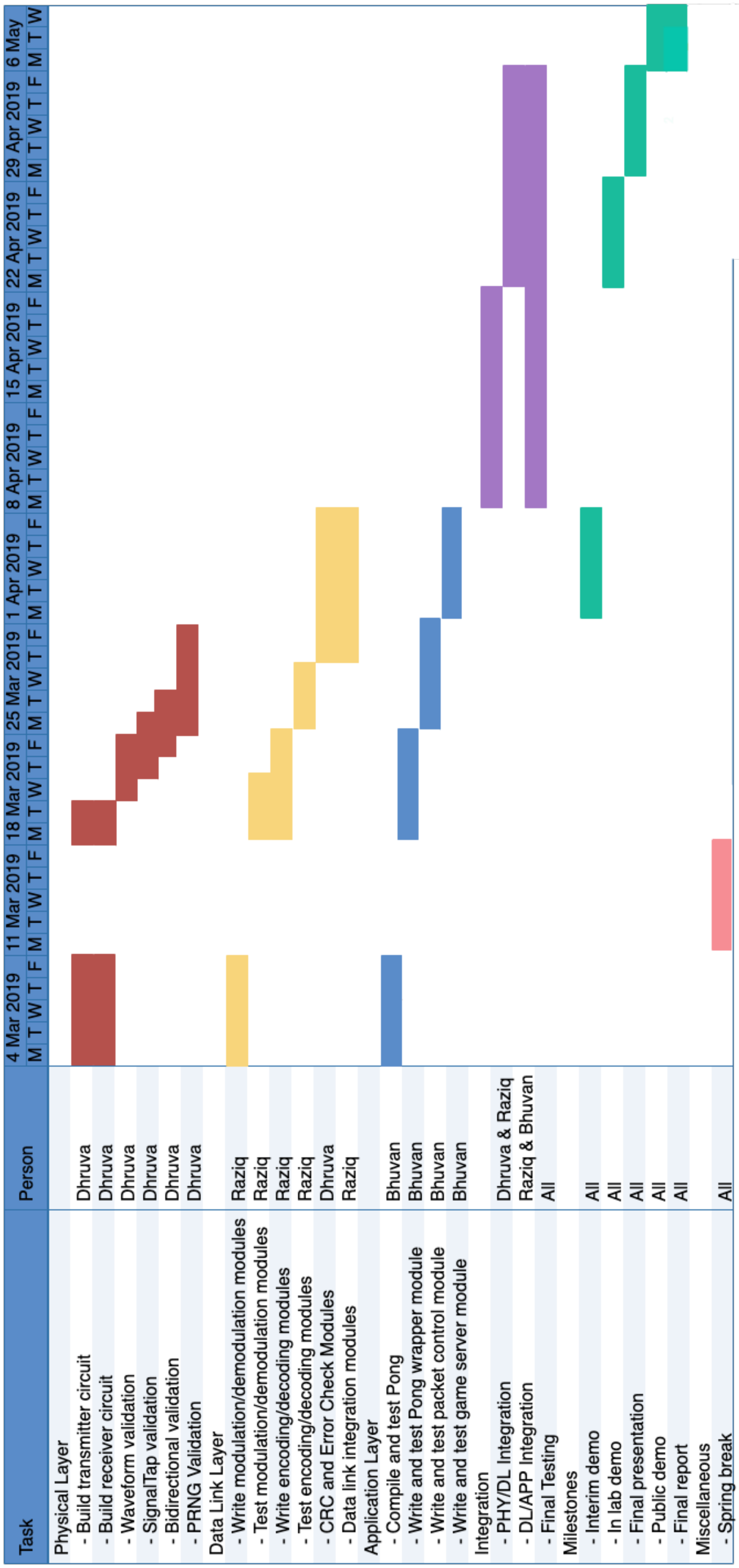

Fig. 16. Projected Execution Plan