Network-Attached Laser Projector Preliminary Report

1st Fischer Moseley
Department of Physics
Massachusetts Institute of Technology
Cambridge, MA, USA
fischerm@mit.edu

2nd Jay Lang
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
jaytlang@mit.edu

***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.**

***Index Terms*—Digital systems, Field programmable gate arrays, Computer networks, Diode lasers, Optical projectors**

I. Physical Construction

The projector itself consists of:

- A RGB laser module. The multicolored beam is formed by combining the outputs of a 660nm, 520nm, and 450nm with a set of dichroic mirrors. This doesn't produce a true RGB beam, but it's close enough for our purposes.
- A galvanometer assembly, with a pair of movable orthogonally-mounted mirrors to steer the beam in the x and y directions.
- A pair of galvonometer drivers. These discipline the the mirror's position to a setpoint defined by an analog input voltage.
- Driver electronics, consisting of a set of two DACs that produce the control signals that steer the beam in the x and y directions, and three constant-current LED drivers, which regulate the current through each of the laser diode.
- A Nexys 4 DDR FPGA, from Digilent.

Eventually the projector will move from its present cardboard backplane to a proper enclosure. Per EHS regulations, this will include interlocks that interrupt power to the device if the enclosure is tampered with during operation.

II. Image Processing

To display an image or video on the wall, the source image must be converted from raster form to a trajectory for the laser to trace on the wall. This takes the form of an ordered list of x, y points, the generation of which happens using OpenCV in a Python script on a host computer. This occurs over a few steps.

- **Rescaling** - To handle arbitrarily sized input, the script rescales the input source to a resolution of 512x512. This simplifies the coordinate calculation math that's performed later in the pipeline.
- **Canny Filtering** - The image is converted to greyscale, and then processed by a Canny filter for edge detection.
- **Point Reordering** - The Canny filter outputs an image with the edges encoded as a mask - white pixels against a black background. The ordering of these pixels follows the image coordinate system, but the points are reordered such that adjacent pixels come after each other in the list. This takes the form of a nearest-neighbors algorithm, and it is incredibly slow as it occurs in Python. Moving it to C/C++ is a goal for the final project.

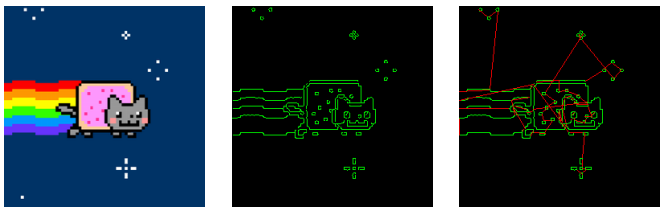


Fig. 1. Output of trajectory planning. On the left, the source image. In the center, the image to be rendered by the projector. On the right, the image to be rendered, but including the jumps between contours in red. Although the actual output is colorized, the trajectory here is shown as monochrome for clarity.

Currently the Python script only outputs trajectory information - since our drive electronics aren't complete yet, we've been prototyping with a red laser pointer. We have no way to enable/disable the laser, so we will add the ability to output full-color once the drive electronics are complete.

Once the trajectory has been calculated, it's sent out over the network. Each point is encapsulated as a network packet using our custom application layer network protocol, and then sent to the FPGA using the Scapy Python library. We hope to use the OS socket library once the networking offload engine is fully working.

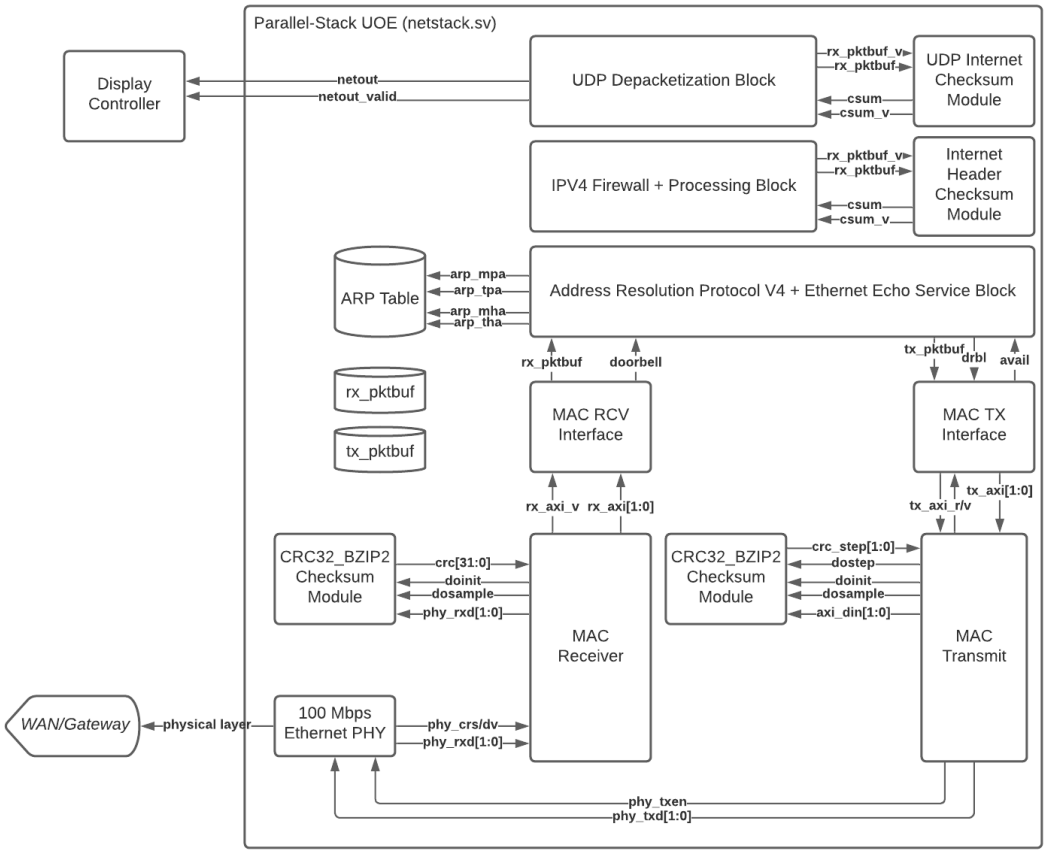


Fig. 2. Block diagram for the Network Offload Engine.

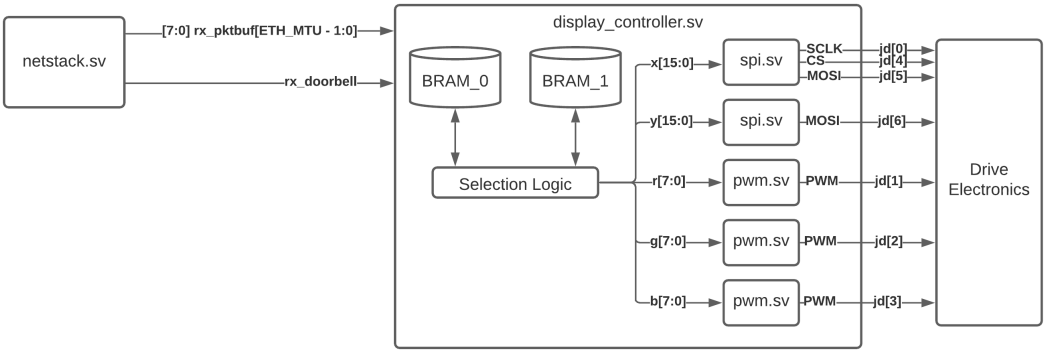


Fig. 3. Block diagram for the Display Controller.

III. Networking Offload Engine

A. 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.

B. 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.

C. Address Resolution

The Address Resolution Protocol (ARP) is implemented above the MAC layer according to RFC 826 [1], 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.

D. The Internet Protocol

Currently, the IPv4 protocol has yet to be implemented in accordance with RFC 791. [2]

E. User Datagram Protocol (UDP)

Currently, the User Datagram Protocol (UDP) has yet to be implemented in accordance with RFC 768. [3]

IV. Laser Display Module (Fischer)

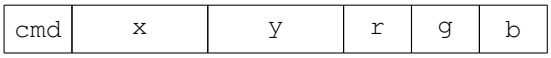
A. Framebuffer

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. The ultimate size of this BRAM is something that is still to be determined.

B. Packet Structure

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



- `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.

C. 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 has been transferred to a custom PCB, which is currently being fabricated at PCBWay.

REFERENCES

[1] An Ethernet Address Resolution Protocol. RFC 826, November 1982.

[2] Internet Program Protocol Specification. RFC 791, September 1981.

[3] User Datagram Protocol. RFC 768, November 1982.