# FPGA Virtual Lightboard

1st Ivy Liu
*Department of Electrical Engineering and Computer Science*
*Massachusetts Institute of Technology*
Cambridge, MA
iliu@mit.edu

2nd Melissa Stok
*Department of Electrical Engineering and Computer Science,*
*Department of Materials Science and Engineering*
*Massachusetts Institute of Technology*
Cambridge, MA
mstok@mit.edu

***Abstract*—We present a design for a virtual lightboard implemented entirely in hardware on an FPGA, which utilizes a wired ethernet connection between a "Transmitter" and "Receiver" FGPA to to stream annotated video footage. We implemented this design using a XC7A100T, evaluated its performance and quality through examining the output video stream and any error in inscription, and discussed possible expansions upon the project.**

***Index Terms*—Field programmable gate arrays, Image processing, Ethernet transmission and receiving**

## I. Introduction

Lightboards allow for the overlaying of text onto images, serving as a valuable education and communication resource. Commercial lightboards cost thousands of dollars, can be damaged by improper use, and are not easily transportable. Virtual lightboards, on the other hand, require no physical board, and simply involve a user "writing" into the air with a camera filming them. Through image processing techniques, the writing is overlaid onto the image and stored until erased, allowing for lightboard functionality.

## II. Physical Construction


Fig. 1. Virtual Lightboard system components

The virtual lightboard consists of the following components:

- Two XC7A100T's from Digilent.
- An OV7670 camera module and a 6.2050 VGA camera board.
- A MAX4466 microphone module.
- A speaker.
- Two monitors.
- A drawing glove with a pink finger and palm for image detection.

Fig. 2. Drawing glove with colored fingertip for writing and colored palm for erasing.

## III. Video Processing

Video capture is performed using an OV7670 camera module and a 6.2050 VGA camera board on the "Transmitter" FPGA. Video processing occurs on a 16.5 MHz clock through a sequence of modules to convert pixel values and perform calculations to determine where the writing is occurring. Pixels are then evaluated and valid greyscale pixels or writing is stored in two pixel block RAMs (BRAMs) which hold identical contents.

### A. Pixel Conversion and Mask Creation

Pixels are transmitted from the camera module and sent through the recover module to pair each pixel value with its respective horizontal and vertical indices. Pixels are then sent through a RGB to YCrCb converter. The Y value, a 10-bit greyscale pixel, contains the pixels that will be stored in the pixel BRAMs. Since we are using a bright pink colored finger and palm as the drawing/erasing device, the upper four bits of the red chrominance (Cr) value are used to create a mask tag if the pixel falls within a certain threshold range. The pixels that are tagged with mask for falling within the threshold are used for the center of mass (COM) calculation.

### B. Center of Mass Calculation

One essential feature of the video processing is to determine where the center of the user's finger is to allow them to write or erase on the screen. As previously done in Lab 4, we use the x and y coordinates of select pixels on the screen to preform a center of mass calculation. Using this count of selected pixels and the sum of coordinates, we can calculate our center of mass.

$$m_{total} = \sum_n 1$$

$$x_{CoM} = \frac{\sum_n x_n}{m_{total}}$$

$$y_{CoM} = \frac{\sum_n y_n}{m_{total}}$$

This calculation is performed for each frame in the COM module, and used to select which pixels in the BRAM have writing stored on them and which just have regular grey-scale values.

### C. Pixel Manager

Since certain pixels are written on but others are just the grey-scale pixel, it is important to be able to distinguish between the two and to store the correct pixels into the BRAM. In addition, pixels that have been written on must not be overwritten by grey-scale pixels, because otherwise the writing on the screen would not be maintained between frames. The compare module is responsible for executing this logic. A diagram of the compare module is shown in Fig. 3. The inputs and outputs of the compare module are as follows:

- **valid, x_com, y_com**: Outputs produced by the COM module. Every time a new COM is generated, the valid signal goes high for one clock cycle. Once received by the compare module, these are stored and used for pixel comparisons until a new value is received.
- **sw[2:1], sw[0]**: Switches used for color select and write/erase mode select, respectively.
- **y**: Pipelined upper 6 bits from YCrCb output.
- **hcount_rec, vcount_rec**: Pipelined horizontal and vertical count values from the recover module.
- **current_pixel**: Current pixel contained in requested address in BRAM.
- **pixel_valid, pixel_addr, pixel**: Valid signal that goes high only when a pixel is being written on, not when being read. Pixel address of pixel that is being requested or pixel that is being written (depending on the valid signal). Pixel value that is being written into the BRAM.

The compare module uses a rotating cycle of 8 states since a new pixel from the camera is only received every 8 clock cycles. When a new pixel is received, the address is first calculated and sent to the BRAM to request the pixel that is currently there. Two clock cycles later, this pixel is received and it is evaluated. If the Transmitter is in write mode, then pixels that have 11 in their MSBs will not be written on, because these are pixels that already have color. Otherwise, a new pixel will be written into the BRAM - either a colored pixel if the pixel falls in the COM or in the 8 pixels around it, or a regular gray-scale pixel otherwise. When in erase mode, any location where there is a pixel that is over the threshold is written to as a gray-scale pixel to get rid of the stored color.

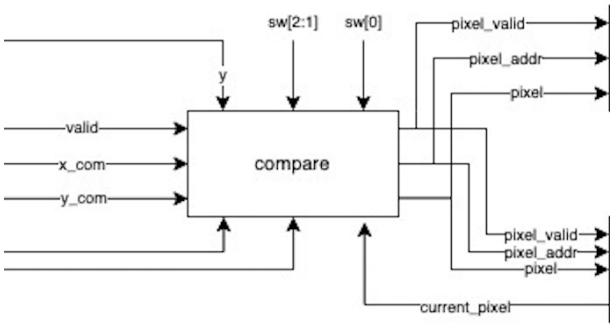


Fig. 3. Diagram of FPGA1 Compare Module

### D. Video Frame Storing

On the Transmitter FPGA (FPGA1), we instantiated two dual-port BRAMs to hold 320x240 8-bit grey-scale pixels. One of the BRAMs is a dual-clock BRAM, with port A being used to store pixel values and port B being used to read pixels to the Ethernet modules. The other BRAM contains identical pixels but is a single-clock BRAM for which port B is used by the VGA to display pixels on the transmitter side. In order to reduce resource utilization, a future improvement could be to only have one BRAM that contains the pixel values. The VGA module could also read from port A in alternation with the pixels that are being written in by the compare module so that it can also be on a 65MHz clock. Port B would be used by the Ethernet on a 50MHz clock.

On the Receiver FPGA (FPGA2), we used one dual-port, dual-clock BRAM to hold the same 320x240 8-bit grey-scale pixels. Port A is written to by the Ethernet module with the pixels that are received from FPGA1, and port B is read from by the modules responsible for displaying to the monitors through VGA.

The pixels that are written on and the greyscale pixels must be distinguishable once read from the BRAMs so that they can be transmitted and displayed properly. The 8-bit pixels which are stored use the following encoding scheme:

| 8-Bit Pixel | Definition |
|---|---|
| 11000000 | Yellow |
| 11010000 | Pink |
| 11100000 | Green |
| 11110000 | Red |
| 00, 6-bit Y | Greyscale |
| 10, 6-bit Y | Threshold |
| 01, 6-bit Y | Crosshair |

## IV. Video and Audio Transmission

The development board comes equipped with an Ethernet chipset capable of implementing full duplex 10/100 Mbps Ethernet according to Diligent's website. For our project, we used 100 Mbps transmission by sending a dibit on every rising edge of our 50MHz clock

We designated one FPGA as the Transmitter, and the other as the Receiver. The Ethernet communication between the two FPGAs is able to run on its own 50MHz clock, separate from the VGA clock, and asynchronously access the data in the pixel BRAM thorough its second port. To avoid the complications of giving unique MAC addresses to the Receiver FPGA boards, packets are sent with the broadcast address FF:FF:FF:FF:FF:FF. A future implementation could verify if we are capable of transmitting to multiple FPGA boards at a time through a router. But, for the moment, we will discuss one-on-one transmission and reception.

### A. Transmission

A challenge of transmission we focused on was how to handle dropped packets. To combat corruption in the received video frames, we decided to also transmit the address of the first pixel of every packet. To accomplish this, we implemented a module that kept track of the pixel address as pixels were sent, allowing us to preserve and sync the address on either side of the transmission. We also used this packaging module to access the pixel BRAM and transmit pixel data as dibits in MSB, LSb in accordance with IEEE 802.3 standard. The same standard mandates a 96 bit time Inter-Packet Gap (IPG) between packets with a max size of 1518 bytes. By transmitting our frame line-by-line, our Ethernet Packets follow the structure given below in Fig. IV-A to meet these standards with a fixed packet size of 356 bytes (362 when including the IPG).

356 Bytes

| 7 bytes | 1 byte | 6 Bytes | 6 Bytes | 2 Bytes | 3 Address + 320 Pixel + 1 Audio Bytes | 4 Bytes |
|---|---|---|---|---|---|---|
| Preamble | Frame Start Delimiter (FSD) | Destination Address | Source Address | Length | Data | Frame Check Sequence (FSC, CRC checksum) |

For the 362 bytes that will ultimately be transmitted per packet, we calculated a frame refresh rate over Ethernet of 143.9 fps, which is well above the camera's 30 fps. We provided the refresh rate calculations below.

$$50\text{MHz} * 2 \text{ dibits/cycle} = 100\text{Mb/s}$$

$$\frac{100 \text{ Mb/s}}{8 \text{ bits/byte}} = 12500000 \text{ bytes/sec}$$

$$\frac{12500000 \text{ bytes/sec}}{362 \text{ bytes/packet}} = 34530.4 \text{ packets/sec}$$

Since we send a line of pixels per packet, we then simply divide by 240 packets/frame to get our final refresh rate.

$$\frac{34530.4 \text{ packets/sec}}{240 \text{ packets/frame}} = 143.9 \text{ frames/sec}$$

During the transmission of the header, checksum, and IPG, a stall signal is used to ensure we do not deviate from the intended line-by-line transmission of the frame preserved in our pixel BRAM. The transmission duration of this signal can be altered to allow us to pick arbitrary packet sizes as long as they meet the RMII specification of being between 64 and 1518 bytes.

As a reminder, our camera generates 320x240 frames at a rate of 30 fps, and our monitors at 60 fps. Since we currently cannot make use of the extra transmission speed without upgrading our camera or monitors to have a higher fps or resolution, it would make sense to change the duration of the stall signal in order to decrease the packet sizes and adjust the rate frames are sent at. This would have implications on how frequently we receive new audio samples. For example, decreasing the packet size by decreasing the number of video pixels sent would allow us to send more packets per second, resulting in a higher number of audio samples sent per second. This also has implications on how much of the data transmitted is user-viewable. Since every packet comes with 41 bytes of boilerplate to make the packet conform to transmission standards (Ethernet header, IPG, etc) and allow the data to be interpreted (BRAM address), decreasing the number of data bytes sent in each packet would decrease the percentage of user viewable data.

| x pixels + 1 Audio Byte | User-Viewable Data % | FPS | Audio Kilosamples/Sec |
|---|---|---|---|
| 81 | 66.39 | 106.73 | 102.46 |
| 161 | 79.70 | 128.92 | 61.88 |
| 321 | 88.67 | 143.92 | 34.53 |
| 641 | 93.99 | 152.74 | 18.33 |
| 1281 | 96.90 | 157.59 | 9.46 |

Alternatively, we have the extra transmission speed necessary to instead increase the size of pixels sent. So we could display full color 12-bit pixels and audio samples and still retain over 50% of our transmission rate. We also have the Block RAM necessary to store these frames, as we are currently using less than 50% available Block RAM sites on the board (see Section VI). If we followed the current pixel encoding scheme that reserves bits on each pixel received to select color, a 16-bit transmitted pixel would allow us to display 12-bit color pixels with an Ethernet transmission rate of 76.26 fps.

If we instead wanted to upgrade our camera and monitor to allow for greater resolution and choose maintain our current 8-bit pixels, we could achieve the following rates below.

| Resolution | FPS | Audio Kilosamples/sec |
|---|---|---|
| 320 x 240 | 143.9 | 34.53 |
| 720 x 480 | 34.10 | 16.40 |
| 1024 x 768 | 15.05 | 11.73 |

### B. Reception

After passing through the Media-Access Control layer that was previously implemented in class, the packet is divided into a write address, pixels, and audio data. Using the write address that was most recently received, the modules will increment and store the pixel data in the pixel BRAM on the Receiver FPGA as it is received.

With high transmissions, pixel corruption during transmission is a negligible concern compared pixel address corruption. A helpful refinement upon the work done thus far would be completing a our checksum only over the pixel address of the packet. Or perhaps, sending packets in an alternating pattern of address-data instead, where a previously received address packet is a requirement before accepting the following data packet.

## V. Video Displaying and Audio Syncing

### A. VGA Displaying

VGA displaying is done through a series of four modules. The VGA module is used to generate VGA display signals for a 1024x768 display at 60Hz. The horizontal and vertical counts (hcount and vcount) that are generated by this module are then used by the mirror module, which scales down the counts to 5/16 of its value and mirrors the hcount so as the grab the pixel from the opposite side of the display. It generates a 16-bit pixel address which is used to index into port B of the BRAM to obtain the desired output pixel. The output pixel is sent to the scale module, which directly transmits the pixel if it falls within the desired displaying region on the monitor or sends a black pixel if this is not the case. The VGA, mirror, and scale modules are identical on both FPGAs.

The one difference between the displaying on the two FPGAs is in the VGA mux module. This module is responsible for taking in the 8-bit values from the BRAM and converting it into a 12-bit {R,G,B} value to be displayed by the VGA. This module sorts out the pixels that have been colored on from those that are greyscale. On FPGA1, the threshold and crosshair pixels are also filtered out and displayed as pink and green, respectively. This helps the user know where on the screen they are writing or erasing so that they can properly orient their hand. On FPGA2, the threshold and crosshair pixels are just read out as regular grey-scale pixels.

### B. Audio Recording and Syncing

We found that due to the high transmission speed of our Ethernet connection, we were able possible to transmit our 8-bit audio with a sample rate of 34.53kHz simply by sending a single audio byte in each packet. This result enabled us sample the current audio sample on the microphone on Transmitter board, send it directly to the Receiver board, and output it through the board's 3.5mm aux port, thus eliminating the need to deal with the syncing issues that come from having a buffer.

Although it was possible to distinguish words from the audio received, we found that the system was somewhat susceptible to noise. A possible future add-on of this system could be a low-pass finite impulse response (FIR) filter to attenuate high frequency noise and to handle anti-aliasing during the down-sampling process. Alternatively, upgrading the speaker or microphone module we used may also help decrease the static heard.

| | Site Type | Available | Used | Utilization % |
|---|---|---|---|---|
| FPGA1 | Memory LUT | 19000 | 5 | 0.03 |
| | Logic LUT | 63400 | 900 | 1.42 |
| | DSP | 240 | 10 | 4.17 |
| | BRAM | 135 | 48 | 35.56 |
| FPGA2 | Memory LUT | 19000 | 2 | 0.01 |
| | Logic LUT | 63400 | 400 | 0.63 |
| | DSP | 240 | 1 | 0.42 |
| | BRAM | 135 | 18 | 13.33 |

## VI. Resource Utilization and Timing

### A. Resource Utilization

### B. Timing

FPGA1 and FPGA2 use 65MHz and 50MHz clocks. The 65MHz clock is used for video capture and displaying, and the 50MHz clock is used for Ethernet and audio capture and playing.

FPGA1 has a WNS of 8.828 ns and a TNS of 0 ns. FPGA2 has a WNS of 9.536

## VII. Retrospective

Throughout our work on this project, we learned several valuable things.

- When creating a complex system, it is easier to start with something small that is functional and incrementally add on components. This allows for an easier debugging process where it is possible to identify issues and where they stem from instead of having many possible connections from which the issues may have arisen.
- In the same spirit, thinking about dependencies between systems when deciding what to develop first leads to less future revision and rework.
- Although test-benching can help in the development stages, it does not catch all possible issues. Hardware debugging takes a lot of time. Having test-benches that cover end-to-end processes developed can be useful in the hardware debugging stages to recreate the errors that are seen and try to fix them quickly.
- Small counting errors can propagate and cause entire systems not to work.
- Establishing a well-defined interface between systems, especially those created by different team members, facilitates integration of all components at the end of the project.

## VIII. Contributions

Melissa worked primarily on the image processing on FPGA1 and the VGA displaying for both FPGAs. Ivy worked on Ethernet packaging on FPGA1 and reception on FPGA2. Audio transmission was a joint effort.


## IX. Acknowledgment

We would like to thank Joe Steinmeyer, Fischer Moseley, Jay Lang, as well as the other 6.2050 Fall 2022 LAs for a wonderful semester and their support in producing this virtual lightboard.

## X. Appendix A: GitHub Repository Link

https://github.com/waterRK9/62050_final_project

## XI. Appendix B: FPGA Block Diagrams

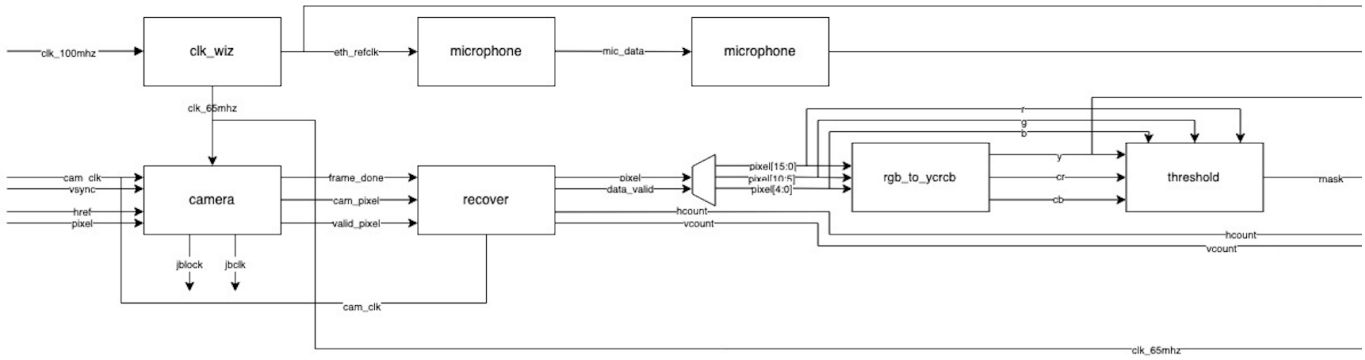


Fig. 4. Diagram of FPGA1/Transmitter

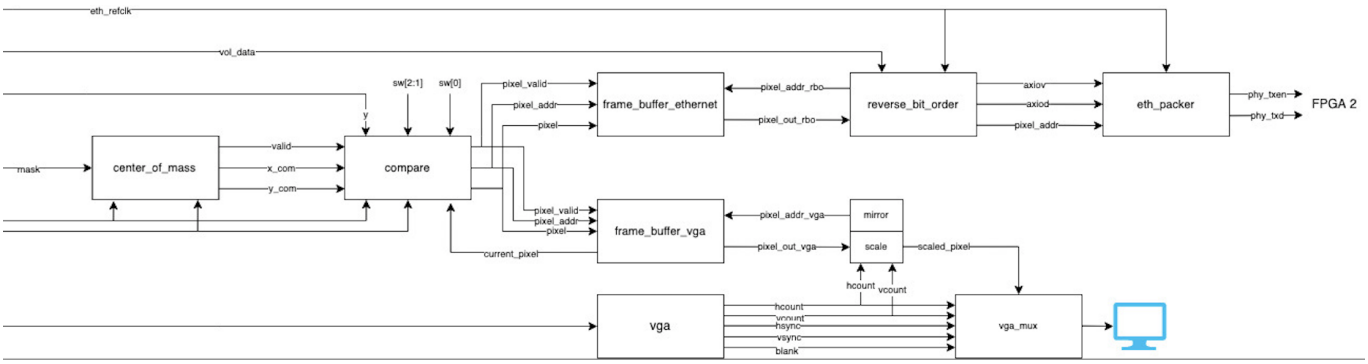


Fig. 5. Diagram of FPGA1/Transmitter

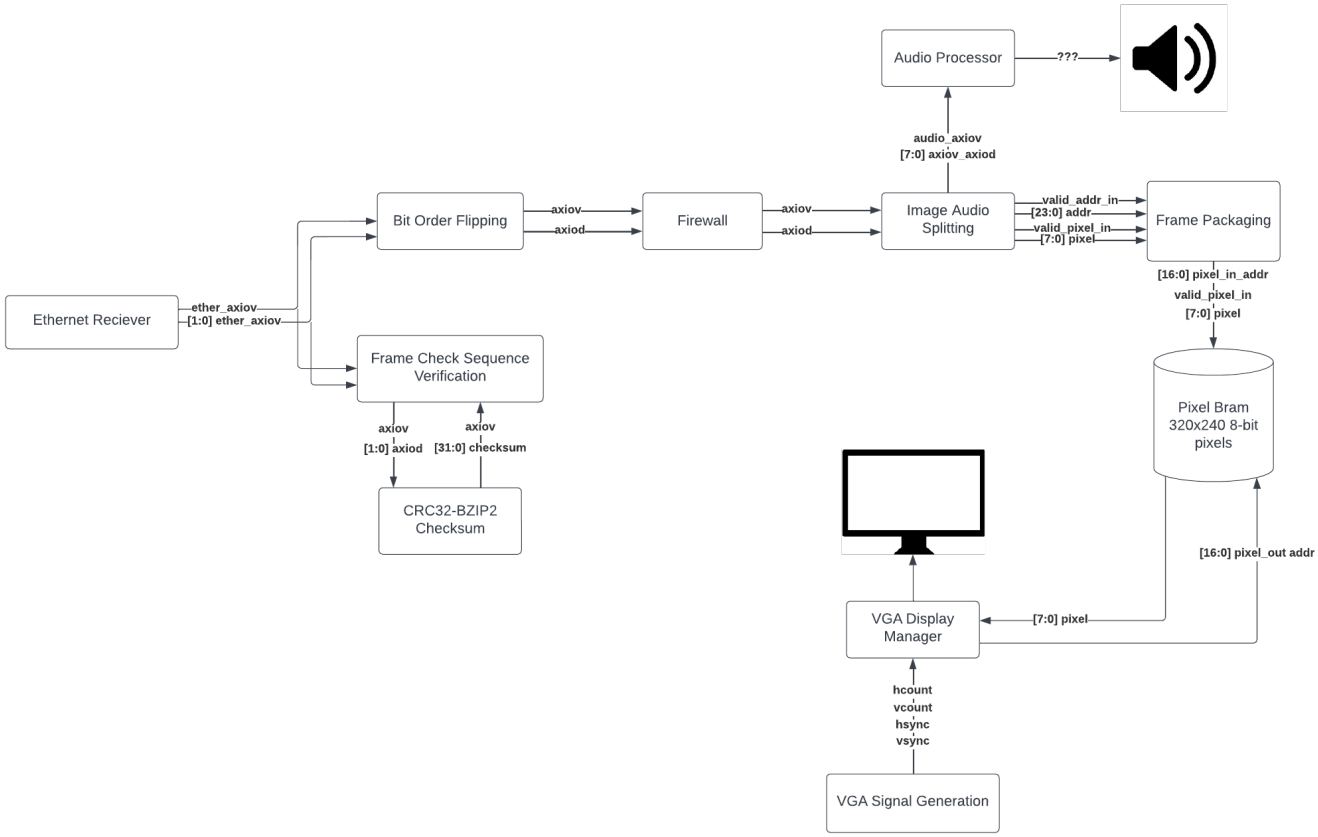


Fig. 6. Diagram of FPGA2/Receiver