Network-Attached Laser Projector

Jay Lang
jaytlang@mit.edu

Fischer Moseley
fischerm@mit.edu

6.111 Final Report
9 December 2020

Contents

1 Abstract

We present a design for a Network-Attached Laser Projector implemented entirely in hardware on an FPGA fabric, which utilizes a novel parallel-stack UDP offload engine to connect to a local area network (LAN), and stream full-color vectorized images to an RGB laser projector. This hardware networking stack interfaces with the laser control module directly, and user packets are processed in real time, allowing for total system throughput exceeding 100 megabits per second. We implement this design using a custom laser module and the Nexys 4 DDR FPGA, evaluate its performance and quality using custom trajectory generation software to stream images over a custom application layer network protocol, and discuss potential areas for future expansion and improvement.

2 Block Diagram

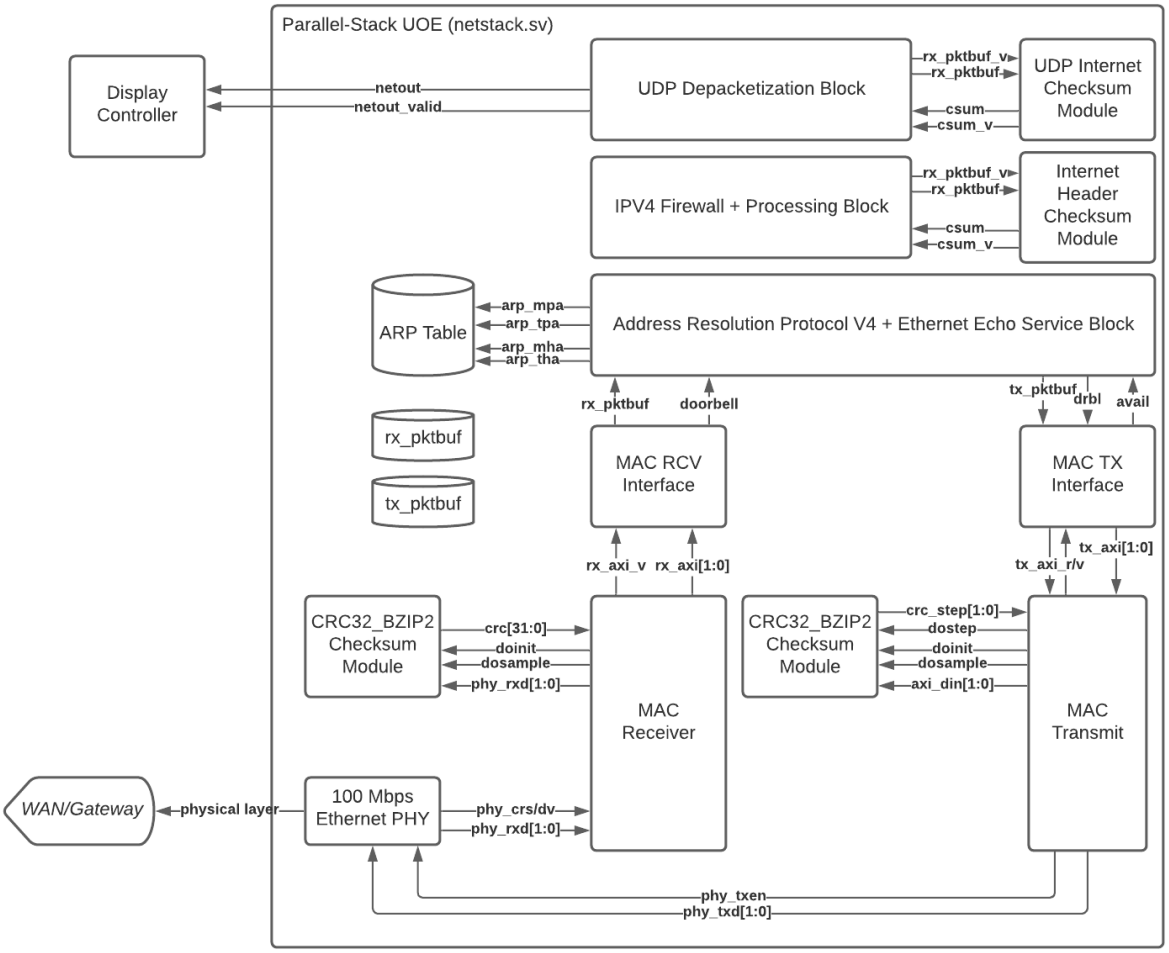


Figure 1: The networking stack, utilizing input from an internet gateway via the Ethernet interface to output depacketized data to the display controller. Note that the two packet buffers are globally shared throughout the top-level network stack module.

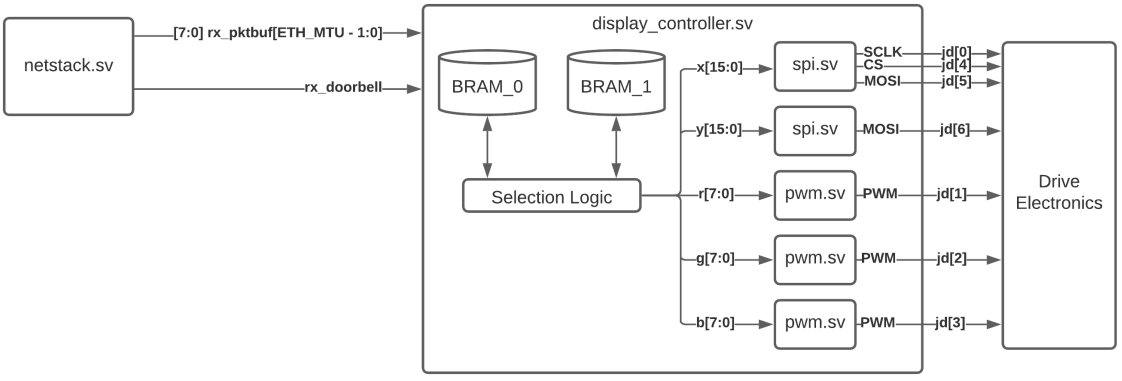


Figure 2: The display logic and external components required, with input from the BRAM controller, and output to a set of SPI DACs.

3 Physical Construction (Fischer)

Figure 3: The top and rear panels of the device.

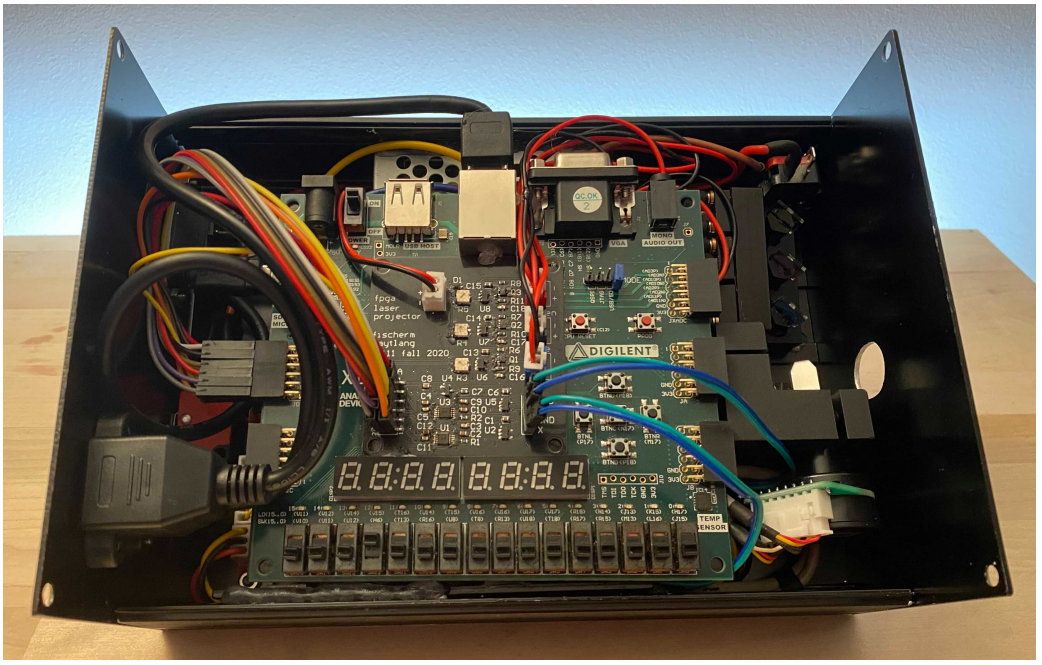


Figure 4: The top of the device, with the top cover removed. The Ethernet cable is visible on the left, the laser optics are on the right, and the custom laser driver board is in the center.

The projector is enclosed in an 82 x 145 x 200mm anodized extruded aluminum enclosure, which is interlocked with a pair of microswitches as per EHS regulations (Appendix A). These interlocks interrupt the power to the mirror glavonometers and laser diode drivers if the front or rear panel is removed.

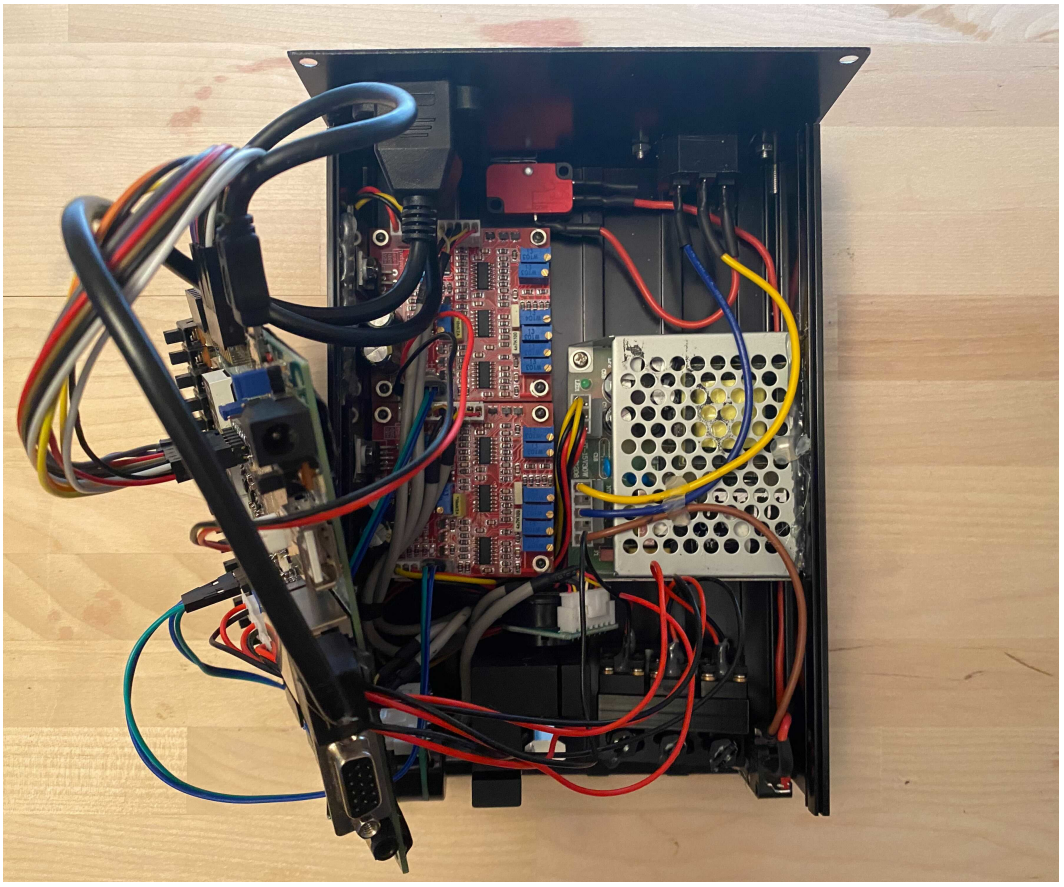


Figure 5: The top of the device, with the Nexys board moved to the side. The $\pm 15V$ power supply is visible on the right, and the pair of glavonometer drivers are on the left, with the laser module and galvonometers at the bottom.

Apart from the interlocks, the enclosure also contains the following:

- An IEC203 connector for 120V power, mounted on the rear panel.
- An Ethernet connector, mounted on the rear panel.
- A micro-USB connector, mounted on the rear panel.
- The Nexys 4 DDR board, with the custom laser/galvo board connected to a PMOD connector.
- A bipolar, ±15V power supply for driving the laser galvos and laser diodes.
- A pair of galvonometer drivers. The galvonomoters have closed-loop control, so these boards implement some kind of control loop in analog electronics.
- A galvonometer module, which includes a set of mirrors mounted orthogonally such that the laser position can be varied in the x and y directions.
- A RGB laser module. The multicolored output is formed by combining 660nm, 520nm, and 450nm lasers, which does not create a true full-color output, but it is sufficient for our purposes. The beams are generated by individual lasers and combined with dichromatic mirrors into a single colinear beam. This is driven by

the custom board connected to the Nexys board's PMOD port. The output of the laser module is directed into the galvonometer's input before being reflected out of an aperture on the front of the module.

4 Image Processing (Fischer)

To pre-prepare images and videos for display via the laser projector, images need to be vectorized and converted into a path of points the projector can understand. This processing currently takes place in Python using the OpenCV toolkit, and consists of a few steps:

- Rescaling. The image processing script is meant to handle any arbitrarily sized input, so it rescales everything to a uniform 512x512 square before any processing begins. This is done to keep the generated trajectory small and quickens the actual processing.
- Canny Filtering. A greyscale map of the image is generated, and then processed by a Canny Filter to detect edges.
- Trajectory Planning. The canny filter only outputs a rasterized image, not lists of connected points. OpenCV's `connectedComponents` function is used here, which returns lists of pixels on the same contour. These lists don't have any order to them, so they must be resorted such that adjacent pixels come after each other in the list. This process is incredibly slow as it occurs in Python instead of C++, like the rest of the OpenCV backend. Once the contours have their pixels properly reordered, a nearest-neighbors algorithm is run to find the next contour to add to the trajectory. The final trajectory is passed on to the next step.
- Recolorization. Each point in the output trajectory is colorized by sampling the color of the same (x, y) point in a blurred version of the original image. It was found that blurring the image before sampling helped to produce a more predictable color of the contour, as blurring effectively averages neighboring pixel colors.
- Network streaming. Each point is encapsulated in our custom application layer network protocol, and then sent out via the conventional OS socket API at the FPGA. Earlier versions of the system also supported receiving information over a direct Ethernet link; The Scapy Python library was used for crafting and sending such packets.

This process can be be seen in the following images:

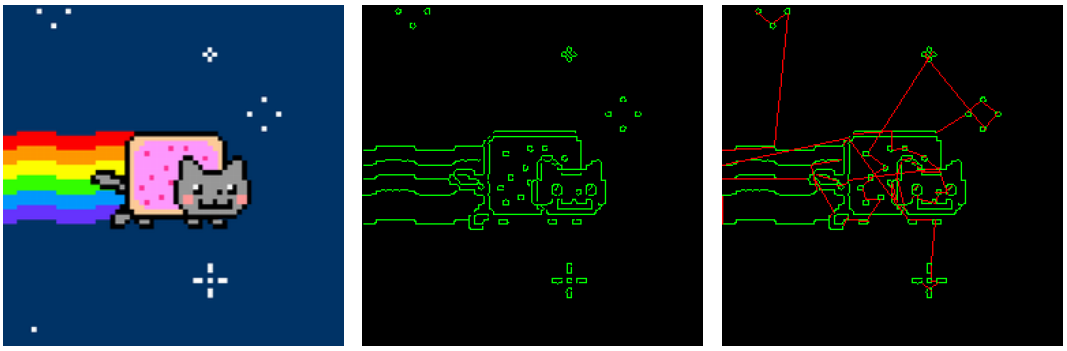


Figure 6: Output of trajectory planning. On the left, the source image. In the center, the image to be rendered by the projector. On the right, the image to be rendered, but including the jumps between contours in red. Although the actual output is colorized, the trajectory here is shown as monochrome for clarity.

The image processing script can process either a video, a static image, or a webcam input by specifying a number of command line options. It was also possible to export the resulting output into either a `.coe`, `.csv`, `.png`, or `.traj` file, which was incredibly useful for debugging.

5 Networking Offload Engine (Jay)

The networking offload engine is a sophisticated, parallel-stack, pure-hardware device which maintains a connection to an Internet Protocol Version 4 (IPv4) gateway, and allows application layer data to be channeled to other hardware modules on the FPGA fabric over UDP.

The design of this module is broken into several parts, corresponding with the appropriate layer of the canonical OSDI model. We introduce these components from the bottom up.

5.1 The Physical Layer

The FPGA comes with an Ethernet chipset implementing the IEEE802.3 fast Ethernet standard, thus rated for 100 Mbps full-duplex operation. The chipset exports a number of configuration registers to the FPGA, in addition to implementing the RMII specification.

5.2 Media Access Controllers

We implement a Media Access Controller (MAC) layer to complement the Ethernet physical (PHY) chipset on the Nexys board, per the IEEE802.3 standard. Separate modules are devised for reception and transmission of packets to support full duplex operation, each translating raw Ethernet II packets back and forth from Ethernet frames. In order to populate the Frame Check Sequence (FCS) and verify it against received packets, an Ethernet checksum (CRC32-BZIP2) module is implemented. As is the case in modern NICs, the FCS is shedded after it is verified, in addition to other Layer-1 specific structures such as the Ethernet preamble.

This module is small, but difficult to test *in vivo* due to the complexity of the RMII interface and the resulting number of partitions on input signals. To test this module, we utilize the popular sniffer Wireshark to view raw packets which contain a passing FCS, and additionally configure a network card on the controlling machine to discard the FCS regardless of its correctness. A custom Ethernet protocol was implemented to accelerate this process; when a packet of this type is received, it along with its data is echoed back to the sender and never processed by the rest of the networking subsystem.

5.3 Address Resolution

The Address Resolution Protocol (ARP) is implemented above the MAC layer according to RFC 826, to enable discovery and static protocol addressing for the LAN-connected FPGA. At compile time, a MAC address and desired IP address are specified within the system logic, and the host LAN is configured to allow static IP addressing outside of its DHCP range (if necessary).

The ARP implementation utilizes a single element table to store protocol and hardware addresses of the gateway. As such, the system is capable of not only responding to ARP requests for its own address, but updates its own table mapping dynamically in accordance to the algorithm specified within the RFC.

To simplify implementation, the ARP module examines a single large buffer (implemented as an array of byte registers and likely synthesized into block RAM), looking for relevant identifiers at appropriate offsets. As this is extremely difficult to replicate in simulation, the full dynamic system is tested via external sniffing (thorugh Wireshark etc.) and fuzzing, in conjunction with several different router configurations.

5.4 The Internet Protocol

A subset of IPv4 is implemented according to (the infamous) RFC 791, and the header checksum logic is implemented in a separate module according to RFC 1071. When an IP packet is passed from the MAC subsystem, the IP block verifies the header checksum using this module and also checks to ensure that the packet doesn't utilize additional options, is not fragmented, and encapsulates a UDP packet. If none of these conditions are true, the packet is dropped.

These steps ensure the integrity of incoming data, and additionally ensure that the packet conforms to only our desired subset of IPv4. While this imposes some restrictions upon incoming data (e.g. width cannot exceed the Maximum Transmission Unit (MTU) of the underlying physical-layer medium), we have found that we implement a sufficient subset of IPv4 to where all major operating systems and all tested router hardware convey packets in a manner our device can understand.

Note that the Internet Control Message Protocol is not implemented, so bad IPv4 packets (e.g. bad checksum, exceeded time to live, etc.) are simply dropped rather than relayed back to the sender.

5.5 User Datagram Protocol

We implement the User Datagram Protocol (UDP) on top of our IPv4 receive layer in accordance with RFC 768. The checksum is currently omitted, in order to provide a constant-time depacketization capability, but the necessary logic to compute this is implemented according to RFC 1071 (using the same checksum logic as the IPv4 layer). Data at this layer is completely depacketized once received, and if the incoming port matches compile-time configuration, data is passed on to the display controller.

5.6 Performance Characteristics

The network stack is novel for a number of reasons: most significantly, since it is implemented in pure hardware, it is capable of depacketizing data in a bounded number of clock cycles. This number varies depending on the presence of certain IPv4 options, but ultimately falls well beneath the interframe gap required by IEEE802.3. This implies packets can be sent along to the laser projector at 100 Mbps reliably - in addition, since the transmission layer is implemented in parallel to the reception layer, full-duplex operation can be properly taken advantage of and impending transmission doesn't impact the integrity of incoming data.

Furthermore, assuming an IPv4 header length of 20 bytes, the system is theoretically capable of computing all checksums and performing all depacketization within 6 clock cycles. This easily puts it under the required time to implement the gigabit Ethernet standard, and potentially allows the implementation of more complex transport-layer protocols (e.g. TCP) using existing lower infrastructure.

6 Laser Display Module (Fischer)

6.1 Framebuffer

As data is streamed off of the network and into the projector, incoming sets of (x, y, r, g, b) points are buffered such that the display controller can later scan through them and write them to the drive electronics. Originally we considered streaming directly into the framebuffer, but the incoming packet stream is of variable speed, meaning that the drive electronics could read from the framebuffer before the network stack is finished writing to it. This would produce a distorted frame.

To mitigate this, we use a pair of BRAM banks inside the display controller. When packets are being received, the network module will write into one BRAM bank, waiting for it to fill. Once the end of the frame is reached, the host computer will signal the FPGA to exchange the banks. The module will then save the current BRAM address as the end of the frame, and then toggle an internal `bram_select` line to indicate that the incoming and outgoing BRAM banks have been swapped. Points are then written out to the drive electronics from the freshly filled BRAM bank, and the newly decomissioned BRAM bank is made available for new points to be recorded into. Each BRAM bank is 20,000 addresses deep and 64-bits wide.

6.2 Packet Structure

The data enclosed in the packets is 64-bits wide, and follows the following structure:

cmd	x	y	r	g	b

- `cmd`: The control signal the FPGA uses to determine when to switch BRAM banks in the framebuffer. This field is set to 0x01 when data is being streamed in, and set to 0x02 when the frame is complete and the BRAM banks should be flipped. This field is 8 bits wide so that the entire packet would be 64 bits wide, which is conveniently the width of our BRAM buffers. This enables us to save the entire packet into BRAM without worrying about rearranging the packet.
- `x`: The position of the laser beam in the x direction. The DAC that feeds the drive electronics is 16-bit, so this field is also 16 bits wide.
- `y`: The position of the laser beam in the y direction. The x and y channels are identical, so this field is also 16 bits wide.
- `r`: The intensity of the red light at the point. Most images use 8-bit color anyway, so using 8 bits here seemed reasonable.
- `g`: The intensity of the green light. 8 bits wide.
- `b`: The intensity of the blue light. 8 bits wide.

These packets are stored in this same format in the 64-bit wide BRAM banks. It is also worth noting that the cmd parameter is only respected on incoming packets, and packets being read out of either BRAM bank ignore this parameter.

6.3 Display Controller

Once a complete framebuffer has been assembled and filled with packets following the above structure, its contents are sent to the drive electronics. For galvonometer control, this takes the form of two SPI-connected DACs, with the output buffered by a pair of unity-gain opamps. SPI was chosen for the interface because initial testing revealed that I^2C was much too slow to update the DACs fast enough to produce a non-flickering image, even when running at 400kHz clock speed. Using SPI also allows us to skip the address byte at the start of every transmission, increasing throughput, and using concurrent SPI buses also allows us to leverage the parallel nature of the FPGA for a tangible speed improvement.

Originally we considered using a non-unity gain amplifier stage on the output of the DACs to use more of the dynamic range of the galvonometers, which supposedly accept a voltage input between $0-15V$. Ultimately, this was decided against as documentation on the laser modules was virtually nonexistent, and it wasn't worth risking damage to the laser driver. This had the unexpected benefit of requiring the mirrors to undergo less displacement to produce the same image, effectively reducing acceleration on the galvonometers and making a less fuzzy image. This does come at the expense of throw distance, and the projector must be nearly 15 feet from the screen in order to produce an image with any discernable detail. We considered this tradeoff reasonable to prevent any accidental damage to the hardware.

On the laser side, a PWM signal was used to control the intensity of each laser. We planned on using DAC-based output stage originally for controlling laser power, but it

was deduced that a PWM signal could be varied faster than the x and y galvonometers could be, and PWM would suffice. This wouldn't be possible on a traditional microcontroller because of the difference in clock speed, and so a high-frequency PWM approach better highlights the unique nature of the FPGA and enables a less elaborate output stage.

Each channel of PWM output from the FPGA is fed into a potentiometer that sets the current on an opamp-based constant current source. This potentiometer can be adjusted to vary the maximum output power for compliance with EHS safety restrictions. The constant current source uses a NPN transistor to regulate the current through the laser diode, and was chosen instead of a MOSFET because of the absence of any gate-source or gate-drain capacitance. Since the BJT is a current-controlled amplifier, parasitic voltages cannot accumulate across the gate capacitance and accidentally turn on the laser diode if the laser driver is tampered with. The lack of gate capacitance also enables a faster switching time, and the lack of any (significant) switching losses.

This design was implemented on a custom PCB, designed in Altium and manufactured by PCBWay.

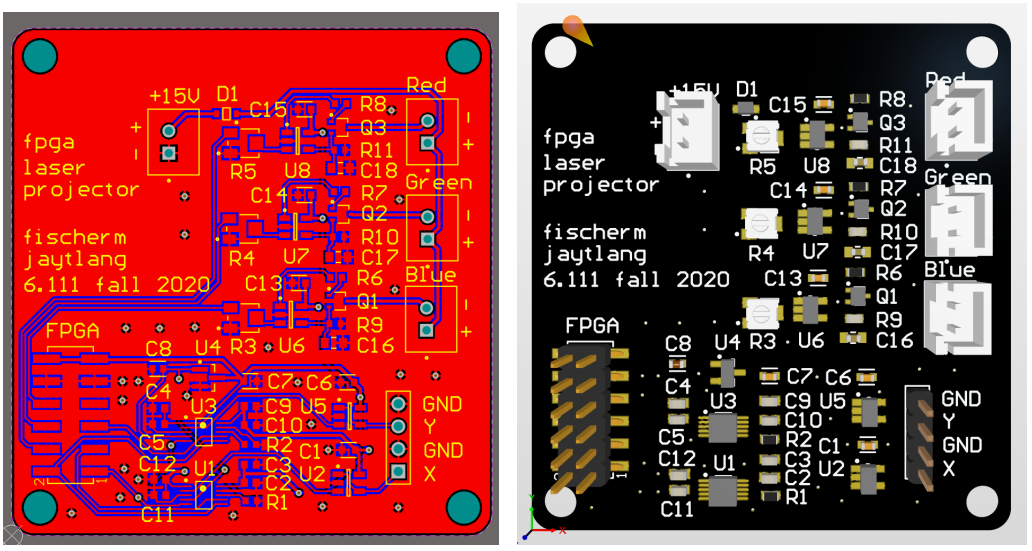


Figure 7: The 2D and 3D renders of the board in Altium.

7 Retrospective

In hindsight, we learned a few rather important lessons during the development process:

- Simulations are incredibly useful when the input space is limited. For instance, most of the display control was verified with simulation before a bitstream was ever generated, and even then it worked the first time on hardware. This saved a significant amount of time.

  However, simulation of modules with more complex input spaces (e.g. the Media Access Controllers) can get unwieldly, especially when the variation of these inputs over time is critical to assessing system correctness. For this reason, it's often useful

to utilize an alternate method of design verification - in our case, the NIC FCS discard combined with the custom Ethernet-layer echo protocol proved invaluable.

- Doing trajectory planning properly is super hard. There's some rather fancy graph-based algorithms for computing a proper path through a set of points, but we didn't feel like spending a bunch of time on that really captured the spirit of 6.111 - and so we consider our nearest-neighbors algorithm sufficient. Those algorithms would take longer to run than our current system (which is slow enough as is) and would probably require porting to a compiled language to make it run in any reasonable amount of time. This would be the most useful improvement to the image processing if more time was had.

- System interfaces are anathema to large systems - whether they lie in software or hardware. Accordingly, if two team members are implementing two different parts of the system, picking a well defined interface, establishing it clearly, and utilizing it throughout design iteration is invaluable. As we developed our system, we determined a suitable interface between the network offload engine and the display controller fairly early on - and obviously the interface between the network offload engine and software image generation is well defined through tens of RFCs. This way we were able to re-integrate the design several times utilizing the same interface, in a bug-free manner and a (surprisingly) rapid pace

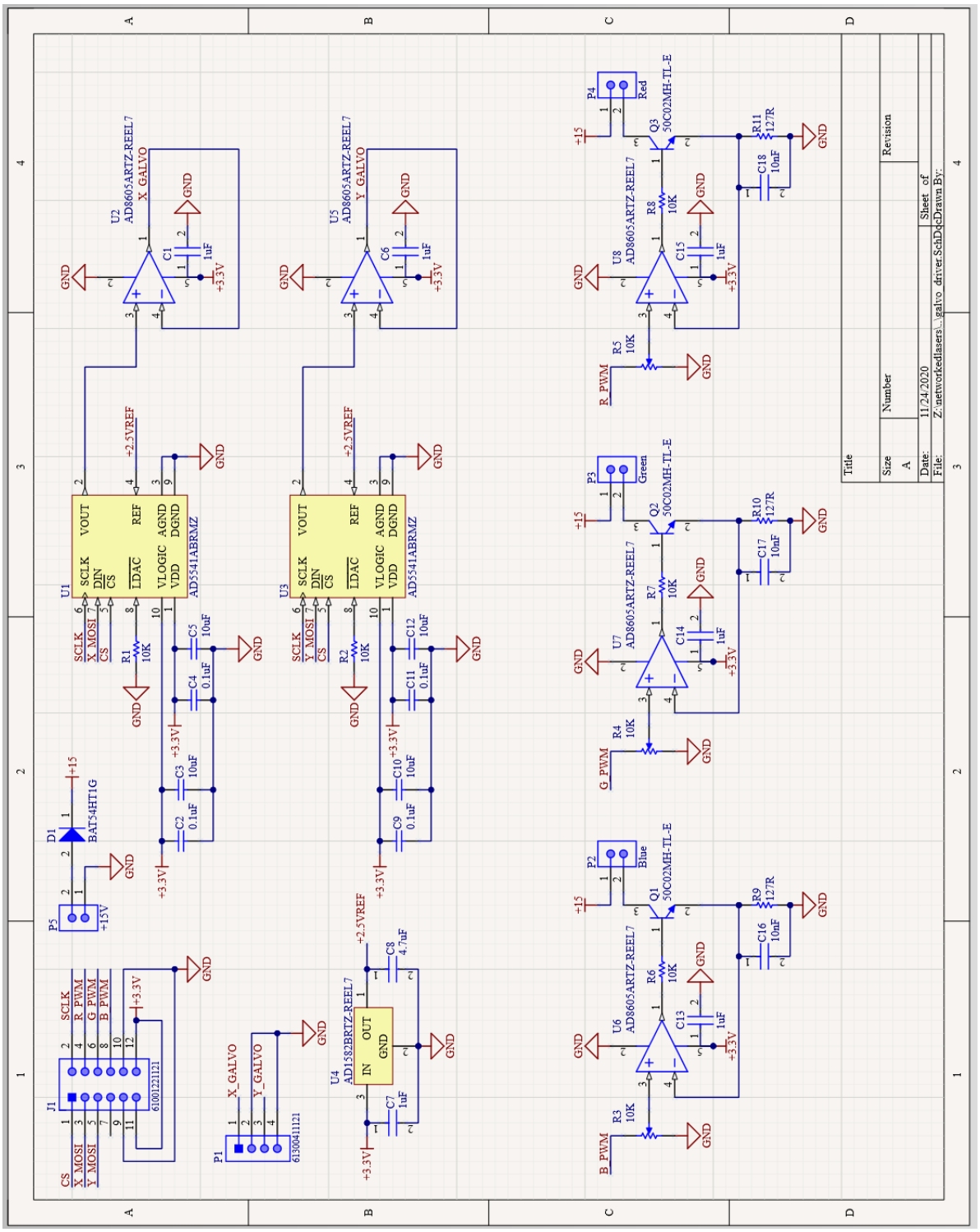
- Don't buy sketchy USB Ethernet adapters off the internet, because sometimes they don't implement the slower 100 Mbps standard correctly. Make sure you've got old and established hardware, or better yet, an actual Ethernet port on your device you can manually modeset with `ethtool`.

8 Appendix A - Laser Safety and Procedures

Lasers are sketchy. Especially cheap, poorly-documented ones procured from the Internet. The RGB laser module that has been procured is a little short on trustworthy specifications, and as a result MIT's Environmental Health and Safety office (EHS) has been consulted to operate the laser safely. To ensure compliance with their requirements, the laser:

- Does not output more than 5mW on any channel. This classifies it as a Class 2M laser.
- Is mounted inside a tamper-proof, interlocked enclosure that cuts power to the laser when opened. This is accomplished with a pair of microswitches in series with the 120V supply inside the enclosure.
- Has been tested by EHS to verify that the laser output spectra is not harmful to humans.

9 Appendix B - Driver Board Schematic



10 Appendix C - Getting Networking to Work

In order to hook the NALP up to your LAN (or my networking stack in general), you'll have to do the following:

- *Optional:* Assign a MAC address to the FPGA. Currently I have my FPGA co-opting a MAC address from a Raspberry Pi I own, for ease of debugging packets by brute-force equality. You might want to change this (especially if I'm on campus or have my raspi plugged in on your network), or better yet, set up your own little range of MAC packets nobody else has claimed yet. This is pretty easy to do, and currently defined in `include/offsets.svh` as well as `hdl/mac_rx.sv`.

- Assign an IP address to the FPGA. This one's pretty simple - figure out the DHCP range of your network and stick the IP address (`include/offsets.svh`) outside of the DHCP range but within the subnet. This should work out of the box...if not, send the IP address you've chosen some ARP requests (e.g. with `arping`) and verify that the system responds to ARP requests. Doing this will force the gateway to recognize the device as well, which might not happen the first time around.

  This might be more complicated on MITnet. Registering a static IP with IST should work without further configuration, and if that's too much work, a Linux machine can be configured as a router, and thus as a gateway to MITnet. This isn't too hard to set up if the Linux box has 2 NICs - there's lots of documentation online - and should work if you port forward your desired port on that Linux machine to the FPGA. You'll need a static IP if you want to talk to this thing over the wider, non-MITnet internet though.

- If you're sending packets over the internet rather than just a LAN (MITnet doesn't fall into this category), you'll have to port forward through your local router.

- *Optional:* Change the UDP port this thing listens on if you feel like it. This is also in `include/offsets.svh`.

Note that if you just wanna do Ethernet-level networking, no router configuration or other black magic is required, besides modifying my network stack to do just that. Just plug your board in and start sending packets, using the Media Access Controller modules and a shared packet buffer without any of the Internet Protocol / Address Resolution business. Ethertype 0x1234 is used for echo functionality, so try sending packets there (e.g. with raw sockets or Scapy) to sanity check your configuration.

11 Appendix D - Verilog Code

11.1 crc32_bzip2.sv

```
`timescale 1ns / 1ps

/* CRC32-BZIP2 implementation for the Ethernet
 * stack. Takes in two bits at a time.
 *
 * This code was generated with the help of CRCGEN.PL v1.7
 */

// Disclaimer: THESE DESIGNS ARE PROVIDED "AS IS" WITH NO WARRANTY
//             WHATSOEVER AND XILINX SPECIFICALLY DISCLAIMS ANY
//             IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR
//             A PARTICULAR PURPOSE, OR AGAINST INFRINGEMENT.
//
// Copyright (c) 2001,2002 Xilinx, Inc.  All rights reserved.

module crc32_bzip2(
    input logic[1:0]    d,
    input logic         calc,
    input logic         init,
    input logic         d_valid,
    input logic         clk,
    input logic         reset,

    output logic[31:0]  crc_reg,
    output logic[1:0]   crc
    );


    ↪ ////////////////////////////////////////////////////////////////////////
    // Internal Signals

    ↪ ////////////////////////////////////////////////////////////////////////
    wire   [31:0] next_crc;


    ↪ ////////////////////////////////////////////////////////////////////////
    // Infer CRC-32 registers
    //
    // The crc_reg register stores the CRC-32 value.
    // The crc register is the most significant 2 bits of the
    // CRC-32 value.
    //
    // Truth Table:
    //
    ↪ -----+---------+----------+-----------------------------------------
    // calc | d_valid | crc_reg  | crc
```

```
//
↪  -----+---------+----------+-------------------------------------------
//  0   |     0   | crc_reg  | crc
//  0   |     1   |  shift   | bit-swapped, complimented msbyte of
↪  crc_reg
//  1   |     0   | crc_reg  | crc
//  1   |     1   | next_crc | bit-swapped, complimented msbyte of
↪  next_crc
//
↪  -----+---------+----------+-------------------------------------------
//

↪  ////////////////////////////////////////////////////////////////////////////

always @ (posedge clk or posedge reset)
begin
   if (reset) begin
      crc_reg <= 32'hFFFFFFFF;
      crc     <= 2'hF;
   end

   else if (init) begin
      crc_reg <= 32'hFFFFFFFF;
      crc     <=  2'hF;
   end

   else if (calc & d_valid) begin
      crc_reg <= next_crc;
      crc     <= ~{next_crc[30], next_crc[31]};
   end

   else if (~calc & d_valid) begin
      crc_reg <=  {crc_reg[29:0], 2'hF};
      crc     <= ~{crc_reg[28], crc_reg[29]};
   end
end


↪  ////////////////////////////////////////////////////////////////////////////
// CRC XOR equations

↪  ////////////////////////////////////////////////////////////////////////////

assign next_crc[0] = d[1] ^ crc_reg[30];
assign next_crc[1] = d[0] ^ d[1] ^ crc_reg[30] ^ crc_reg[31];
assign next_crc[2] = crc_reg[31] ^ crc_reg[30] ^ d[1] ^ d[0] ^
↪  crc_reg[0];
assign next_crc[3] = crc_reg[31] ^ d[0] ^ crc_reg[1];
assign next_crc[4] = crc_reg[30] ^ d[1] ^ crc_reg[2];
```

```
    assign next_crc[5] = crc_reg[3] ^ d[0] ^ d[1] ^ crc_reg[30] ^
    ↪  crc_reg[31];
    assign next_crc[6] = crc_reg[31] ^ d[0] ^ crc_reg[4];
    assign next_crc[7] = crc_reg[30] ^ crc_reg[5] ^ d[1];
    assign next_crc[8] = crc_reg[30] ^ crc_reg[31] ^ crc_reg[6] ^ d[0] ^
    ↪  d[1];
    assign next_crc[9] = d[0] ^ crc_reg[7] ^ crc_reg[31];
    assign next_crc[10] = crc_reg[30] ^ crc_reg[8] ^ d[1];
    assign next_crc[11] = crc_reg[31] ^ crc_reg[30] ^ d[1] ^ crc_reg[9] ^
    ↪  d[0];
    assign next_crc[12] = d[0] ^ d[1] ^ crc_reg[10] ^ crc_reg[30] ^
    ↪  crc_reg[31];
    assign next_crc[13] = d[0] ^ crc_reg[11] ^ crc_reg[31];
    assign next_crc[14] = crc_reg[12];
    assign next_crc[15] = crc_reg[13];
    assign next_crc[16] = crc_reg[14] ^ d[1] ^ crc_reg[30];
    assign next_crc[17] = crc_reg[31] ^ d[0] ^ crc_reg[15];
    assign next_crc[18] = crc_reg[16];
    assign next_crc[19] = crc_reg[17];
    assign next_crc[20] = crc_reg[18];
    assign next_crc[21] = crc_reg[19];
    assign next_crc[22] = d[1] ^ crc_reg[20] ^ crc_reg[30];
    assign next_crc[23] = d[0] ^ d[1] ^ crc_reg[30] ^ crc_reg[31] ^
    ↪  crc_reg[21];
    assign next_crc[24] = d[0] ^ crc_reg[22] ^ crc_reg[31];
    assign next_crc[25] = crc_reg[23];
    assign next_crc[26] = crc_reg[24] ^ d[1] ^ crc_reg[30];
    assign next_crc[27] = crc_reg[31] ^ crc_reg[25] ^ d[0];
    assign next_crc[28] = crc_reg[26];
    assign next_crc[29] = crc_reg[27];
    assign next_crc[30] = crc_reg[28];
    assign next_crc[31] = crc_reg[29];
endmodule
```

11.2 display_controller.sv

```
`timescale 1ns / 1ps

module display_controller(
    input logic         reset_in,
    input logic         clock_in,
    input logic [15:0]  frame_delay,

    input logic [7:0]   pkt_buf_in [1518 - 1:0], // ETH_MTU - 1
    input logic         pkt_buf_doorbell_in,

    output logic x_sclk,
    output logic x_mosi,
    output logic x_cs,
    output logic y_sclk,
    output logic y_mosi,
    output logic y_cs,

    output logic r_pwm,
    output logic g_pwm,
    output logic b_pwm,

    output logic frame_sync
    );

    // Number of bits to allocate for X, Y, R, G, B data
    parameter X_LENGTH = 16;
    parameter Y_LENGTH = 16;

    parameter B_LENGTH = 8;
    parameter G_LENGTH = 8;
    parameter R_LENGTH = 8;

    // BRAM framebuffer
    logic        bram_select; // 0 to read from 0 and write to 1, 1 to
    ↪  read from 1 and write to 0
    logic [14:0] bram_0_addr, bram_1_addr;
    logic [14:0] bram_0_max_addr, bram_1_max_addr;
    logic [63:0] bram_0_data_in, bram_1_data_in;
    logic [63:0] bram_0_data_out, bram_1_data_out;

    blk_mem_gen_0 bram0 (.addra(bram_0_addr),
                         .clka(clock_in),
                         .dina(bram_0_data_in),
                         .douta(bram_0_data_out),
                         .wea(bram_select),
                         .ena(1));

    blk_mem_gen_0 bram1 (.addra(bram_1_addr),
```

```
                    .clka(clock_in),
                    .dina(bram_1_data_in),
                    .douta(bram_1_data_out),
                    .wea(!bram_select),
                    .ena(1));

// Data extraction from packet buffer
logic [63:0] current_flattened_packet;
assign current_flattened_packet = {pkt_buf_in[0], pkt_buf_in[1],
                                   pkt_buf_in[2], pkt_buf_in[3],
                                   pkt_buf_in[4], pkt_buf_in[5],
                                   pkt_buf_in[6], pkt_buf_in[7]};

// SPI controllers
logic x_start, y_start;
logic x_busy, y_busy;
logic [15:0] x, y;
logic [7:0] r, g, b;

spi x_spi_controller(
        .reset_in(reset_in),
        .clock_in(clock_in),
        .data_in(x),
        .data_length_in(X_LENGTH),
        .start_in(x_start),
        .busy_out(x_busy),

        .sclk_out(x_sclk),
        .mosi_out(x_mosi),
        .cs_out(x_cs));

spi y_spi_controller(
        .reset_in(reset_in),
        .clock_in(clock_in),
        .data_in(y),
        .data_length_in(Y_LENGTH),
        .start_in(y_start),
        .busy_out(y_busy),

        .sclk_out(y_sclk),
        .mosi_out(y_mosi),
        .cs_out(y_cs));

// PWM controllers

pwm r_pwm_controller(
        .reset_in(reset_in),
        .clock_in(clock_in),
        .value(r),
```

```
        .pwm_out(r_pwm));

pwm g_pwm_controller(
        .reset_in(reset_in),
        .clock_in(clock_in),
        .value(g),
        .pwm_out(g_pwm));

pwm b_pwm_controller(
        .reset_in(reset_in),
        .clock_in(clock_in),
        .value(b),
        .pwm_out(b_pwm));

// State Machine
logic [18:0] frame_delay_counter;
logic old_pkt_buf_doorbell; // Used to track when the packet doorbell
↪  changes

always_ff @(posedge clock_in) begin
    if(reset_in) begin
        bram_select <= 0;

        bram_0_addr <= 0;
        bram_1_addr <= 0;
        bram_0_max_addr <= 0;
        bram_1_max_addr <= 0;
        bram_0_data_in <= 0;
        bram_1_data_in <= 0;

        old_pkt_buf_doorbell <= 0;

        frame_delay_counter <= 0;
        x_start <= 0;
        y_start <= 0;
        frame_sync <= 0;
    end else begin
        // state machine that takes data from the packet buffer and
        ↪  stashes it in BRAM

        if (pkt_buf_doorbell_in && !old_pkt_buf_doorbell) begin
            // load into BRAM
            if(bram_select) begin
                bram_0_data_in <= current_flattened_packet;
                bram_0_addr <= bram_0_addr + 1;
                bram_0_max_addr <= bram_0_addr + 1;
            end else begin
                bram_1_data_in <= current_flattened_packet;
                bram_1_addr <= bram_1_addr + 1;
```

```
            bram_1_max_addr <= bram_1_addr + 1;
        end

        // switch BRAM if we need to
        if (current_flattened_packet[63:56] == 8'h02) begin
            if(bram_select) begin
                bram_1_max_addr <= 0;
                bram_1_addr <= 0;
            end else begin
                bram_0_max_addr <= 0;
                bram_0_addr <= 0;
            end
            bram_select <= !bram_select;
        end
    end
    old_pkt_buf_doorbell <= pkt_buf_doorbell_in;

    // state machine that gets pulls data from the active BRAM and
    ↪  writes it to the display
    frame_delay_counter <= frame_delay_counter + 1;

    if(frame_delay_counter == frame_delay - 2) begin
        r = bram_select ? bram_1_data_out[7:0] :
        ↪  bram_0_data_out[7:0];
        g = bram_select ? bram_1_data_out[15:8] :
        ↪  bram_0_data_out[15:8];
        b = bram_select ? bram_1_data_out[23:16] :
        ↪  bram_0_data_out[23:16];
        y = bram_select ? bram_1_data_out[39:24] :
        ↪  bram_0_data_out[39:24];
        x = bram_select ? bram_1_data_out[55:40] :
        ↪  bram_0_data_out[55:40];

        x_start <= 1;
        y_start <= 1;
    end

    if(frame_delay_counter == frame_delay - 1) begin
        x_start <= 0;
        y_start <= 0;
        frame_sync <= ~frame_sync;
    end

    if(frame_delay_counter == frame_delay) begin
        frame_delay_counter <= 0;
        // update bram address, wrapping around if necessary
        if(!bram_select) begin
            if(bram_0_addr == bram_0_max_addr) begin
                bram_0_addr <= 0;
```

```
                end else begin
                    bram_0_addr <= bram_0_addr + 1;
                end
            end

            if(bram_select) begin
                if(bram_1_addr == bram_1_max_addr) begin
                    bram_1_addr <= 0;
                end else begin
                    bram_1_addr <= bram_1_addr + 1;
                end
            end
        end
    end
end
endmodule
```

11.3 hex_display.sv

```
`timescale 1ns / 1ps

module hex_display (
    input               clk_in,      // system clock
    input [31:0]        data_in,     // 8 hex numbers, msb first
    output logic [6:0]  seg_out,     // seven segment display output
    output logic [7:0]  strobe_out   // digit strobe
    );

    localparam bits = 13;

    logic [bits:0] counter = 0;  // clear on power up

    logic [6:0] segments[15:0]; // 16 7 bit memorys
    assign segments[0]  = 7'b100_0000;  // inverted logic
    assign segments[1]  = 7'b111_1001;  // gfedcba
    assign segments[2]  = 7'b010_0100;
    assign segments[3]  = 7'b011_0000;
    assign segments[4]  = 7'b001_1001;
    assign segments[5]  = 7'b001_0010;
    assign segments[6]  = 7'b000_0010;
    assign segments[7]  = 7'b111_1000;
    assign segments[8]  = 7'b000_0000;
    assign segments[9]  = 7'b001_1000;
    assign segments[10] = 7'b000_1000;
    assign segments[11] = 7'b000_0011;
    assign segments[12] = 7'b010_0111;
    assign segments[13] = 7'b010_0001;
    assign segments[14] = 7'b000_0110;
    assign segments[15] = 7'b000_1110;

    always_ff @(posedge clk_in) begin
      // Here I am using a counter and select 3 bits which provides
      // a reasonable refresh rate starting the left most digit
      // and moving left.
      counter <= counter + 1;
      case (counter[bits:bits-2])
          3'b000: begin  // use the MSB 4 bits
                  seg_out <= segments[data_in[31:28]];
                  strobe_out <= 8'b0111_1111 ;
                 end

          3'b001: begin
                  seg_out <= segments[data_in[27:24]];
                  strobe_out <= 8'b1011_1111 ;
                 end

          3'b010: begin
```

```
                seg_out <= segments[data_in[23:20]];
                strobe_out <= 8'b1101_1111 ;
               end
        3'b011: begin
                seg_out <= segments[data_in[19:16]];
                strobe_out <= 8'b1110_1111;
               end
        3'b100: begin
                seg_out <= segments[data_in[15:12]];
                strobe_out <= 8'b1111_0111;
               end

        3'b101: begin
                seg_out <= segments[data_in[11:8]];
                strobe_out <= 8'b1111_1011;
               end

        3'b110: begin
                seg_out <= segments[data_in[7:4]];
                strobe_out <= 8'b1111_1101;
               end
        3'b111: begin
                seg_out <= segments[data_in[3:0]];
                strobe_out <= 8'b1111_1110;
               end

    endcase
   end
endmodule
```

11.4 ipv4_csum.sv

```
`timescale 1ns / 1ps

module ipv4_csum(input logic clk,
                 input logic rst,

                 input logic[7:0] pktbuf[1517:0],
                 input logic pkt_valid,

                 output logic[31:0] csum,
                 output logic csum_valid
                 );

    `include "offsets.svh"

    /* All parameters here */
    parameter ST_WAIT = 2'b00;
    parameter ST_CALC = 2'b01;
    parameter ST_FINISH = 2'b11;

    /* All logics */
    logic[1:0] state;
    logic[3:0] counter;
    logic[3:0] i;

    /* All clocked logic */
    always_ff @(posedge clk) begin
        if(rst == 1'b1) begin
            csum <= 0;
            csum_valid <= 0;
            state <= ST_WAIT;
            i <= 0;
            counter <= 0;
        end else if(state == ST_WAIT) begin
            if(pkt_valid == 1'b1) begin

                // Grab the IHL and load it into the counter
                // This has already been validated, i.e. the IHL isn't
                // outside of the bounds imposed by IPv4 header
                // Multiply * 32 bit words, divide into bytes from bits =>
                ↪  32/8 => 4
                counter <= pktbuf[IPV4_VSN_IHL][IPV4_IHL_TOP:IPV4_IHL_BOT]
                ↪  * 4;
                state <= ST_CALC;
                csum <= 0;

            end else begin
                csum <= 0;
                state <= ST_WAIT;
```

```
                counter <= 0;
            end
            csum_valid <= 0;
            i <= 0;

        // Compute the IPV4 checksum, asserting csum_valid
        // for a single cycle when completed. Then, reset.
        end else if(state == ST_CALC) begin
            // Last cycle?
            if(counter < 2) begin
                // Add the leftover byte, if any
                if(counter == 1) csum <= csum + pktbuf[ETH_DATA_START +
                ↪  i];
                state <= ST_FINISH;
            end else begin
                i <= i + 2;
                counter <= counter - 2;
                csum <= csum + pktbuf[ETH_DATA_START + i] +
                ↪  pktbuf[ETH_DATA_START + i + 1];
            end
            csum_valid <= 0;

        end else if(state == ST_FINISH) begin
            if(csum >> 16 != 0) begin
                csum <= (csum & 16'hffff) + (csum >> 16);
            end

            if(pkt_valid == 1'b0) state <= ST_WAIT;
            counter <= 0;
            i <= 0;
            csum_valid <= 1;
        end
    end

endmodule
```

11.5 mac_rx.sv

```
`timescale 1ns / 1ps

module mac_rx(
              input logic clk,
              input logic reset,

              input logic phy_crsdv,
              input logic[1:0] phy_rxd,

              output logic axi_rx_valid,
              output logic[1:0] axi_rx_data
              );

    /* All parameters here */
    parameter ST_IDLE     = 3'h0;
    parameter ST_PREAMBLE = 3'h1;
    parameter ST_FCEVENT  = 3'h2;
    parameter ST_DST      = 3'h3;
    parameter ST_BADDST   = 3'h4;
    parameter ST_SRC      = 3'h5;
    parameter ST_ETYPE    = 3'h6;
    parameter ST_DATA     = 3'h7;

    // My MAC address: b8:27:eb:a4:30:73
    parameter MAC_LEN     = 24;
    parameter ETYPE_LEN   = 8;
    parameter THIS_MAC    = 48'hb8_27_eb_a4_30_73;
    parameter CRC_RESIDUE = 32'hc704dd7b;

    /* All logics here */
    logic docrcsampling;
    logic[31:0] currentcrc;
    logic[31:0] counter;
    logic[47:0] macbuf;
    logic[2:0] state;

    /* All submodules here */
    crc32_bzip2          rx_crc(.crc_reg(currentcrc),
                                .d(phy_rxd),
                                .calc(docrcsampling),
                                .init(phy_crsdv == 1'b1 && phy_rxd ==
                                ↪  2'b11 && state == ST_PREAMBLE),
                                .d_valid(1'b1),
                                .clk(clk),
                                .reset(reset));

    // ILA: suggested debugging configuration
```

```
// ila_0 rxila(.clk(clk), .probe0(state), .probe1(phy_rxd),
↪  .probe2(currentcrc));

/* All clocked logic */
always_ff @(posedge clk) begin
    if(reset == 1'b1) begin
        state <= ST_IDLE;
        docrcsampling <= 1'b0;
        counter <= 32'b0;
        macbuf <= 47'b0;

        axi_rx_valid <= 1'b0;
        axi_rx_data <= 2'b0;

    end else begin
        case(state)
            ST_IDLE: begin
                axi_rx_valid <= 1'b0;
                if(phy_crsdv == 1'b1 && phy_rxd == 2'b01) state <=
                ↪  ST_PREAMBLE;
                else if(phy_crsdv == 1'b1 && phy_rxd == 2'b10) state
                ↪  <= ST_FCEVENT;
            end

            ST_PREAMBLE: begin
                if(phy_crsdv == 1'b1 && phy_rxd == 2'b11) begin
                    state <= ST_DST;
                    counter <= 32'd40;
                    docrcsampling <= 1'b1;
                end
                else if(phy_crsdv == 1'b1 && phy_rxd == 2'b10) state
                ↪  <= ST_FCEVENT;
                else if(phy_crsdv == 1'b0) state <= ST_IDLE;
            end

            ST_FCEVENT: begin
                if(phy_crsdv == 1'b0) state <= ST_IDLE;
            end

            ST_DST: begin
                if(phy_crsdv == 1'b0) begin
                    axi_rx_valid <= 1'b1;
                    axi_rx_data <= 2'b00;
                    state <= ST_IDLE;
                    docrcsampling <= 1'b0;
                    macbuf <= 48'b0;
                    counter <= 32'b0;
                end else begin
                    axi_rx_valid <= 1'b1;
```

```
            axi_rx_data <= phy_rxd;
            if(counter == 6) begin      // 6 due to somewhat
            ↪  backwards byte ordering
                // Promiscuous mode not supported; it would go
                ↪  here if so
                state <= (THIS_MAC ==
                ↪  {macbuf[47:8],phy_rxd,macbuf[5:0]}) ?
                ↪  ST_SRC :
                        ({macbuf[47:8],phy_rxd,macbuf[5:0]}
                        ↪  == 48'hff_ff_ff_ff_ff_ff) ?
                        ↪  ST_SRC : ST_BADDST;
                counter <= 32'b0;
                macbuf <= 48'b0;
            end else begin
                macbuf[counter +: 2] <= phy_rxd;
                counter <= (counter == 46 | counter == 38 |
                ↪  counter == 30 | counter == 22 | counter ==
                ↪  14) ? counter - 14 : counter + 2;
            end
        end
    end

    ST_BADDST: begin
        docrcsampling <= 1'b0;
        if(phy_crsdv == 1'b0) state <= ST_IDLE;
    end

    // We WILL need this!
    ST_SRC: begin
        if(phy_crsdv == 1'b0) begin
            state <= ST_IDLE;
            docrcsampling <= 1'b0;
            counter <= 32'b0;
            axi_rx_valid <= 1'b1;
            axi_rx_data <= 2'b00;
        end else begin
            axi_rx_valid <= 1'b1;
            axi_rx_data <= phy_rxd;
            counter <= counter + 1;
            if(counter == MAC_LEN - 1) begin
                state <= ST_ETYPE;
                counter <= 32'b0;
            end
        end
    end

    ST_ETYPE: begin
        if(phy_crsdv == 1'b0) begin
            state <= ST_IDLE;
```

```
                    docrcsampling <= 1'b0;
                    counter <= 32'b0;
                    axi_rx_valid <= 1'b1;
                    axi_rx_data <= 2'b00;
                end else begin
                    axi_rx_valid <= 1'b1;
                    axi_rx_data <= phy_rxd;
                    counter <= counter + 1;
                    if(counter == ETYPE_LEN - 1) begin
                        state <= ST_DATA;
                        counter <= 32'b0;
                    end
                end
            end


            ST_DATA: begin
                if(phy_crsdv == 1'b1) begin
                    axi_rx_valid <= 1'b1;
                    axi_rx_data <= phy_rxd;
                end else begin
                    docrcsampling <= 1'b0;
                    axi_rx_data <= (currentcrc == CRC_RESIDUE) ?
                    ↪  2'b11: 2'b00;
                    state <= ST_IDLE;
                    axi_rx_valid <= 1'b1;
                end
            end

        endcase
    end
end


endmodule
```

11.6 mac_rx_ifc.sv

```
`timescale 1ns / 1ps

module mac_rx_ifc(
                  input logic clk,
                  input logic rst,

                  input logic rx_axi_valid,
                  input logic[1:0] rx_axi_data,

                  output logic[7:0] pktbuf[1517:0],
                  output logic[10:0] pktbuf_maxaddr,
                  output logic doorbell
    );

    /* All parameters here */
    parameter ST_WAIT   = 1'b0;
    parameter ST_RX     = 1'b1;

    /* All logics here */
    logic[10:0] pktbuf_addr;
    logic[2:0] bytectr;
    logic state;

    /* All preliminary assignments here */

    /* All submodules here */

    /* All clocked logic here */
    always_ff @(posedge clk) begin
        if(rst == 1'b1) begin
            foreach(pktbuf[i]) pktbuf[i] <= 0;
            pktbuf_maxaddr <= 0;
            pktbuf_addr <= 0;
            bytectr <= 0;
            doorbell <= 0;
            state <= ST_WAIT;
        end else begin
            if(state == ST_WAIT) begin
                if(rx_axi_valid == 1'b1) begin
                    state <= ST_RX;
                    bytectr <= 2;
                    foreach(pktbuf[i]) pktbuf[i] <= (i == 0) ?
                    ↪  {6'b0,rx_axi_data} : 0;
                    pktbuf_addr <= 0;
                    pktbuf_maxaddr <= 0;
                    doorbell <= 0;

                end else begin
```

```
                    state <= ST_WAIT;
                    bytectr <= 0;
                    // Keep the pktbuf the same
                    pktbuf_addr <= 0;
                    // No change to maxaddr
                    // No change to doorbell
                end
            end else begin
                if(rx_axi_valid == 1'b0) begin
                    // Pktbuf remains untouched
                    // Collect some garbage
                    bytectr <= 0;
                    pktbuf_addr <= 0;
                    state <= ST_RX;

                    // CRC check
                    if(rx_axi_data != 2'b11) begin
                        pktbuf_maxaddr <= 0;
                        doorbell <= 0;
                    end else begin
                        // Strip the CRC and mark the packet valid
                        pktbuf_maxaddr <= pktbuf_addr - 5;
                        doorbell <= 1;
                    end
                end else begin
                    // Counter management
                    if(bytectr == 6) begin
                        bytectr <= 0;
                        pktbuf_addr <= pktbuf_addr + 1;
                    end else begin
                        bytectr <= bytectr + 2;
                        // No change to pktbuf addressing
                    end

                    // Insert the new nibble
                    pktbuf[pktbuf_addr][bytectr +: 2] <= rx_axi_data;
                    state <= ST_RX;
                    doorbell <= 0;
                    // Maxaddr unchanged
                end
            end
        end
    end

endmodule
```

11.7 mac_tx.sv

```
`timescale 1ns / 1ps

module mac_tx(
    input logic         clk,
    input logic         reset,

    input logic         axi_valid,
    input logic[1:0]    axi_din,
    output logic        axi_ready,

    output logic        phy_txen,
    output logic[1:0]   phy_txd
    );

    /* All parameters first */
    parameter ST_IDLE           = 3'h0;
    parameter ST_PREAMBLE       = 3'h1;
    parameter ST_DATA           = 3'h2;
    parameter ST_PAD            = 3'h3;
    parameter ST_VERIFY         = 3'h4;
    parameter ST_IFRAME_GAP     = 3'h5;

    parameter PREAMBLE_DIBITS   = 32;
    parameter MIN_DATA_DIBITS   = 240;
    parameter CRC_DIBITS        = 16;
    parameter IFG_PERIOD        = 48;

    /* All logics here */
    logic[31:0] counter;
    logic[2:0]  state;
    logic       is_calculation_cycle;
    logic       is_valid_cycle;
    logic       soft_reset;

    logic[31:0] cumulative_crc;
    logic[1:0]  stepwise_crc;

    /* All preliminary assignments here */
    assign is_calculation_cycle = (state == ST_DATA && (axi_valid ||
    ↪  counter < MIN_DATA_DIBITS))
                                || (state == ST_PREAMBLE && counter ==
                                ↪  PREAMBLE_DIBITS)
                                || (state == ST_PAD && counter <
                                ↪  MIN_DATA_DIBITS);
    /* All submodules here */
    crc32_bzip2     tx_crc(.clk(clk),
                           .reset(reset),
                           .d(axi_din),
```

```
                        .d_valid(is_valid_cycle),
                        .calc(is_calculation_cycle),
                        .init(soft_reset),
                        .crc_reg(cumulative_crc),
                        .crc(stepwise_crc));

/* Suggested ILA configuration:
tx_ila          tila(.clk(clk),
                     .probe0(axi_din),
                     .probe1(state),
                     .probe2(phy_txd),
                     .probe3(is_calculation_cycle),
                     .probe4(phy_txen),
                     .probe5(stepwise_crc),
                     .probe6(counter));

*/

/* All clocked logic here */
always_ff @(posedge clk) begin
    if(reset == 1'b1) begin
        axi_ready <= 1'b0;
        phy_txen <= 1'b0;
        phy_txd <= 2'b0;

        counter <= 32'd1;
        state <= ST_IDLE;
        is_valid_cycle <= 1'b0;
        soft_reset <= 1'b0;

    end else begin
        if(state == ST_IDLE) begin
            if(axi_valid == 1'b1) begin
                state <= ST_PREAMBLE;
            end else begin
                state <= ST_IDLE;
            end

            counter <= 32'd1;
            axi_ready <= 1'b0;
            phy_txen <= 1'b0;
            phy_txd <= 2'b0;

            is_valid_cycle <= 1'b0;
            soft_reset <= 1'b1;

        end else if(state == ST_PREAMBLE) begin
            if(counter == PREAMBLE_DIBITS) begin
                phy_txd <= 2'b11;
```

```
            counter <= 32'd0;
            state <= ST_DATA;
            soft_reset <= 1'b0;
            axi_ready <= 1'b1;
            is_valid_cycle <= 1'b1;
        end else begin
            phy_txd <= 2'b01;
            counter <= counter + 1;
            state <= ST_PREAMBLE;
            soft_reset <= 1'b1;
            axi_ready <= 1'b0;
            is_valid_cycle <= 1'b0;
        end

        phy_txen <= 1'b1;

    end else if(state == ST_DATA) begin
        if(axi_valid == 1'b0) begin

            if(counter < MIN_DATA_DIBITS) begin
                state <= ST_PAD;
                phy_txd <= 2'b00;
                counter <= counter + 1;
            end else begin

                state <= ST_VERIFY;
                phy_txd <= stepwise_crc;
                counter <= 32'd1;
            end

            axi_ready <= 1'b0;

        end else begin
            state <= ST_DATA;
            phy_txd <= axi_din;
            counter <= counter + 1;
            axi_ready <= 1'b1;
        end
        is_valid_cycle <= 1'b1;
        phy_txen <= 1'b1;
        soft_reset <= 1'b0;


    end else if(state == ST_PAD) begin
        if(counter < MIN_DATA_DIBITS) begin
            phy_txd <= 2'b00;
            counter <= counter + 1;
            state <= ST_PAD;
        end else begin
```

```
                phy_txd <= stepwise_crc;
                counter <= 32'd1;
                state <= ST_VERIFY;
            end

            is_valid_cycle <= 1'b1;
            axi_ready <= 1'b0;
            phy_txen <= 1'b1;
            soft_reset <= 1'b0;


        end else if(state == ST_VERIFY) begin
            if(counter <= CRC_DIBITS) begin
                phy_txen <= 1'b1;
                phy_txd <= stepwise_crc;
                counter <= counter + 1;
                state <= ST_VERIFY;
                soft_reset <= 1'b0;
                is_valid_cycle <= 1'b1;
            end else begin
                phy_txen <= 1'b1;
                phy_txd <= 2'b11;
                counter <= 32'd1;
                state <= ST_IFRAME_GAP;
                soft_reset <= 1'b1;
                is_valid_cycle <= 1'b0;
            end

            axi_ready <= 1'b0;

        end else if(state == ST_IFRAME_GAP) begin
            if(counter == IFG_PERIOD) begin
                counter <= 32'd1;
                state <= ST_IDLE;
            end else begin
                counter <= counter + 1;
                state <= ST_IFRAME_GAP;
            end

            axi_ready <= 1'b0;
            phy_txen <= 1'b0;
            phy_txd <= 2'b00;
            is_valid_cycle <= 1'b0;
            soft_reset <= 1'b1;

        end
    end
  end
endmodule
```

11.8 mac_tx_ifc.sv

```
`timescale 1ns / 1ps

module mac_tx_ifc(
                    input logic clk,
                    input logic rst,

                    input logic tx_axi_ready,
                    output logic tx_axi_valid,
                    output logic[1:0] tx_axi_data,

                    input logic[7:0] pktbuf[1517:0],
                    input logic[10:0] pktbuf_maxaddr,
                    input logic doorbell,
                    output logic available
    );

    /* All parameters here */
    parameter ST_WAIT   = 1'b0;
    parameter ST_TX     = 1'b1;

    /* All logics here */
    logic[2:0] bytectr;
    logic[10:0] pktbuf_addr;
    logic state;

    /* All preliminary assignments here */

    /* All submodules here */
    /* Suggested ILA configuration:
    tx_ifc_ila      ila(.clk(clk),
                        .probe0(state),
                        .probe1(tx_axi_data),
                        .probe2(tx_axi_valid),
                        .probe3(tx_axi_ready));
    */
    /* All clocked logic here */
    always_ff @(posedge clk) begin
        if(rst == 1'b1) begin
            bytectr <= 0;
            pktbuf_addr <= 0;
            tx_axi_valid <= 0;
            tx_axi_data <= 0;
            state <= ST_WAIT;
            available <= 1;
        end else begin
            if(state == ST_WAIT) begin

                if(doorbell == 1'b1) begin
```

```
            // Prepare for transit
            state <= ST_TX;
            pktbuf_addr <= 0;
            bytectr <= 0;
            available <= 0;
        end else begin
            // Idle time
            state <= ST_WAIT;
            pktbuf_addr <= 0;
            bytectr <= 0;
            available <= 1;
        end

        tx_axi_valid <= 0;
        tx_axi_data <= 0;

    end else begin
        available <= 0;
        // Can send new data down the line
        if(tx_axi_ready == 1'b1) begin

            // Need to advance the counts
            if(bytectr == 6) begin
                bytectr <= 0;
                pktbuf_addr <= pktbuf_addr + 1;

                // Are we done?
                if(pktbuf_addr == pktbuf_maxaddr) begin
                    state <= ST_WAIT;
                    tx_axi_valid <= 1'b0;
                end else begin
                    state <= ST_TX;
                    tx_axi_valid <= 1'b1;
                end

                // Regardless, update the data as if we r
                ↪  advancing
                tx_axi_data <= pktbuf[pktbuf_addr + 1][1:0];

            // Don't need to advance the counts
            end else begin
                bytectr <= bytectr + 2;
                tx_axi_valid <= 1'b1;
                state <= ST_TX;
                tx_axi_data <= pktbuf[pktbuf_addr][bytectr + 2 +:
                ↪  2];
                // No change to pktbuf_addr
            end
```

```
                // Can't send new data down the line yet
                end else begin
                    tx_axi_valid <= 1'b1;
                    state <= ST_TX;
                    tx_axi_data <= pktbuf[pktbuf_addr][bytectr +: 2];
                    // No change to bytectr, pktbuf_addr
                end
            end
        end
    end


endmodule
```

11.9 netstack.sv

```
`timescale 1ns / 1ps

/* Top level networking stack module */
module netstack(
    input logic         sys_clk,
    input logic         sys_rst,

    input logic         eth_crsdv,
    input logic[1:0]    eth_rxd,

    output logic        eth_txen,
    output logic[1:0]   eth_txd,
    output logic        eth_refclk,
    output logic        eth_rstn,

    output logic[15:0] errno,
    output logic[7:0] databuf[1517:0],
    output logic[15:0] databuf_len,
    output logic databuf_valid
    );

    `include "offsets.svh"
    `include "errno.svh"

    /* All parameters here */
    parameter ST_PULL = 2'b00;
    parameter ST_PROC = 2'b01;
    parameter ST_PUSH = 2'b10;
    parameter ST_CONFIRM = 2'b11;

    /* All logics here */
    // No reset

    logic rx_axi_valid;
    logic[1:0] rx_axi_dout;

    logic tx_axi_ready;
    logic tx_axi_valid;
    logic[1:0] tx_axi_din;

    logic[7:0] rx_pktbuf[ETH_MTU - 1:0];
    logic[10:0] rx_pktbuf_maxaddr;
    logic rx_doorbell;

    logic tx_available;

    logic csum_out_valid;
    logic[31:0] csum_out;
```

```
// Reset required
logic[7:0] tx_pktbuf[ETH_MTU - 1:0];
logic[10:0] tx_pktbuf_maxaddr;
logic tx_doorbell;
logic[1:0] state;

logic[7:0] arp_mha[ETH_ADDRSZ-1:0];
logic[7:0] arp_mpa[IPV4_ADDRSZ-1:0];

logic csum_in_valid;

/* All preliminary assignments here */
assign eth_refclk = sys_clk;
assign eth_rstn = ~sys_rst;

/* Suggested ILA configurations:
eth_ila             ila(.clk(sys_clk),
                        .probe0(state),
                        .probe1(csum_in_valid),
                        .probe2(csum_out),
                        .probe3(csum_out_valid),
                        .probe4(eth_rxd));
*/

ipv4_csum           ipcsum(.clk(sys_clk),
                           .rst(sys_rst),
                           .pktbuf(rx_pktbuf),
                           .pkt_valid(csum_in_valid),
                           .csum(csum_out),
                           .csum_valid(csum_out_valid));

mac_tx              resptx(.clk(sys_clk),
                           .reset(sys_rst),
                           .axi_valid(tx_axi_valid),
                           .axi_din(tx_axi_din),
                           .axi_ready(tx_axi_ready),
                           .phy_txen(eth_txen),
                           .phy_txd(eth_txd));

mac_rx                reqrx(.clk(sys_clk),
                            .reset(sys_rst),
                            .phy_crsdv(eth_crsdv),
                            .phy_rxd(eth_rxd),
                            .axi_rx_valid(rx_axi_valid),
                            .axi_rx_data(rx_axi_dout));


mac_rx_ifc          rcvifc(.clk(sys_clk),
```

```
                            .rst(sys_rst),
                            .rx_axi_valid(rx_axi_valid),
                            .rx_axi_data(rx_axi_dout),
                            .pktbuf(rx_pktbuf),
                            .pktbuf_maxaddr(rx_pktbuf_maxaddr),
                            .doorbell(rx_doorbell));

mac_tx_ifc          tsmifc(.clk(sys_clk),
                            .rst(sys_rst),
                            .tx_axi_valid(tx_axi_valid),
                            .tx_axi_data(tx_axi_din),
                            .tx_axi_ready(tx_axi_ready),
                            .pktbuf(tx_pktbuf),
                            .pktbuf_maxaddr(tx_pktbuf_maxaddr),
                            .doorbell(tx_doorbell),
                            .available(tx_available));

/* Clocked logic here */
always_ff @(posedge sys_clk) begin

    /* Main system runtime loop */
    if(sys_rst == 1'b1) begin
        foreach(databuf[i]) databuf[i] <= 0;
        databuf_valid <= 0;
        databuf_len <= 0;

        foreach(tx_pktbuf[i]) tx_pktbuf[i] <= 0;
        tx_pktbuf_maxaddr <= 0;
        tx_doorbell <= 0;
        state <= ST_PULL;
        errno[15:0] <= EIDLE;

        csum_in_valid <= 0;

        foreach(arp_mha[i]) arp_mha[i] <= 0;
        foreach(arp_mpa[i]) arp_mpa[i] <= 0;

    end else begin
        if(state == ST_PULL) begin
            // Don't transmit right now.
            tx_doorbell <= 0;

            // Wait for new stuff
            if(rx_doorbell == 1'b1) begin
                // Process packets into the buffer immediately.
                // They will stay valid for ~48 clock cycles, so this
                // approach will work for us.
```

```
// Immediately, clear out the databuf and set it to
↪  invalid,
// so there's some hang time to reset the framebuffer
↪  etc.
foreach(databuf[i]) databuf[i] <= 0;
databuf_valid <= 0;
databuf_len <= 0;

// Get the ethertype of the packet and check it
// Currently supported services are ARP, ECHOSVC, IPv4

if(rx_pktbuf[ETH_ETYPE_MAX] == ETH_ARP_ETYPE_2 &&
   rx_pktbuf[ETH_ETYPE_MIN] == ETH_ARP_ETYPE_1) begin

    // ADDRESS RESOLUTION PROTOCOL v4
    // If the H/PTYPE or H/PLEN fields don't match,
    ↪  drop the packet
    if(rx_pktbuf[ARP_HTYPE_MAX] != ARP_ETH_HTYPE_2)
    ↪  state <= ST_CONFIRM;
    else if(rx_pktbuf[ARP_HTYPE_MIN] !=
    ↪  ARP_ETH_HTYPE_1) state <= ST_CONFIRM;

    else if(rx_pktbuf[ARP_PTYPE_MAX] !=
    ↪  ARP_IPV4_PTYPE_2) state <= ST_CONFIRM;
    else if(rx_pktbuf[ARP_PTYPE_MIN] !=
    ↪  ARP_IPV4_PTYPE_1) state <= ST_CONFIRM;

    else if(rx_pktbuf[ARP_HLEN_OFF] != ARP_ETH_HLEN)
    ↪  state <= ST_CONFIRM;
    else if(rx_pktbuf[ARP_PLEN_OFF] != ARP_IPV4_PLEN)
    ↪  state <= ST_CONFIRM;

    // Valid ARP packet received for Ethernet + IPv4.
    // Proceed to process it.
    else begin
        // If we are the TPA, update the mapping.
        if(rx_pktbuf[ARP_TPA_MAX] == IPV4_MYADDR_4 &&
           rx_pktbuf[ARP_TPA_MAX - 1] == IPV4_MYADDR_3
           ↪  &&
           rx_pktbuf[ARP_TPA_MAX - 2] == IPV4_MYADDR_2
           ↪  &&
           rx_pktbuf[ARP_TPA_MAX - 3] ==
           ↪  IPV4_MYADDR_1) begin

            arp_mha <=
            ↪  rx_pktbuf[ARP_SHA_MAX:ARP_SHA_MIN];
            ↪  // Thoughts and prayers
            arp_mpa <=
            ↪  rx_pktbuf[ARP_SPA_MAX:ARP_SPA_MIN];
```

```
errno[15:0] <= EARPQ;

// If this is a request, issue a reply.
// Form a from-scratch reply in the
↪  tx_pktbuf and ring the doorbell
if(rx_pktbuf[ARP_OPCODE_MAX] ==
↪  ARP_OPCODE_REQ &&
   rx_pktbuf[ARP_OPCODE_MIN] ==
   ↪  ARP_OPCODE_UPPER) begin

    // MAC header
    tx_pktbuf[ETH_DST_MAX:ETH_DST_MIN] <=
    ↪  rx_pktbuf[ARP_SHA_MAX:ARP_SHA_MIN];

    tx_pktbuf[ETH_SRC_MIN + 5] <=
    ↪  ETH_MYADDR_6; // ETH_SRC_MAX
    tx_pktbuf[ETH_SRC_MIN + 4] <=
    ↪  ETH_MYADDR_5;
    tx_pktbuf[ETH_SRC_MIN + 3] <=
    ↪  ETH_MYADDR_4;
    tx_pktbuf[ETH_SRC_MIN + 2] <=
    ↪  ETH_MYADDR_3;
    tx_pktbuf[ETH_SRC_MIN + 1] <=
    ↪  ETH_MYADDR_2;
    tx_pktbuf[ETH_SRC_MIN + 0] <=
    ↪  ETH_MYADDR_1;

    tx_pktbuf[ETH_ETYPE_MAX:ETH_ETYPE_MIN]
    ↪  <=
    ↪  rx_pktbuf[ETH_ETYPE_MAX:ETH_ETYPE_MIN];

    // ARP header
    tx_pktbuf[ARP_HTYPE_MAX:ARP_HTYPE_MIN]
    ↪  <=
    ↪  rx_pktbuf[ARP_HTYPE_MAX:ARP_HTYPE_MIN];
    tx_pktbuf[ARP_PTYPE_MAX:ARP_PTYPE_MIN]
    ↪  <=
    ↪  rx_pktbuf[ARP_PTYPE_MAX:ARP_PTYPE_MIN];
    tx_pktbuf[ARP_HLEN_OFF] <=
    ↪  rx_pktbuf[ARP_HLEN_OFF];
    tx_pktbuf[ARP_PLEN_OFF] <=
    ↪  rx_pktbuf[ARP_PLEN_OFF];
    tx_pktbuf[ARP_OPCODE_MIN] <=
    ↪  ARP_OPCODE_UPPER;
    tx_pktbuf[ARP_OPCODE_MAX] <=
    ↪  ARP_OPCODE_RESP;

    // ARP body: S/TPA, S/THA
```

```
                    tx_pktbuf[ARP_SHA_MIN + 5] <=
                    ↪  ETH_MYADDR_6;
                    tx_pktbuf[ARP_SHA_MIN + 4] <=
                    ↪  ETH_MYADDR_5;
                    tx_pktbuf[ARP_SHA_MIN + 3] <=
                    ↪  ETH_MYADDR_4;
                    tx_pktbuf[ARP_SHA_MIN + 2] <=
                    ↪  ETH_MYADDR_3;
                    tx_pktbuf[ARP_SHA_MIN + 1] <=
                    ↪  ETH_MYADDR_2;
                    tx_pktbuf[ARP_SHA_MIN + 0] <=
                    ↪  ETH_MYADDR_1;

                    tx_pktbuf[ARP_THA_MAX:ARP_THA_MIN] <=
                    ↪  rx_pktbuf[ARP_SHA_MAX:ARP_SHA_MIN];

                    tx_pktbuf[ARP_SPA_MIN + 0] <=
                    ↪  IPV4_MYADDR_1;
                    tx_pktbuf[ARP_SPA_MIN + 1] <=
                    ↪  IPV4_MYADDR_2;
                    tx_pktbuf[ARP_SPA_MIN + 2] <=
                    ↪  IPV4_MYADDR_3;
                    tx_pktbuf[ARP_SPA_MIN + 3] <=
                    ↪  IPV4_MYADDR_4;

                    tx_pktbuf[ARP_TPA_MAX:ARP_TPA_MIN] <=
                    ↪  rx_pktbuf[ARP_SPA_MAX:ARP_SPA_MIN];

                    // Done. Send the packet with proper
                    ↪  padding
                    tx_pktbuf_maxaddr <= ARP_HDR_END;
                    state <= ST_PUSH;
                end else begin
                    state <= ST_CONFIRM;
                end
            end else begin
                state <= ST_CONFIRM;
            end

        end

    end else if(rx_pktbuf[ETH_ETYPE_MAX] ==
    ↪  ETH_IPV4_ETYPE_2 &&
                rx_pktbuf[ETH_ETYPE_MIN] ==
                ↪  ETH_IPV4_ETYPE_1) begin

        // Configure the packet for reception: validate
        ↪  some basic
        // parameters before sending it on to processing
```

```
// Is the version IPv4? If not drop

↪  if(rx_pktbuf[IPV4_VSN_IHL][IPV4_VSN_TOP:IPV4_VSN_BOT]
↪  != 4) begin
    state <= ST_CONFIRM;
    errno[15:0] <= ENOV4;
end

// Is the IHL at least 5 (*32 bits) => at least 20
↪  bytes?

↪  if(rx_pktbuf[IPV4_VSN_IHL][IPV4_IHL_TOP:IPV4_IHL_BOT]
↪  < 5) begin
    errno[15:0] <= ESHDR;
    state <= ST_CONFIRM;
end

// Is the TTL zero? If so, drop the packet.
// If/when we implement full ICMP, send a time
↪  exceeded back
if(rx_pktbuf[IPV4_TTL] == 0) begin
    state <= ST_CONFIRM;
    errno[15:0] <= EDEAD;
end

// Is the protocol UDP? We don't do anything else.
if(rx_pktbuf[IPV4_PROTOCOL] != IPV4_UDP_PROTO)
↪  begin
    errno[15:0] <= EPROT;
    state <= ST_CONFIRM;
end

// Does the packet utilize fragmentation? If so,
↪  drop

↪  if(rx_pktbuf[IPV4_FLAGS_STARTOF][IPV4_FLAGS_DNF_OFFSET]
↪  != 1) begin
    errno[15:0] <= EFRAG;
    state <= ST_CONFIRM;
end

// Are we the intended receiver, or is the packet
↪  a broadcast packet?
if((rx_pktbuf[IPV4_DSTADDR_4] != IPV4_MYADDR_4 ||
    rx_pktbuf[IPV4_DSTADDR_3] != IPV4_MYADDR_3 ||
    rx_pktbuf[IPV4_DSTADDR_2] != IPV4_MYADDR_2 ||
    rx_pktbuf[IPV4_DSTADDR_1] != IPV4_MYADDR_1) &&
   (rx_pktbuf[IPV4_DSTADDR_4] != 8'hff ||
    rx_pktbuf[IPV4_DSTADDR_3] != 8'hff ||
```

```
                    rx_pktbuf[IPV4_DSTADDR_2] != 8'hff ||
                    rx_pktbuf[IPV4_DSTADDR_1] != 8'hff)) begin

                    // If the above fails, we are not the intended
                    ↪  receiver,
                    // drop the packet entirely
                    errno[15:0] <= EPDST;
                    state <= ST_CONFIRM;

                end else begin
                    // Initial checks pass
                    // Start checksum calculation, move to
                    ↪  processing state
                    csum_in_valid <= 1;
                    state <= ST_PROC;
                end
            end else if(rx_pktbuf[ETH_ETYPE_MAX] ==
            ↪  ETH_ECHOSVC_ETYPE_2 &&
                        rx_pktbuf[ETH_ETYPE_MIN] ==
                        ↪  ETH_ECHOSVC_ETYPE_1) begin

                // Echo service. Ethertype 1234.
                // Swap MAC address and ping the client back.
                tx_pktbuf[ETH_DST_MAX:ETH_DST_MIN] <=
                ↪  rx_pktbuf[ETH_SRC_MAX:ETH_SRC_MIN];
                tx_pktbuf[ETH_SRC_MAX:ETH_SRC_MIN] <=
                ↪  rx_pktbuf[ETH_DST_MAX:ETH_DST_MIN];
                tx_pktbuf[ETH_MTU - 1:ETH_ETYPE_MIN] <=
                ↪  rx_pktbuf[ETH_MTU - 1:ETH_ETYPE_MIN];

                tx_pktbuf_maxaddr <= rx_pktbuf_maxaddr;
                state <= ST_PUSH;
                errno[15:0] <= EECHO;

            end else begin
                // Unknown ethertype. Drop packet.
                state <= ST_CONFIRM;
                errno[15:0] <= EETYP;
            end
        end // else don't do anything lol

    // IP Offload Engine sits here
    // Checks off the checksum and then dissects what's inside.
    end else if(state == ST_PROC) begin
        // No valid data on the input side,
        // wait for clocked output
        csum_in_valid <= 0;
        if(csum_out_valid == 1) begin
            // Verify the checksum against the input.
```

```
            if(csum_out != 0) begin
                state <= ST_CONFIRM;
                errno[15:0] <= ECSUM;
            end else begin
                // Handle UDP. Particularly, check that the
                ↪  destination
                // port is 0xA455 ;) Disregard the UDP checksum,
                ↪  and set
                // databuf_len to be the proper length-of-packet
                ↪  to kill off
                // padding + header data.
                if(rx_pktbuf[UDP_DPORT_MIN] != UDP_MYPORT_1 ||
                   rx_pktbuf[UDP_DPORT_MAX] != UDP_MYPORT_2) begin
                    errno[15:0] <= EPORT;
                    state <= ST_CONFIRM;
                end else begin
                    // Packet is GOOD!
                    if(rx_pktbuf[UDP_DATA_START] == 8'h02)
                    ↪  errno[15:0] <= EGOOD;

                    // Set databuf here. Treat it as ephemerally
                    // as the RX packet buffer. Obtain its length
                    // by taking the length bit of the UDP header,
                    // and subtracting the 8 bytes of header

                    // Still not sure if this is correct now. Only
                    // thing I can confirm is that UDP packets are
                    ↪  in
                    // fact checking out via IPv4 checksum and
                    ↪  CRC32, plus
                    // port stuffs.
                    databuf_len <=
                    ↪  {rx_pktbuf[UDP_PKTLEN_MIN],rx_pktbuf[UDP_PKTLEN_MAX]}
                    ↪  - 8;
                    databuf_valid <= 1;
                    foreach(rx_pktbuf[i]) databuf[i] <= (i +
                    ↪  UDP_DATA_START >= ETH_MTU) ? 0
                                                          :
                                                          ↪  rx_pktbuf[i
                                                          ↪  +
                                                          ↪  UDP_DATA_START]
                    state <= ST_CONFIRM;
                end

            end

        end

    end else if(state == ST_PUSH) begin
```

```
                if(tx_available == 1'b1) begin

                    // tx_pktbuf already set. Go!
                    // tx_pktbuf_maxaddr also already set
                    tx_doorbell <= 1'b1;
                    state <= ST_CONFIRM;

                end // else do nothing

            // Block until new data comes in
            end else if(state == ST_CONFIRM) begin
                tx_doorbell <= 1'b0;
                if(rx_doorbell == 1'b0) state <= ST_PULL;
            end
        end
    end
endmodule
```

11.10 pwm.sv

```
`timescale 1ns / 1ps

module pwm(
    input logic         reset_in,
    input logic         clock_in,
    input logic [7:0]   value,

    output logic        pwm_out
    );

    parameter FREQ_PRESCALER = 4;
    logic [4:0] freq_prescaler_counter;
    logic [7:0] duty_cycle_counter;

    always_ff @(posedge clock_in) begin
        if (reset_in) begin
            pwm_out <= 0;
            freq_prescaler_counter <= 0;
            duty_cycle_counter <= 0;
        end else begin
            freq_prescaler_counter <= freq_prescaler_counter + 1;

            if(freq_prescaler_counter == FREQ_PRESCALER) begin
                freq_prescaler_counter <= 0;
                duty_cycle_counter <= duty_cycle_counter + 1;
            end

            pwm_out <= duty_cycle_counter >= value ? 0:1;
        end
    end
endmodule
```

11.11 spi.sv

```
`timescale 1ns / 1ps

module spi(
    input logic         reset_in,
    input logic         clock_in,
    input logic [15:0]  data_in,
    input logic [5:0]   data_length_in,
    input logic         start_in,
    output logic        busy_out,

    output logic        sclk_out,
    output logic        mosi_out,
    output logic        cs_out
    );

    parameter SCLK_PRESCALER = 100;
    logic [12:0] prescale_counter;
    logic [5:0]  data_index;

    always_ff @(posedge clock_in) begin
        if (reset_in) begin
            data_index <= 0;
            prescale_counter <= 0;

            busy_out <= 0;
            sclk_out <= 0;
            mosi_out <= 0;
            cs_out <= 1;
        end

        if (!reset_in && start_in && !busy_out) begin
                prescale_counter <= 0;
                mosi_out <= 0;
                data_index <= 0;
                cs_out <= 0;
            busy_out <= 1;
        end

        if (!reset_in && busy_out)begin

            // if we're not resetting, check if we need to start sending
            ↪  out data
            if (prescale_counter == 0) begin
                mosi_out <= data_in[data_length_in - 1 - data_index];
            end

            if (prescale_counter == SCLK_PRESCALER/2) begin
                sclk_out <= !sclk_out;
```

```
            end

            if (prescale_counter == SCLK_PRESCALER) begin
                sclk_out <= !sclk_out;
                data_index <= data_index + 1;
                prescale_counter <= 0;
            end

            else begin
                prescale_counter <= prescale_counter + 1;
            end

            if (data_index == data_length_in) begin
                mosi_out <= 0;
                cs_out <= 1;
                busy_out <= 0;
            end
        end
    end
endmodule
```

11.12 sys_top.sv

```
`timescale 1ns / 1ps

/* Top level everything */
module sys_top(
    input logic         clk_100mhz,

    input logic         btnc,
    input logic[15:0]   sw,
    output logic[7:0]   ja, jb, jc, jd,

    output logic [7:0]  an,
    output logic        ca, cb, cc, cd, ce, cf, cg, dp,

    input logic         eth_crsdv,
    input logic[1:0]    eth_rxd,

    output logic        eth_txen,
    output logic[1:0]   eth_txd,
    output logic        eth_refclk,
    output logic        eth_rstn,


    output logic[15:0] led
    );

    `include "offsets.svh"
    `include "errno.svh"

    logic sys_clk;
    logic sys_rst;

    logic[7:0] netout[ETH_MTU - 1:0];
    logic[15:0] netout_len;
    logic netout_valid;
    logic[31:0] display_data;
    logic[6:0] segments;

    /* All preliminary assignments */
    assign display_data = {netout[0],
                           netout[1],
                           netout[2],
                           netout[3]};

    assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];

    /* All submodules here */
```

```
    hex_display           hd(.clk_in(sys_clk),
                             .data_in(display_data),
                             .seg_out(segments),
                             .strobe_out(an));

    eth_refclk_divider  erd(.in(clk_100mhz),
                            .out(sys_clk),
                            .reset(sys_rst));

    display_controller  dctl(.reset_in(sys_rst),
                             .clock_in(sys_clk),
                             .frame_delay(sw),
                             .pkt_buf_in(netout),
                             .pkt_buf_doorbell_in(netout_valid),

                             .x_sclk(jd[0]),
                             .x_mosi(jd[5]),
                             .x_cs(jd[4]),
                             .y_sclk(),
                             .y_mosi(jd[6]),
                             .y_cs(),

                             .r_pwm(jd[1]),
                             .g_pwm(jd[2]),
                             .b_pwm(jd[3]),
                             .frame_sync(jd[7]));

    netstack            netstack(.sys_clk(sys_clk),
                                 .sys_rst(sys_rst),
                                 .eth_crsdv(eth_crsdv),
                                 .eth_rxd(eth_rxd),
                                 .eth_txen(eth_txen),
                                 .eth_txd(eth_txd),
                                 .eth_refclk(eth_refclk),
                                 .eth_rstn(eth_rstn),
                                 .errno(led),
                                 .databuf(netout),
                                 .databuf_len(netout_len),
                                 .databuf_valid(netout_valid));

    /* All preliminary assignments here */
    assign sys_rst = btnc;

endmodule
```

12 Appendix E - Python Code

12.1 laser.py

```
# Laser Projector Preprocessor
#
# Used to turn arbitary images and video into laser trajectories, that are
↪  either sent over the network to a
# FPGA or stored locally as a CSV, PNG, or COE file.
#
# fischerm@mit.edu, Fall 2020

from sys import argv
import cv2
import numpy as np
import trajectory_planner as tp
import os
import socket

cv2.setUseOptimized(True) # no idea if this does anything but hahaa openCV
↪  go brrrrr

# parse input options
if len(argv) == 1:
    raise SystemExit("No options specified. Try 'laser.py -h' for more
    ↪  information.")

if '-h' in argv:
    print("""Usage: python3 laser.py [OPTION]
    -i      input source (required). path to file if source is a file, or
↪  number if webcam.
    -o      output destination. path to directory, but use -n for
↪  streaming to projector.
    -t      output file type. options include any combination of 'png' or
↪  'csv' or 'coe'.
    -n      network address of the laser, if it's output is desired.
    -q      quiet mode. will not show the rendered frame.

examples:
    python3 app/laser.py -i input.jpg -o output/ -n f0:0d:be:ef:ba:be
    python3 app/laser.py -i 0 -n f0:0d:be:ef:ba:be""")
    exit()

if '-i' not in argv:
    raise SystemExit("No input file specified, specify one with -i or run
    ↪  'laser.py -h' for more information.")

if '-t' in argv and '-o' not in argv:
```

```
    raise SystemExit("Output type specified, but no output directory
    ↪  specified. Specify one with -o or run 'laser.py -h' for more
    ↪  information.")


# Set capture to whatever was passed in with the -i option
input_filename = argv[argv.index('-i') + 1]
try:
    source = int(input_filename)
except:
    source = input_filename

cap = cv2.VideoCapture(source)
if (cap.isOpened() == False):
    raise SystemExit(f"Error opening input file {input_filename}")

# Set output directory to whatever was passed in with the -o option
output_directory = argv[argv.index('-o') + 1] if '-o' in argv else None

# Set output type to whatever was passed in with the -t option
output_types = []
if '-t' in argv:
    for arg in argv[argv.index('-t')+1:]:
        if len(arg) != 2 and '-' not in arg:
            output_types.append(arg)
        else:
            break

# Create network interface file descriptor if the -n parameter is
↪  specified
if '-n' in argv:
    fd = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Resize image to desired x and y resolution. Used to decrease the number
↪  of points in the
# image and to maintain aspect ratio
def prep_frame(frame, desired_x_resolution, desired_y_resolution):
    x_resolution = frame.shape[0]
    y_resolution = frame.shape[1]

    x_scale_factor = desired_x_resolution/x_resolution
    y_scale_factor = desired_y_resolution/y_resolution

    if x_scale_factor > y_scale_factor:
        return cv2.resize(frame, None, fx=y_scale_factor,
        ↪  fy=y_scale_factor)

    else:
```

```
        return cv2.resize(frame, None, fx=x_scale_factor,
        ↪  fy=x_scale_factor)

# Export image to directory path as n.png, where n is the number of .png
↪  files in directory plus one
def save_png(path, image):
    files = [i for i in os.listdir(path) if '.png' in i]
    filename = f'{path}/{len(files)}.png'
    cv2.imwrite(filename, image)

# Export image to directory path as n.csv, where n is the number of .csv
↪  files in directory plus one
def save_csv(path, trajectory):
    import pandas as pd
    files = [i for i in os.listdir(path) if '.csv' in i]
    filename = f'{path}/{len(files)}.csv'
    pd.DataFrame(trajectory.astype(int)).to_csv(filename, header=False)

# Export image to directory path as n.coe, where n is the number of .coe
↪  files in directory plus one
def save_coe(path, trajectory):
    files = [i for i in os.listdir(path) if '.coe' in i]
    filename = f'{path}/{len(files)}.coe'

    output_lines =
    ↪  ['memory_initialization_radix=16;\n','memory_initialization_vector=\n']

    input_lines = trajectory.tolist()

    zero_pad = lambda input_str, length: '0'*(length - len(input_str)) +
    ↪  input_str

    for input_line_number, input_line in enumerate(input_lines):
        x, y, r, g, b = [format(int(i), 'x') for i in input_line]

        if input_line_number == len(input_lines) - 1:
            output_lines.append( zero_pad(x, 4) + zero_pad(y, 4) +
            ↪  zero_pad(r, 2) + zero_pad(g,2) + zero_pad(b,2) + ';')

        else:
            output_lines.append( zero_pad(x, 4) + zero_pad(y, 4) +
            ↪  zero_pad(r, 2) + zero_pad(g,2) + zero_pad(b,2) + ',\n')

    with open(filename, 'w') as output_file:
        output_file.writelines(output_lines)

def save_traj(path, trajectory):
    files = [i for i in os.listdir(path) if '.traj' in i]
    filename = f'{path}/{len(files)}.traj'
```

```
    output_lines = []

    input_lines = trajectory.tolist()
    for input_line_number, input_line in enumerate(input_lines):
        x, y, r, g, b = [int(i) for i in input_line]

        control = '02' if input_line_number == len(input_lines) - 1 else
        ↪  '01'

        x = format(65535 - (x*128), 'x')
        y = format(65535 - (y*128), 'x') # mirror y because galvos are
        ↪  oriented wierdly
        r = format(r, 'x')
        g = format(g, 'x')
        b = format(b, 'x')

        output_lines.append(control + zero_pad(x, 4) + zero_pad(y, 4) +
        ↪  zero_pad(r, 2) + zero_pad(g,2) + zero_pad(b,2) + '\n')

    with open(filename, 'w') as output_file:
        output_file.writelines(output_lines)

def send_trajectory_udp(fd, trajectory, destination_ip='142.79.194.65',
↪  port_number=42069):
    input_lines = trajectory.tolist()
    packet_list = []

    # Packet format is as follows:
    # Control (1 byte) - either 0x01 for adding to framebuffer, or 0x02 to
    ↪  swap framebuffers. Other values are invalid.
    # x (2 bytes)
    # y (2 bytes)
    # r (1 byte)
    # g (1 byte)
    # b (1 byte)

    # Total: 8 bytes

    for input_line_number, input_line in enumerate(input_lines):
        x, y, r, g, b = [int(i) for i in input_line]

        control = '02' if input_line_number == len(input_lines) - 1 else
        ↪  '01'

        x = format(65535 - (x*128), 'x')
        y = format(65535 - (y*128), 'x') # mirror y because galvos are
        ↪  oriented wierdly
        r = format(r, 'x')
```

```
        g = format(g, 'x')
        b = format(b, 'x')

        zero_pad = lambda input_str, length: '0'*(length - len(input_str))
        ↪  + input_str

        data = control + zero_pad(x, 4) + zero_pad(y, 4) + zero_pad(r, 2)
        ↪  + zero_pad(g,2) + zero_pad(b,2)
        fd.sendto(bytes.fromhex(data), (destination_ip, port_number))


while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        # Rescale frame to 512x512
        frame = prep_frame(frame, 512, 512)

        # Canny filtering
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)        # Convert
        ↪  image to grayscale
        gray_filtered = cv2.bilateralFilter(gray, 7, 50, 50)  # Smooth
        ↪  without removing edges
        edges = cv2.Canny(gray, 30, 120)                      # Apply
        ↪  canny filter
        edges_filtered = cv2.Canny(gray_filtered, 60, 120)

        # Trajectory Planning
        blur = cv2.blur(frame,(5,5))
        trajectory, degeneracies = tp.calculate_trajectory(edges_filtered)
        colorized_trajectory = tp.colorize_trajectory(blur, trajectory)
        rendered_trajectory = tp.draw_trajectory(np.zeros_like(frame),
        ↪  colorized_trajectory)

        # Stack images to display together for comparison
        edges_filtered_colored = cv2.cvtColor(edges_filtered,
        ↪  cv2.COLOR_GRAY2BGR)
        images = np.hstack((frame, edges_filtered_colored,
        ↪  rendered_trajectory))

        # Save the frame if option specified in argv
        if 'png' in output_types:
            save_png(output_directory, rendered_trajectory)

        if 'csv' in output_types:
            save_csv(output_directory, colorized_trajectory)

        if 'coe' in output_types:
            save_coe(output_directory, colorized_trajectory)
```

```
            if 'traj' in output_types:
                save_traj(output_directory, colorized_trajectory)

            # Write frame over the network, if the option is specified
            if '-n' in argv:
                send_trajectory_udp(fd, colorized_trajectory)

            # Display the resulting frame if -q not in options
            if '-q' not in argv:
                cv2.imshow('Frame', images)
                if cv2.waitKey(25) & 0xFF == ord('q'): # Press Q to exit
                    break

        else:
            break

cap.release()
cv2.destroyAllWindows()
```

12.2 trajectory_planning.py

```
# Laser Trajectory Planning Library
#
# Used to turn images into laser trajectories, which are then sent over
↪  the network to a TCP offload engine
# running on a Nexsys 4 DDR, which then pushes image data out to a RGB
↪  laser.
#
# fischerm@mit.edu, Fall 2020

import cv2
import numpy as np

# Draws each point trajectory on a template, showing what the image
↪  written to the laser should look like.
# Primiarily used for debug.
def draw_trajectory(template, trajectory, color=(255,0,0), popup=False):
    _, depth = trajectory.shape

    if depth == 2: # trajectory is monochrome, no color information is
    ↪  present in the row
        formatted_contour = np.expand_dims(trajectory, 1)
        render = cv2.drawContours(template, formatted_contour, -1, color,
        ↪  1)

        if popup:
            cv2.imshow('render.png', render)
            cv2.waitKey()
        return render

    if depth == 5: # trajectory is colored, BGR values are present
        render = template
        for row in trajectory:
            y, x, b, g, r = row
            render[x, y] = [b, g, r]

        if popup:
            cv2.imshow('render.png', render)
            cv2.waitKey()
        return render

# Draws each point trajectory on a template, but animated to showing what
↪  path is taken in drawing the trajectory.
# Primiarily used for debug.
def animate_trajectory(template, contour, speed):
    formatted_contour = np.expand_dims(contour, 1)
    for i in range(0, len(formatted_contour), speed):
        img = cv2.drawContours(template, formatted_contour[:i], -1,
        ↪  (0,255,0), 1)
```

```
            width = img.shape[0]
            height = img.shape[1]
            img = cv2.resize(img, (width*3, height*3), interpolation =
            ↪  cv2.INTER_AREA)
            cv2.imshow('labeled.png', img)

            # Press Q on keyboard to  exit
            if cv2.waitKey(5) & 0xFF == ord('q'):
                break

    # Draws each point trajectory on a template, but animated to showing what
    ↪  path is taken in drawing the trajectory.
    # Draws jumps between noncontinuous points with red lines, and draws
    ↪  continous points with green lines.
    # Primiarily used for debug.
    def animate_trajectory_with_jumps(template, contour, speed):
        formatted_contour = np.expand_dims(contour, 1)
        total_jumps = 0
        total_jump_distance = 0

        for i in range(0, len(formatted_contour) - 1, speed):
            current_x = contour[i][0]
            current_y = contour[i][1]
            next_x = contour[i+1][0]
            next_y = contour[i+1][1]

            if is_adjacent(current_x, current_y, next_x, next_y):
                template[current_y, current_x] = (0, 255, 0)

            else:
                template = cv2.line(template,(current_x, current_y),(next_x,
                ↪  next_y),(0,0,255),1)
                total_jumps += 1
                total_jump_distance += np.sqrt((current_x-next_x)**2 +
                ↪  (current_y-next_y)**2)

            width = template.shape[0]
            height = template.shape[1]
            thicco = cv2.resize(template, (width*3, height*3), interpolation =
            ↪  cv2.INTER_AREA)
            cv2.imshow('labeled.png', thicco)
            cv2.imwrite('output.png', thicco)
            print(f'total jumps: {total_jumps}')
            print(f'total distance jumped: {total_jump_distance}px')

            # Press Q on keyboard to  exit
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
```

```
# Image Processing Functions

# Take every point in the trajectory, and fill it with color information
↪  from the source image.
# Passing a blurred image here usually works best
def colorize_trajectory(img, trajectory):
    num_points, _ = trajectory.shape
    colors = np.zeros((num_points, 3), dtype=int)

    for i in range(num_points - 1):
        y_current, x_current = trajectory[i]
        y_next, x_next = trajectory[i+1]

        if is_adjacent(x_current, y_current, x_next, y_next):
            colors[i] = img[x_current, y_current]

        else:
            colors[i] = (0,0,0)
    return np.hstack((trajectory, colors))

# The trajectory planning algorithim, which is a nearest-neighbors
↪  implementation
def calculate_trajectory(img, start_x=0, start_y=0):
    # calculate the trajectory for the entire image, and export as list of
    ↪  x,y points
    # also export an array of all the degenercies found in the image
    binary_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]  #
    ↪  ensure image is binary
    contours = find_contours(binary_img)

    # if there aren't any contours in the image, just return nothing
    if len(contours) == 0:
        return np.asarray([[0, 0]]), np.asarray([])

    current_x = start_x
    current_y = start_y

    remaining_contours = [contour.tolist() for contour in contours]
    output_trajectory = []
    degeneracies = []

    while remaining_contours:
        next_contour_unordered = get_nearest_contour(remaining_contours,
        ↪  current_x, current_y)
        next_contour_ordered =
        ↪  get_reordered_contour(next_contour_unordered.tolist(),
        ↪  current_x, current_y)
```

```
            next_contour_ordered, next_degeneracies =
            ↪  order_contour(next_contour_ordered)
            remaining_contours.remove(next_contour_unordered.tolist())
            output_trajectory.append(next_contour_ordered)

            if next_degeneracies.tolist():
                for degeneracy in next_degeneracies.tolist():
                    degeneracies.append(degeneracy)

            current_x = output_trajectory[-1][-1][0]
            current_y = output_trajectory[-1][-1][1]

        return np.vstack(tuple(output_trajectory)), np.asarray(degeneracies)

    # Find all the connected points, and package them together into contours
    def find_contours(img):
        # splits image into a list of numpy arrays, each corresponding to a
        ↪  contour in the image
        num_labels, labels = cv2.connectedComponents(img)
        return [np.column_stack((np.where(labels == i)[::-1])) for i in
        ↪  range(1, np.max(labels))]

    # Take an unsorted list of points that are supposedly connected to each
    ↪  other, and order them
    # such that the nth point is adjacent to the n-1th point and the n+1th
    ↪  point.
    def order_contour(contour):
        # take first element of contour as starting, then keep finding
        ↪  adjacent points until the curve has been linearized
        # return contour afterwards as numpy array, as well as degeneracies
        remaining_points = contour.tolist()
        output_trajectory = [remaining_points[0]]

        degeneracies = [] # tracks the degenercies found

        while remaining_points:
            current_x = output_trajectory[-1][0] # the point that we're
            ↪  searching for
            current_y = output_trajectory[-1][1]

            # if this is the last point, add it to trajectory and exit
            if len(remaining_points) == 1:
                output_trajectory.append(remaining_points[0])
                remaining_points.remove(remaining_points[0])

            # if this isn't the last point, find the next adjacent point
            else:
                next_point = find_next_point(remaining_points, current_x,
                ↪  current_y)
```

```
            if(next_point): # if there is a point adjacent to the current
            ↪  one
                output_trajectory.append(next_point)
                remaining_points.remove(next_point)

            else:
                # found a degeneracy, back up algorithim to the most
                ↪  recent point that has an adjacency
                #print(f'Found degeneracy at ({current_x},{current_y})')

                degeneracies.append([current_x, current_y])
                for point in output_trajectory[::-1]:
                    if find_next_point(remaining_points, point[0],
                    ↪  point[1]):
                        output_trajectory.append(point)
                        break

    return np.asarray(output_trajectory), np.asarray(degeneracies)

# Find the next point in the contour that is adjacent to the point at (x,
↪  y)
def find_next_point(points, x, y):
    # Given some point in the contour, find the next (adjacent) point in
    ↪  the contour

    # try using four_level adjacency first, and if nothing is found, use
    ↪  eight_level adjacency
    for point in points:
        if is_adjacent(x, y, point[0], point[1], adjacency='four_level'):
            return point


    for point in points:
        if is_adjacent(x, y, point[0], point[1], adjacency='eight_level'):
            return point

    return None

# Returns true if (x1, y1) is adjacent to (x2, y2). four_level adacency
↪  does not allow diagonally connected
# points, but eight_level does.
def is_adjacent(x1, y1, x2, y2, adjacency='eight_level'):
    if(x1 == x2 and y1 ==y2):
        return False

    if(adjacency == 'four_level'):
        if (x1 == x2 and abs(y1-y2) == 1):
            return True
```

```
        if (y1 == y2 and abs(x1-x2) == 1):
            return True
        return False

    if(adjacency == 'eight_level'):
        if(abs(x1-x2) <= 1 and abs(y1-y2) <= 1):
            return True
        return False

# Reorders the contour such that the first point is the one closest to the
↪  (x,y) point passed in
def get_reordered_contour(contour, x, y):
    # figure out what index the nearest point occurs at
    closest_point = get_nearest_point([contour], x, y).tolist()[::-1]
    contour_list = contour#.tolist()

    index = contour_list.index(closest_point)
    reordered_contour = [contour_list[i%len(contour_list)] for i in
    ↪  range(index, index + len(contour_list))]
    return np.asarray(reordered_contour)

# Get the contour in which an (x,y) point is found
def get_nearest_contour(contours, x, y):
    # turns out numpy doesn't have a built in method for seeing if a row
    ↪  is in a matrix,
    # so instead we have to convert to a list first, which is slow. big
    ↪  sad.
    closest_point = get_nearest_point(contours, x, y).tolist()[::-1]
    contours_list = contours #[contour.tolist() for contour in contours]

    for contour in contours_list:
        if closest_point in contour:
            return np.asarray(contour)

# Find the nearest point on any contour to a specified (x,y) point
def get_nearest_point(contours, x, y):
    # put all x, y points into a list
    all_points = np.vstack(tuple(contours))

    # compute distances of all points
    dist_squared = lambda row: (row[0]-x)**2 + (row[1]-y)**2
    dist_squared_map = np.apply_along_axis(dist_squared, axis=1,
    ↪  arr=all_points)

    # find closest point
    closest_point = all_points[np.argmin(dist_squared_map)]
    return np.flip(closest_point) # for some reason x and y are reversed,
    ↪  so we should flip it before sending it out
```

nice