Boston University
Electrical & Computer Engineering

EC463 Capstone Senior Design Project

First Prototype Testing Report

CyberTap

by

Team 2
CyberTap

Team Members:
Felipe Dale Figeman fdale@bu.edu
Alex Fatyga afatyga@bu.edu
Evan Lang evanlang@bu.edu
Noah Malhi malhin@bu.edu
Justin Morgan justinfm@bu.edu

Required Materials:

Hardware:

- 2x Raspberry Pi 3 (with Ethernet cable)
- TP-LINK TL-SG105E 5-Port Gigabit Easy Smart Network Switch
- Desktop PC
- Nexys A7 FPGA

Software:

- Xilinx C/C++ SDK 2019.1
- Node.js Web Client
    - main.html
        - Front end that receives data from server.js and puts it into the table
    - testUart.js
        - Back end that receives output through serial output, sends it to front end
- VHDL/Verilog
    - Microblaze server and other required Vivado modules such as AXI ethernet, clk wizard, and MII_to_RMII
    - Utilized Vivado IP's for the above

Setup:

The overall test was separated into two parts. The first part involved testing the end to end functionionality of the current networking set up with the FPGA Microblaze echo server. After loading the Microblaze echo server onto the FPGA, the desktop was connected to the FPGA via Ethernet and an app called Tera Term was used to send packets through the connection to the Microblaze server's IP address, 192.168.1.10 (on port 7). The packet payload was then not only echoed back to the source IP (Tera Term) but also sent to the serial port COM6 (UART). The serial port was then read by the node.js file testUart.js, which displayed the data in the web app. The success of this test was determined by whether or not all packets sent through Tera Term were correctly echoed by the FPGA (the payloads should match) and received by the web app interface via UART communication.

The second part of testing involved creating the simulated network by having one Raspberry Pi send TCP packets to the other through the network switch, and then monitoring the SPAN port on the switch (port 4 is the port set up to mirror all traffic coming out of port 3 on the switch) via Wireshark. The test's success metric was quantifiably measured by comparing the packets sent and received by the Pi's against the packets shown in Wireshark.

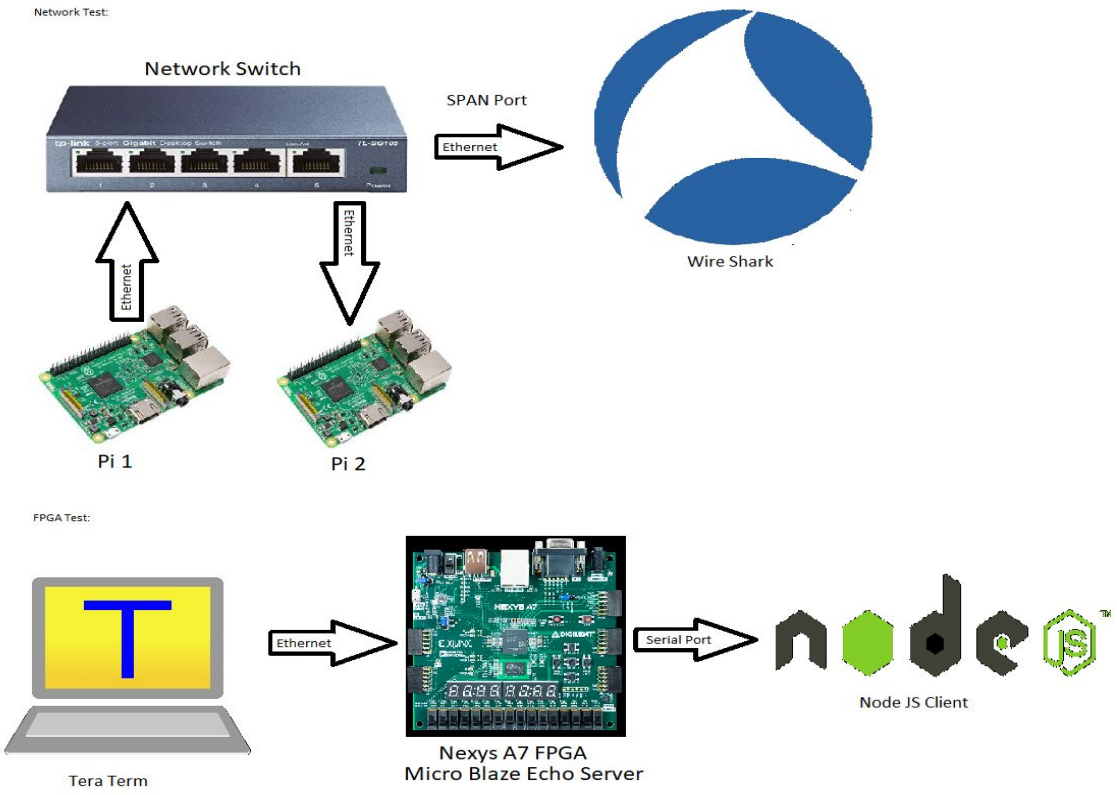

Figure 1: Testing Set up

Measurements:

Part 1: Echo

Packet Sent	FPGA blink? (Y/N)	Bytes Received on Web Client
123	Y	123
110	Y	110
463	Y	463
464	Y	464
489	Y	489
502	Y	502
508	Y	8
721	Y	1
1	Y	1
123	Y	23
Result: 94.56%		

Blink = 5 Points, % Of packets received = 0-5 points 10 blinks +(41/46) packets received = 10(5+(41/46)*5) = 94.56

Part 2: Test Server Proof of Concept

Packet Sent (Payload)	Matched on Client Pi? (Y/N)	Shown on Wireshark? (Y/N)
543	Y	Y
123	Y	Y
924	Y	Y
823	Y	Y
543	Y	Y
924	Y	Y
123	Y	Y
124	Y	Y
123	Y	Y
823	Y	Y
Result: 100%		

Our first test shows some packet loss. Although it is within range of the set success rate, there is still room for improvement which will be done through changing the design. Our second test shows no packet loss as expected.

Conclusions:

Based on Part 1 of testing, it's now confirmed that the throughput provided by UART is too slow to transmit the packet payloads. Using UART for packet reception confirmation is being considered as a possible option, as the ease of use from STDOUT being redirected through the serial port will greatly simplify debugging. Additionally, meaningful tests of realistically simulated networks through the FPGA cannot be performed until packet sniffing is added. Currently, the FPGA can only grab packets addressed to its specific IP address and port. This prevents the FPGA from accepting SPAN port traffic, rendering network tap test impossible. Additionally, once the FPGA is able to receive the traffic we will be able to increase the realism of our tests by not having a set wait time between packets, or one in the ms range.

After research and discussion with experienced FPGA engineers, the implementation of the design has been revised. Instead of utilizing a motherboard and implementing a form of Direct Memory Access, a 'System on Chip' (SoC) FPGA will be used. Using DMA comes with many potential issues and hardships. Synchronization of the system would be a key issue since as the FPGA receives ethernet data there will be different speeds of memory transfers within the FPGA, to the motherboard and with the DMA itself. Implementing the DMA for the intended use would be difficult to implement in the given time due to the various troubleshooting that would have to be done. By using a SoC FPGA with an ARM core, any software programming that would be needed can be done on chip. For example the web server could be hosted via specific FPGAs.

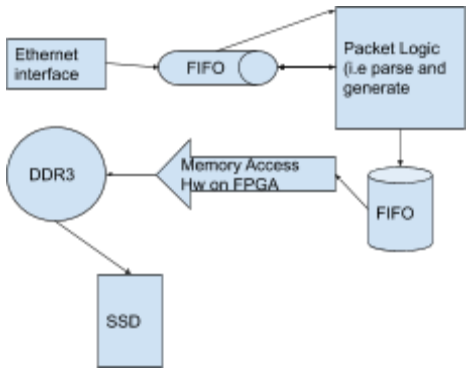

Figure 2: A System Utilizing On Chip Hardware

The first step is the same as the original concept with regards to ethernet interfacing. This interface allows for the traffic flow which is readable to the device. Since the flow in can be faster than the speed of parsing and metadata generation, a buffer (an FPGA FIFO structure) will be utilized. The buffer acts as a channel between the input and the parser so that no packets are lost. It can be dynamically changed in size and can be opened and closed like a pipe. The packet logic module can tell the buffer when it's ready for more data thus allowing the buffer to empty. After the packet metadata generation, the data is sent to another FIFO structure for the same synchronization reason. From there, using the FPGA's memory access (not the same as the CPU DMA), it will write the data to the onboard DDR3 where it will then be sent to the SD slot to be sent to the external storage. With this new system, since the flow of data isn't constant and typically going to be more in bursts as networking data is sent every micro time interval, this allows for a more synchronized and controlled system that ensures no data is lost. This also allows the memory transfers to be easier to implement and more efficient.