Individual Contribution Report

ECE532 2021, Quinn Smith

0.0 What I Did

0.1 Partitioning

The work on our project was partitioned to optimize for completion time and concurrency. Since I had some experience with networking and distributed systems I was responsible for building the networking for the accelerator. Interface design was done as a group in order to ensure the systems subcomponents were compatible with each other. The work on the team was very balanced but each team member had their workload peak at a different time during the semester. The majority of my technical tasks were in the first half of the project because we needed working accelerator networking for the mid project demo.

0.2 Primary Contribution

My primary role on the project was developing a networking stack in pure hardware for use on the accelerator. The purpose of the accelerator's networking stack was to communicate with the load balancer and get inference data on and off the FPGA. The networking stack was initially designed to support IP packets but was upgraded to support UDP/IP packets instead.

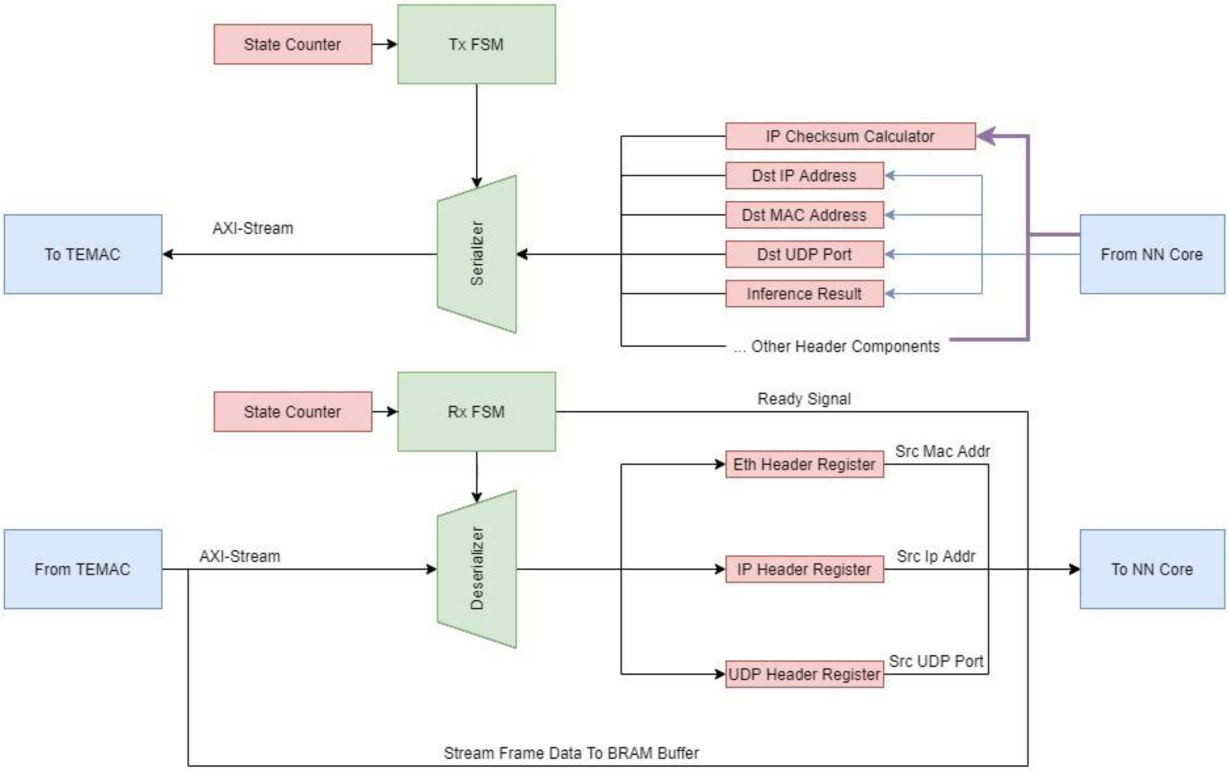


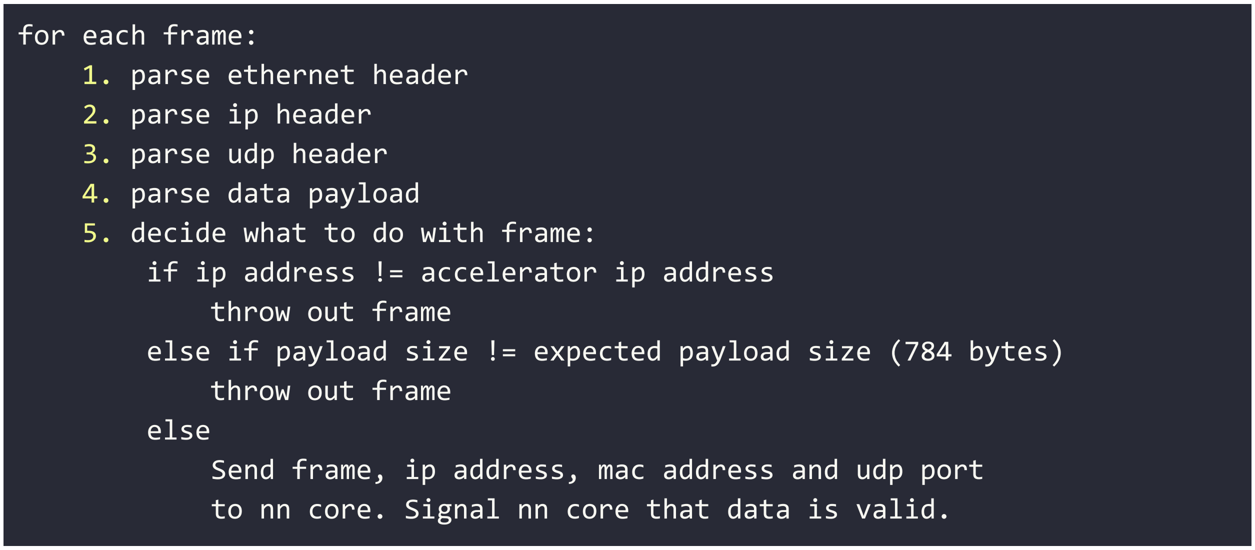
Figure 0: High Level Block Diagram of the IP/UDP layer of the networking stack

0.2.1 Integration with the TEMAC

The Tri Mode Ethernet MAC Xilinx IP (TEMAC) was used to interface with the PHY and handled a significant amount of the layer 2 networking logic. The TEMAC handled the layer 2 frame checksum (FCS) calculation for both sending and receiving of packets and was configured to throw out packets whose data did not match the FCS. The TEMACs operating speed and other settings were configured by modifying the Xillinx provided AXI-Lite TEMAC controller IP.

0.2.3 The UDP/IP Core

The UDP/IP core was responsible for parsing ethernet frames received from the TEMAC through an AXI-Stream interface. The UDP/IP module would pass frames to the NN core in accordance with the following algorithm:



```
for each frame:
    1. parse ethernet header
    2. parse ip header
    3. parse udp header
    4. parse data payload
    5. decide what to do with frame:
        if ip address != accelerator ip address
            throw out frame
        else if payload size != expected payload size (784 bytes)
            throw out frame
        else
            Send frame, ip address, mac address and udp port
            to nn core. Signal nn core that data is valid.
```

The UDP/IP core was also responsible for sending inference results to the TEMAC for transmission. This involved taking the inference result, IP address, MAC address and UDP port from the neural network core and composing it into a valid ethernet frame. This ethernet frame would be composed of an ethernet header, ip header, udp header and the inference result padded with zeros so that the total frame size was 64 bytes (the minimum ethernet frame size). The UDP/IP core also calculated a valid IP checksum for the given IP header data.

0.2.4 Integrating with the MNIST inference core

Integrating the UDP/IP core with the MNIST inference core was a collaborative effort between Andrew and I. The UDP/IP core writes the frame payload (image data) into a BRAM buffer so that the NN core can read from it. The addressing of the had to be done as a concatenation of the row number and column number of the pixel. This extra BRAM buffer also acted as a guard to prevent the NN data from being corrupted during operation.

0.3 Secondary Contributions

0.3.1 GUI Prototype

I produced a prototype GUI using python during the later half of the project. This GUI was used to prototype the load balancer and accelerator interactions with the client. James wrote a more functional and aesthetically pleasing GUI which is what we used to demo infernet.

0.3.2 Timing Closure

Andrew and I worked together to close timing on the accelerator. Due to the high congestion and heavy use of DRAM to DSP connections the design was only able to close timing using specific place and route strategies.

0.4 Tooling and Validation

0.4.1 Design Flow and Validation

My design flow was very simulation heavy, especially during the early development of the IP/UDP development module. The receiver and sending functionality of the networking stack was first simulated separately and then simulated together. Once the networking stack was simulated on it’s own, Andrew and I integrated the NN core with the networking stack and simulated that as well. Once we did as much we could in simulation we moved to testing in hardware. There is no testbench for our top level design because accurately simulating the ethernet PHY would take more time than we could afford given the duration of the project. All modifications to the top level module were done using pair programming to reduce the risk of errors. Once the board was configured in hardware we sent test images to it using a python command line tool that uses scapy to form network packets. We used a combination of ILAs and this command line tool to validate and debug the networking module in hardware.

0.5 Source Control

The RTL and testbenches for the networking module and accelerator were stored in the project's git repository. I tried to commit whenever a block was functional and validated in either simulation or hardware.

0.6 Tooling Used

The Xillinx tooling I used consisted of the following:

- Xsim
- Vivado’s hardware compiler
- Vivado’s compiler directives to close timing

Since my design flow was very standard because of my dependency on Xillinx IP. Other than Xillinx tooling I did quite a bit of debugging of the hardware using Python’s scapy and tkinter libraries.

1.0 Learning and Reflection

Overall, I think I learned a lot about how networking functions at the layer 2 and layer 3 level. I also gained more experience with hardware interface design and was able to develop a GUI in python which I have never done before.

I think the team functioned very well together mostly because all three of us have experience working with FPGAs or digital systems design in industry. Andrew and James have also successfully worked together in the past so the onus was on me to prove that I could fit within the team dynamic. Here is a list of our teams strengths and weaknesses:

Strengths	**Weaknesses**
- Communication was strong - Flexible about scheduling - Mutual trust	- Had not built a NN accelerator before

2.0 Community Contribution

I made a few piazza post in the spirit of information sharing:

Post 1: Post to board xdc files on Digilent github page
Post 2: Timing closure guide posted close to demo week.

I also helped a student on another team understand how to configure the TEMAC and modify the example design to work on the boards provided by the course.

3.0 Feedback to Xillinx

Overall I thought most of the tooling was pretty solid. My one complaint is that timing closure is irritating because seed sweeping has been removed as a feature. The vivado compiler strategies eventually closed timing for us but I felt like having the option to pick a seed for place and route would have given me more flexibility if timing was extremely marginal. This feature is still available in Quartus and perhaps I don't know the full history why this was removed in Vivado.

4.0 Course Feedback

Overall I enjoyed the course tremendously. I did come in with a background in software development of CAD tooling which I think gave me a significant advantage over a student who has not had these opportunities. Doing research in Professor Chow's lab after second year was also very helpful because concepts such as hardware hypervisors and cloud hardware were also familiar to me. I enjoyed the lectures and the open structure of the labs

If I had less experience with CAD tools, networking and heterogeneous computing I think I would have benefited from these things:

1. A list of cores that are known to work with the licencing and board setup. This list could be compiled by past projects.
2. I also found the instructional videos helpful to get familiar with Vivado's tooling. It was a very nice reference to have. Perhaps more of these videos could be made in the future for a wider range of topics.