ECE496: Implementation Plan

Integrating Wireshark with an FPGA BPF Engine (FFShark) for Packet Analysis of High Speed Networks (100G)

Project Number: 2020174

Supervisor Name: Professor Paul Chow

Administrator Name: Inci McGreal (Section 5)

Students: Kanver Bhandal, Alexander Buck, Tobias Rozario

Date of submission: November 24, 2020

1.0 Summary of Project Status and Changes

We have made progress towards integrating FFShark into Wireshark by creating and testing functional modules and subsystems. One subsystem was created to simulate network packets and send them to FFShark. Another subsystem reads filtered packets from FFShark. We have also discovered that we can use *sshdump*, a tool to connect remote machines to Wireshark, to display packets from FFShark. Currently, we can create sample test packets on the ARM machine, which is a Linux system running on the ARM CPU. We can send the packets from the ARM machine through sshdump to Wireshark, which is running on a remote workstation. Wireshark then displays the received packets. Additionally, we can send the sample packets into FFShark and read them out on the ARM machine's terminal. Now we need to combine these components by taking the filtered packets from FFShark and use the ARM machine to send them to Wireshark through sshdump. This will form the basis of our implementation.

One obstacle we are facing is a bug we found in the FFShark design. The bug corrupts packet data that is not in multiples of 8 bytes. We are currently analyzing FFShark signal waveforms and consulting with Marco Merlini, one of the designers of FFShark, to find the root cause of this issue. Since the bug only appears when data is not in multiples of 8 bytes, we can work around it by ensuring our test packets are sized in multiples of 8 bytes.

We have planned out our next set of steps for the upcoming weeks. First, we need to finish implementing the basic functionalities of our design by connecting our current working submodules. In addition, we need to create modules to enable Wireshark to send filtering instructions to FFShark. In parallel to completing functional requirements, we will continue to debug the FFShark bug. Next, we will need to create automated performance and functionality tests for our overall design. We will use performance tests to find bottlenecks within our design to improve the rate of packet transfer, while functionality tests will ensure that we continue to maintain correctness.

2.0 Possible Alternative Solution

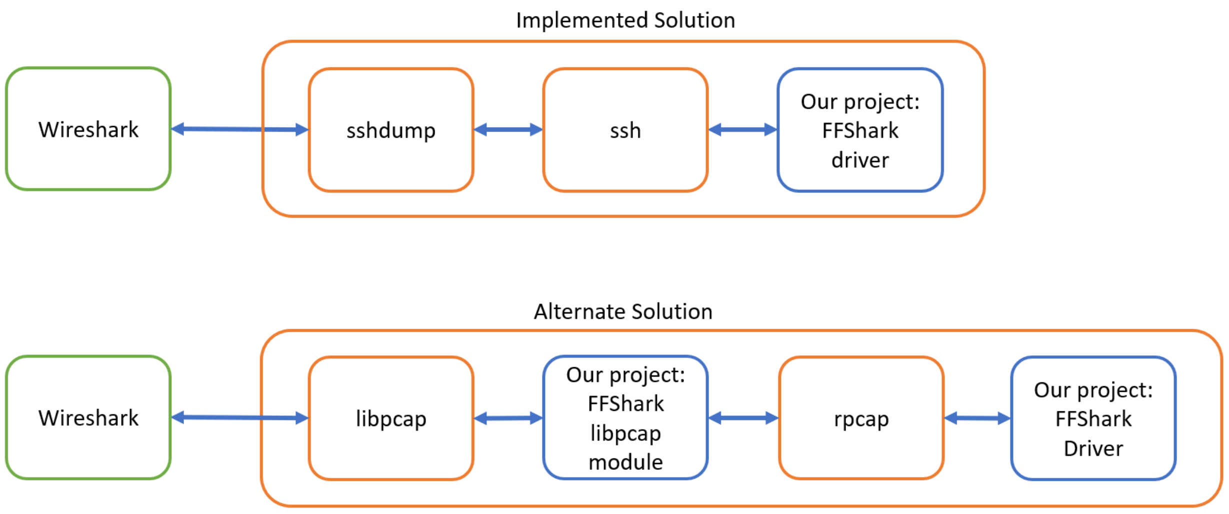

Implemented Solution
Wireshark
sshdump
ssh
Our project: FFShark driver
Alternate Solution
Wireshark
libpcap
Our project: FFShark libpcap module
rpcap
Our project: FFShark Driver

Figure 1 - Two system diagrams showcasing our alternatives. For the alternate solution an additional part of our project, an FFShark libpcap module, is required to interface with libpcap. The FFShark driver module represents our code running on the ARM machine and communicating with FFShark.

Our alternative solution was to extend *libpcap*, the library that Wireshark currently uses to interface with the BPF filter in the Linux kernel, and add support for communicating with FFShark. Figure 1 shows both our implemented and alternate solutions and the differences between the two. The key difference is that libpcap requires an additional module to be implemented to communicate between the libpcap API and our code running on the ARM machine. Libpcap provides documentation on how to add a new libpcap module to libpcap within its GitHub repository [1, 2]. The API required to be implemented by our FFShark libpcap module can be found in the Linux manual page for pcap [3].

Creating an FFShark libpcap module allows Wireshark to communicate with our FFShark Driver. First, Wireshark calls the libpcap API to interface with libpcap's internal modules. As the FFShark libpcap module obeys the libpcap API, Wireshark can communicate with the module through libpcap. Then, to establish a remote connection to the ARM machine running the FFShark Driver, *rpcap*, which is an internal libpcap module used for remote connections, can be used [4]. This allows the FFShark libpcap module to communicate with the FFShark driver. Overall, with this alternative, we would be extending libpcap, the default library Wireshark uses to get packet filtering data, and communicating with our driver to read filtered packets from FFShark.

In Table 1, we show the key differences that would result if we used the libpcap approach versus the sshdump approach.

Table 1 - Table of comparisons between the solutions

Enhancing libpcap	**Using sshdump**
Generic for use with other tools (like tcpdump)	Only works for Wireshark
API often changing, requiring maintenance [5]	sshdump interface stable since 2018 [6]
Complex interface to libpcap	Simple interface to sshdump
Uses TCP sockets implemented in rpcap for remote connection [4]	Uses ssh for remote connection [7]
Would require merging to official release code or providing users with static merged version of code	Users only require our drivers to connect FFShark to sshdump
Larger codebase	Smaller codebase

We chose to use sshdump rather than enhancing libpcap as it provided stronger benefits than the alternative. Firstly, sshdump already established a remote connection to the ARM machine. This solved one of our main problems that FFShark is not directly connected to the computer running Wireshark. Another advantage of using sshdump was that our code would add on top of the existing infrastructure. This means that when releasing code to users, they would only need to download our drivers and not modify any existing software. Had we enhanced libpcap instead, we would have either needed to merge the FFShark libpcap module to the official release or get a stale copy to merge to and provide users with that. This would make deployment more cumbersome. Additionally, from a risk perspective, it is easier to start working with sshdump as the codebase is smaller requiring less up-front learning. The interface to connect to sshdump is also simpler than the libpcap API. Therefore, it is more likely that we could get a working solution by using sshdump rather than enhancing libpcap.

While sshdump did provide us with advantages, one main drawback is that we are stuck with using ssh as our communication protocol to go from the ARM machine to Wireshark on the remote machine. Since ssh is a protocol designed to provide security, it encrypts all data. This can be a costly operation when trying to maximize transfer speed as is our aim in Objective 5. However, rather than prematurely optimizing, we wanted to get a working solution first and therefore still selected the sshdump approach.

3.0 Technical Design and Implementation

3.1 System Level Overview

Wireshark and FFShark are two existing tools for network analysis that complement one another as Wireshark has a GUI to display filtered packets and FFShark allows high speed packet filtering. Yet the two have not been connected together. Our implementation allows Wireshark running on a remote machine to communicate with FFShark on a FPGA by going through the ARM machine that is connected to FFShark. This is seen in our system block diagram in Figure 2. First, we start with Wireshark on a remote machine. Sshdump is used to establish a remote connection between the remote machine and the ARM machine. Then a user sends a PCAP filter, which consists of packet filtering instructions, using sshdump to the FFShark Driver. Inside the driver, a BPF compiler is used to translate the PCAP filter into machine code instructions. These machine code instructions are then programmed into FFShark through the SmartConnect after acquiring a lock. The lock is needed to prevent multiple threads or programs from accessing the SmartConnect block at the same time. If there was no lock, it could lead to system instability or a crash.

Next, the verification blocks simulate network packet traffic by generating and sending random packet data to FFShark through the SmartConnect FPGA fabric and the Send FIFO. FFShark then sends out filtered packets through the Receive FIFO. The FFShark Driver subsystem reads these packets from the Receive FIFO and formats them into the PCAP file format which is compatible with Wireshark. FFShark Driver sends the PCAP file to Wireshark on the remote machine through sshdump. Finally, the filtered packets are displayed on the Wireshark GUI.

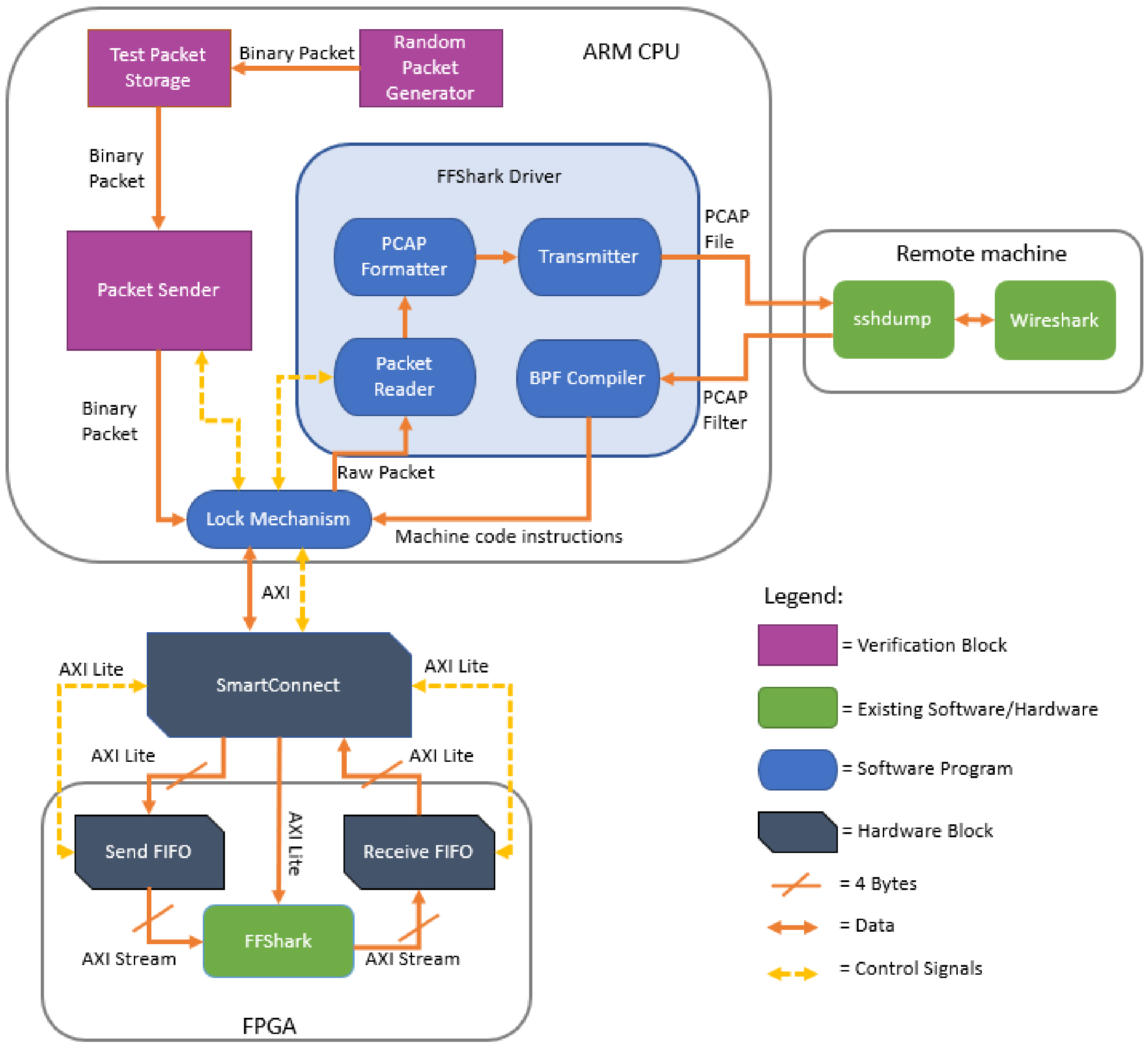


Figure 2 - A system block diagram of the design.

3.2 Module Level Descriptions

Table 2 - Verification Blocks

Block	**Inputs**	**Outputs**	**Function**
Random Packet Generator	User arguments of protocol type & size	Random packet files	Creates random packet files of varying sizes and protocols
Test Packet Storage	Random packet files	N/A	Directory to store pre-generated packet files
Packet Sender	Test Packet Storage Directory files	Packets forwarded from the test packet storage directory	Sends packet files from a given directory to FFShark

Table 3 - Software Blocks

Block	**Inputs**	**Outputs**	**Function**
FFShark Driver - Packet Reader	Raw Filtered Packets	Raw Filtered Packets	Reads the filtered packets from Receive FIFO
FFShark Driver - PCAP Formatter	Raw Filtered Packets	PCAP file	Converts packets from Packet Reader into PCAP files
FFShark Driver - Transmitter	PCAP File	PCAP file	Transmits PCAP file to sshdump
FFShark Driver - BPF Compiler	User’s PCAP filter	BPF Machine code	Translates PCAP filter into machine code for FFShark
Lock Mechanism	FPGA read/write request	Thread-safe FPGA read/write request	Allows only one thread or external tool to access the FPGA at a time

Table 4 - Hardware Blocks

Block	**Inputs**	**Outputs**	**Function**
SmartConnect	Data in AXI format	Data in AXI Lite format	Communicates between the ARM Machine and FPGA
Send FIFO	Packets in AXI Lite format	Packets in AXI Stream format	Converts and buffers unfiltered packets from ARM Machine and sends them to FFShark
Receive FIFO	Filtered packets in AXI Stream format	Filtered packets in AXI Lite format	Converts and buffers filtered packets from FFShark and sends them to the ARM machine

Table 5 - Existing Blocks

Block	**Inputs**	**Outputs**	**Function**
FFShark	- Input packet stream - BPF Filter	Filtered packets	Filters network packets according to the filter
sshdump	- PCAP file - User’s PCAP filter	- PCAP file - User’s PCAP filter forwarded	- Sends packets in PCAP file to Wireshark - Sends the filter to FFShark Driver
Wireshark	- PCAP file - User’s PCAP filters	- Displays packets on GUI - User’s PCAP filters	- Displays filtered packets to the user on Wireshark GUI - Allows users to enter filters that are forwarded to sshdump

4.0 Test and Verification

4.1 Current Testing

We verified the functionality of our currently implemented blocks manually during development. For testing that we can send and receive packets from FFShark, we sent in sample packets and manually verified that the received packets matched. To verify the sshdump connection, we created a script to send sample packets in PCAP format to sshdump and read those in Wireshark. We then manually compared the packets to ensure they matched. To test the lock mechanism, we added a delay to the read and write requests going through the lock mechanism block. This forces the first request to hold the lock longer allowing us to verify that the second request only executed once the first finished.

Table 6 in Section 4.2 showcases how we will verify that our functionality, constraints, and objectives have been met.

4.2 Verification Table

Table 6 - Verification Table

ID	Project Requirement	Verification Method
F1, F2	Display filtered packets from FFShark on Wireshark and send filters to FFShark via Wireshark	**TEST:** Automated script to compare packets displayed on Wireshark to packets sent to FFShark. Script will also make sure expected filtered packets are displayed
C1	Must use Wireshark packet analyzer to display FFShark	**REVIEW of DESIGN:** Wireshark is used
C2	Must run Wireshark on an external workstation	**REVIEW of DESIGN:** Wireshark does not run on the ARM machine
C3	Must use the Ethernet port from the ARM machine to connect to the internet	**REVIEW of DESIGN:** Ethernet is the only connection from the ARM machine to the internet
C4	Must not utilize the *MUL* (multiplication), *DIV* (division), and *MOD* (modulo) filtering instructions	**TEST:** Send filters that use these instructions and report an error to the user
O1, O2, and O3	Limit number of user clicks in Wireshark main menu to 2, number of scripts to run per session to 3, and number of initial setup scripts to 2	**REVIEW of DESIGN:** Manually count and check that each objective is met
O4	Remove the need for users to access the Linux machine running on the ARM machine	**REVIEW of DESIGN:** Manually verify no scripts or programs must be user initiated on the ARM machine
O5	Support 1G Rate of captured filters	**TEST:** Measure and calculate the rate of packets displayed on Wireshark using Wireshark timestamps and packet byte sizes
O6	Support minimum of 2 operating systems	**REVIEW of DESIGN:** Wireshark can connect to FFShark on 2 of the following operating systems: Linux, Windows, or MacOS

5.0 References

[1] Wireshark, “Chapter 8. Packet Capture,” [Online]. Accessed 24-Nov-2020. https://www.wireshark.org/docs/wsdg_html_chunked/ChapterCapture.html.

[2] The-Tcpdump-Group, “the-tcpdump-group/libpcap,” *GitHub*. [Online]. Accessed: 24-Nov-2020. https://github.com/the-tcpdump-group/libpcap/blob/master/doc/README.capture-module.

[3] *Man page of PCAP*. [Online]. Accessed: 24-Nov-2020. https://www.tcpdump.org/manpages/pcap.3pcap.html.

[4] S. Krishnan, “README for rpcap,” *Main*. [Online]. Accessed: 24-Nov-2020. http://rpcap.sourceforge.net/README.html.

[5] Linux Kernel, “libcap/libcap.git”, [Online]. Accessed November 22, 2020. https://git.kernel.org/pub/scm/libs/libcap/libcap.git/log/

[6] Wireshark, “sshdump.c commit history”, [Online]. Accessed November 22, 2020. https://github.com/wireshark/wireshark/commits/master/extcap/sshdump.c

[7] D. Lombardo, “sshdump,” *sshdump - The Wireshark Network Analyzer 3.2.7*. [Online]. Accessed September 21, 2020. Available: https://www.wireshark.org/docs/man-pages/sshdump.html.

[8] C. Vega, M. Merlini and P. Chow, “FFShark: A 100G FPGA Implementation of BPF Filtering for Wireshark,” in 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). [Online]. IEEE, 2020, Accessed September 21, 2020. https://www.fccm.org/past/2020/proceedings/2020/pdfs/FCCM2020-65FOvhMqzyMYm99lfeVKyl/580300a047/580300a047.pdf.

[9] Wireshark, “Wireshark User’s Guide Version 3.3.1,” 2020, [Online]. Accessed September 21, 2020. https://www.wireshark.org/docs/wsug_html_chunked/.

[10] A. Cardigliano, “PF-RING - Wireshark Extcap”, 2020, [Online]. Accessed September 21, 2020.
https://github.com/ntop/PF_RING/tree/dev/userland/wireshark/extcap.

[11] N. Bowden, “WLAN-Pi - WLANPiShark2”, 2019, [Online]. Accessed September 21, 2020. https://github.com/WLAN-Pi/WLANPiShark2.

[12] GitHub, “Pricing”, [Online]. Accessed November 22, 2020.
https://github.com/pricing

[13] Start, “Internet Packages”, [Online]. Accessed November 22, 2020.
https://www.start.ca/services/high-speed-internet#packages

[14] Xilinx, “Vivado Design Suite - HLx Editions”, [Online]. Accessed November 22, 2020.
https://www.xilinx.com/products/design-tools/vivado.html#buy

[15] Best Buy, “ASUS ZenBook 14" Laptop”, [Online]. Accessed November 22, 2020.
https://www.bestbuy.ca/en-ca/product/asus-zenbook-14-laptop-dark-royal-blue-intel-core-i7-10510u-512gb-ssd-16gb-ram-windows-10-en/14470973

Appendices

Appendix A: Requirements Specification

Project requirements have been divided into three categories: functions, constraints, and objectives. Functions, shown in Table 7, allow the design to meet the basic needs of industry professionals to debug network connections. Constraints, shown in Table 8, are restrictions the solution must follow. Lastly, objectives, shown in Table 9, are additional targets, and ways of determining the effectiveness of the design.

Table 7: Functions defining the basic criteria of the design

ID	**Function**	**Explanation**
F1	Display captured filtered packets from FFShark on Wireshark	Wireshark needs to be used to display filtered packets to the user.
F2	Send filters to FFShark via Wireshark	The design needs to allow Wireshark to send filters to FFShark.

Table 8: Constraints the design must follow

ID	**Constraint**	**Explanation**
C1	Must use Wireshark packet analyzer to display FFShark	Client requires Wireshark to be used.
C2	Must run Wireshark on an external workstation	The ARM machine does not have an environment for GUIs. Thus, Wireshark cannot be run directly on the ARM machine.
C3	Must use the Ethernet port from the ARM machine to connect to the internet	There are a limited number of ports on the FPGA. The current physical setup for the FFShark design has the Ethernet port, connected to the ARM machine, as the only medium for FFShark to send over filtered packets to an external interface [8].
C4	Must not use multiplication, division, and modulo filtering instructions	FFShark omitted the instructions to optimize performance since they are rarely used in filtering applications [8].

Table 9: Objectives defining the goals of the design

ID	**Objective**	**Explanation**
O1	Limit the number of clicks from the Wireshark main menu to view FFShark captured filters (at most 2 clicks)	Two clicks were chosen because users perform 1 click on the GUI to select the capture interface (Wi-Fi, Ethernet, LAN etc.) [9]. 1 extra click should be required at most to see FFShark captured data.
O2	Limit the number of extra scripts to run per session when running our design with Wireshark (at most 3)	A maximum of 3 scripts was chosen since many open-source designs interfacing Wireshark to custom hardware have 2-4 [10], [11].
O3	Limit the number of scripts to run or plugins to install during the initial setup (2 at most)	Initial setup is the installation steps performed when running our design with Wireshark for the first time. Many custom open-source Wireshark based projects require only 1-2 plugins/scripts to be installed/run [10], [11].
O4	Remove the need for users to access the Linux machine running on the ARM machine	Currently, users need to remotely access the Linux machine running on the ARM machine to program the FFShark bitstream onto the FPGA. The goal is to provide an abstraction such that users do not need to be aware of these steps.
O5	Support 1G rate of captured filters	FFShark has a 1G Ethernet port connected to the ARM machine [8]**.** The goal is to use the entire bandwidth of the port to send filtered data.
O6	Support several operating systems (minimum of 2).	Common operating systems are Windows, MacOS and Linus. The goal is to support at least 2 of these operating systems to support more users.

Appendix B: Risk assessment

Two key risks to the project were discovered during the initial stages of implementation. One is a bug found in the FFShark design, and another is an implementation strategy that possibly reduces performance.

We found a bug in FFShark where it forwards corrupt packet data when given network packets that are not multiples of 8 bytes. To solve this issue, we plan on setting up tests for packets that are multiples of 8 bytes and packets which are not. We will compare FFShark signal waveforms from both tests to narrow down the cause. In addition, we are also getting debugging assistance from Marco Merlini, one of the designers of FFShark. If we run out of time on solving this issue, we will limit the scope of our solution by having it only accept packets that are multiples of 8 bytes. We will still be able to connect FFShark to Wireshark, which is the overall project goal, but be limited in our solution's functionality.

The current implementation strategy has a risk of reducing performance because it needs to perform arbitration between the two subsystems that access the FPGA running FFShark. The verification blocks need to send packets to FFShark, while the FFShark Driver blocks need to read filtered packets from FFShark. If both subsystems interact with FFShark simultaneously, data corruption or crashing may occur. To prevent this, a module will simply lock the FPGA resource when using it. If the resource has already been locked by the other module, the module attempting to gain access will simply wait until the FPGA has been unlocked. Modules having to wait on each other could bottleneck performance. If this form of arbitration is too slow, we will use more complex resource access synchronization strategies. We are capable of implementing a solution using concepts from operating systems as all of us have taken courses in this area. If we are unable to solve the potential bottleneck within the project time limit, we will solely focus on meeting our basic functional requirements of connecting FFShark to Wireshark.

Appendix C: Gantt Chart

In Figure 3, the Gantt chart for our project is shown. Currently we are on track for completing our project as we have completed or are working on all tasks that are meant to be done before November 23. There is a 5-week period labeled "Winter Break & Exam Period" where we will not be working as much on the project due to finals or spending time with family.

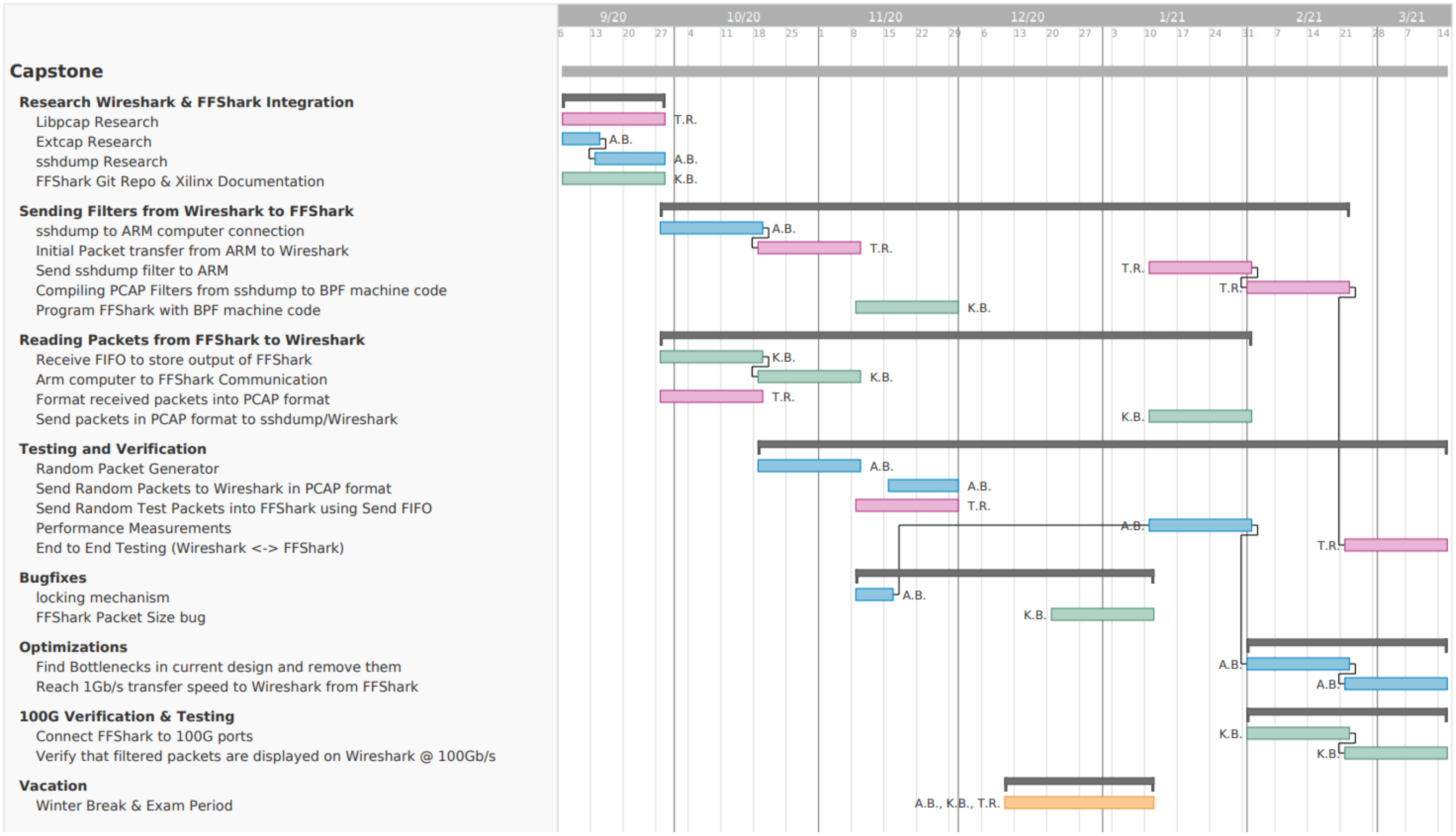
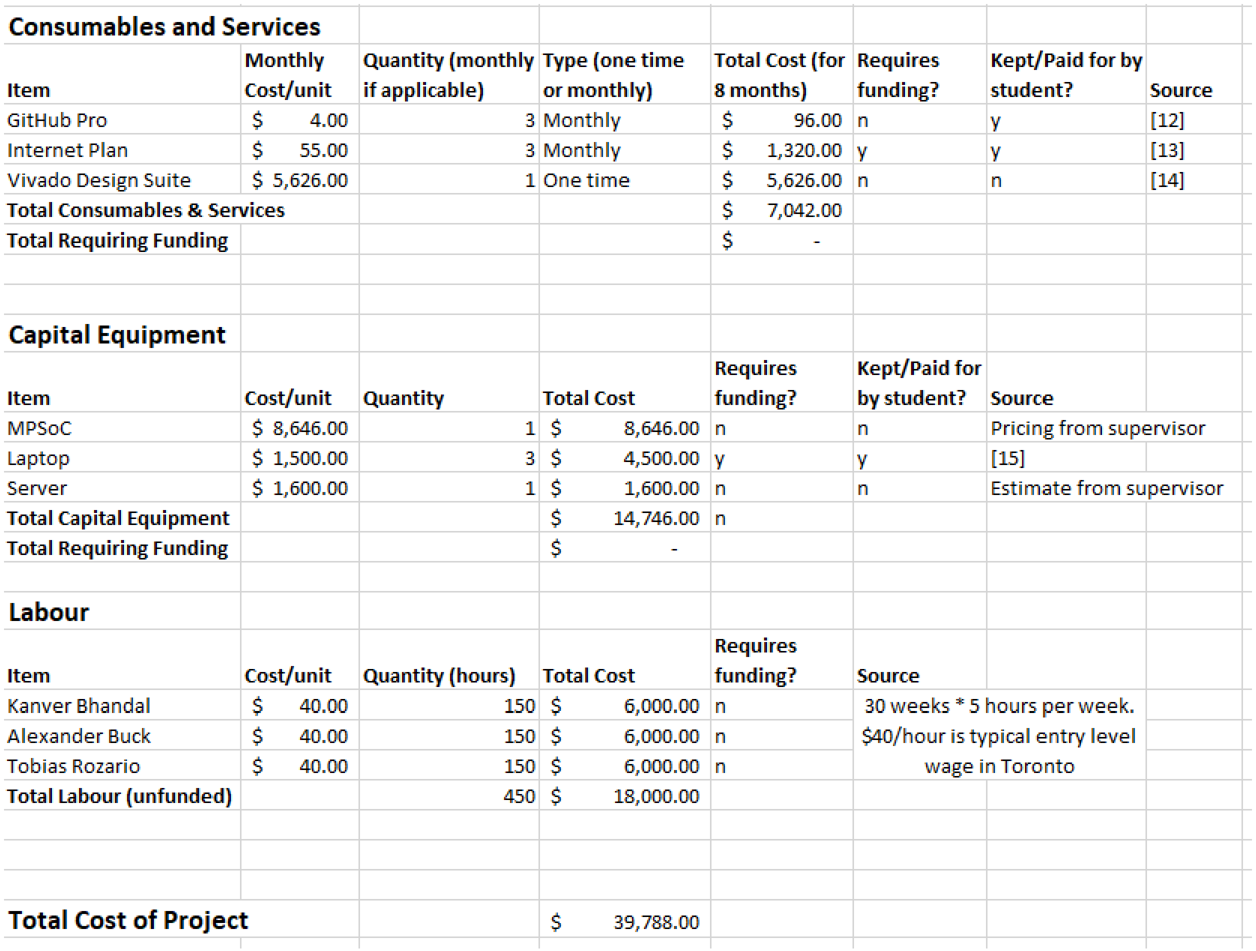


Figure 3 - Gantt Chart

Appendix D: Financial Plan

The total cost of our project is $39,788 and is detailed in Table 10. However, our project requires no funding since our supervisor already has the necessary components for us to use. These components are the Vivado Design Suite, MPSoC, and server. The other items, namely the internet plan and laptop, we already own. The GitHub license is free for us as students. All prices are listed in CAD. Applicable items, namely the Vivado Design Suite and MPSoC, were converted from USD at a rate of $1.31.

Table 10 - The budget of our project.



Consumables and Services							
Item	**Monthly Cost/unit**	**Quantity (monthly if applicable)**	**Type (one time or monthly)**	**Total Cost (for 8 months)**	**Requires funding?**	**Kept/Paid for by student?**	**Source**
GitHub Pro	$ 4.00	3	Monthly	$ 96.00	n	y	[12]
Internet Plan	$ 55.00	3	Monthly	$ 1,320.00	y	y	[13]
Vivado Design Suite	$ 5,626.00	1	One time	$ 5,626.00	n	n	[14]
Total Consumables & Services				$ 7,042.00			
Total Requiring Funding				$ -			
Capital Equipment							
Item	**Cost/unit**	**Quantity**	**Total Cost**	**Requires funding?**	**Kept/Paid for by student?**	**Source**	
MPSoC	$ 8,646.00	1	$ 8,646.00	n	n	Pricing from supervisor	
Laptop	$ 1,500.00	3	$ 4,500.00	y	y	[15]	
Server	$ 1,600.00	1	$ 1,600.00	n	n	Estimate from supervisor	
Total Capital Equipment			$ 14,746.00	n			
Total Requiring Funding			$ -				
Labour							
Item	**Cost/unit**	**Quantity (hours)**	**Total Cost**	**Requires funding?**	**Source**		
Kanver Bhandal	$ 40.00	150	$ 6,000.00	n	30 weeks * 5 hours per week.		
Alexander Buck	$ 40.00	150	$ 6,000.00	n	$40/hour is typical entry level		
Tobias Rozario	$ 40.00	150	$ 6,000.00	n	wage in Toronto		
Total Labour (unfunded)		450	$ 18,000.00				
Total Cost of Project			$ 39,788.00				