WES Cohort 9 Capstone
WES269 Final Progress Report
V2X Motorcycle HUD

Team Members
Ryan Hiser
Jorge Pacheco
Jacob Nguyen

Table of contents

Introduction **2**
- Use Case 2
- Materials (Apparatus) 2
- System Diagram 4

Waveform and System Description **5**
- Data Packet 5
- System Specifications 5

Schedule **6**

Results **6**
- Simulink Models 6
    - TX 6
    - RX 7
- HUD 8
- System Setup (FMComms4 and Zedboard Integration) 9
- FPGA Implementation 10

Conclusion **11**

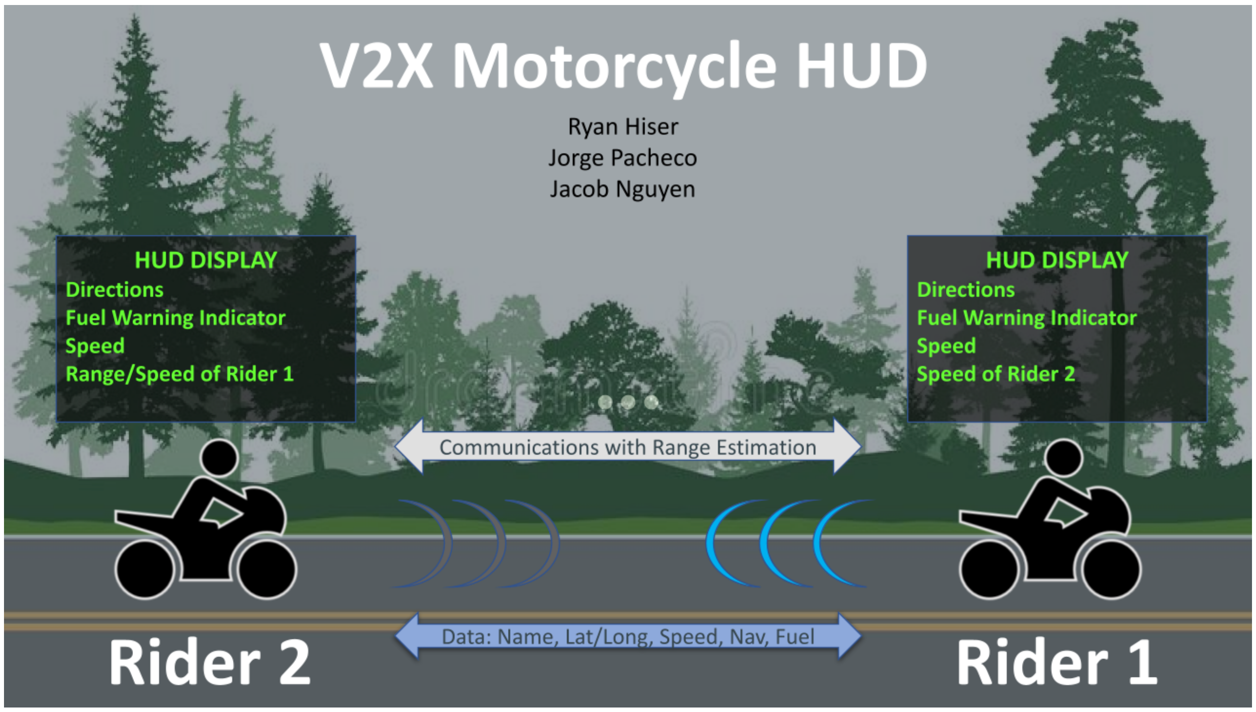
Citations **11**

Appendix **12**
- FMComms Loop Back Test 12

Introduction

Our team seeks to create a V2X (vehicle-to-everything) system using two motorcycles, in which each vehicle can speak with each other through an audio channel and share its driving status. Each driver shall see this shared information through their helmet mounted head-up display (HUD).This information includes the other vehicle's name, navigation directions, speed, and geographic location.

Use Case



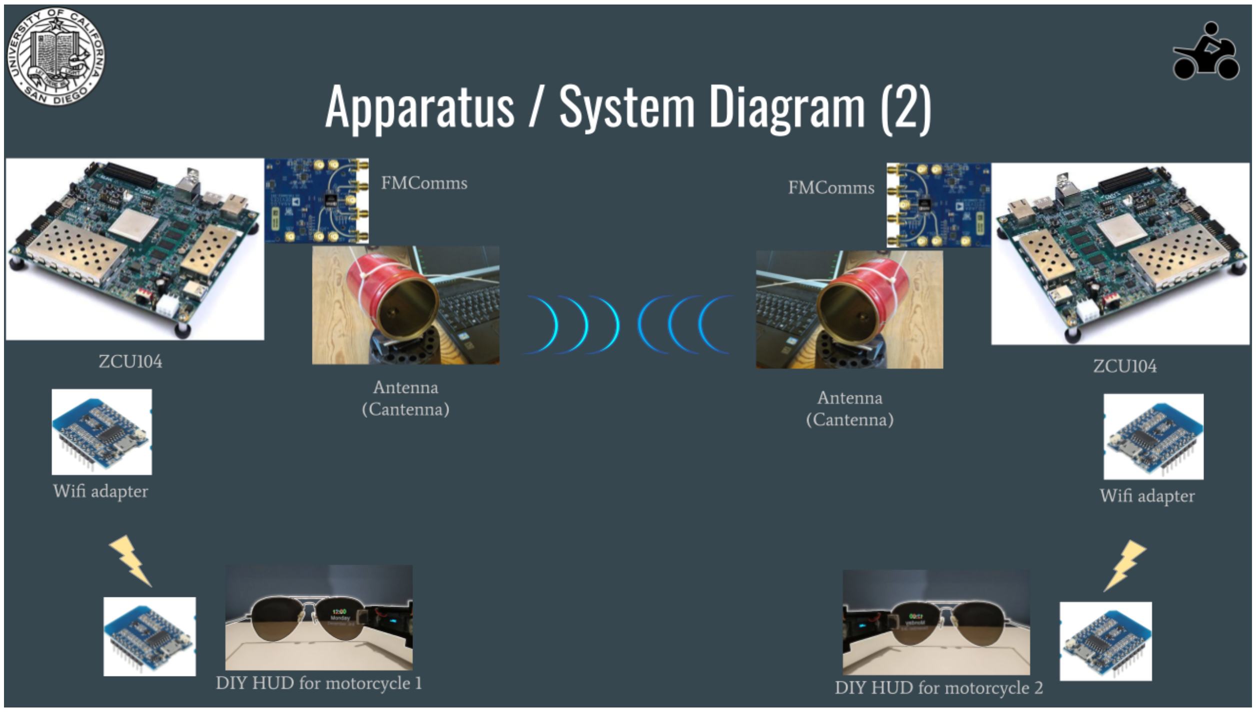
The scenario above shows two riders communicating information with each other. Both riders will mount a transceiver (XCVR), and they will communicate using a Time Division Multiple Access (TDMA) protocol. In order to sync the devices and assign time slots, the riders will need to synchronize their clocks at the start of operations. Distance calculations will be determined using the time delay between a message being transmitted and received. Each transceiver is responsible for capturing and recording both its local status information and the received status information. This information will be displayed on each rider's HUD.

Materials (Apparatus)

The materials listed below are all the chosen components for a singular Rider V2X system.

Item	Qty	Price	Notes
Zedboard	3	$0	Borrowed
FMComms4	3	$0	Borrowed
HUD			
OLED - SSD1306	3	$21.99	In Hand (x3)
Battery (LIPO) - LP542730	3	$0	In Hand (x3)
Battery Charger - ADA1904	1	$12.28 ea	Already Have (x1)
Google Cardboard	2	$9.99 ea	Already Have (x3)
Wifi/Microprocessor - ESP8266	1	$16.99	Already Have (x5)
GPS Unit	3	$29.95 ea	Adafruit Ultimate GPS Breakout – 66 Channel w/10 Hz updates -Version 3 (or we will use a preloaded LUT to simulate)

System Diagram



The V2X diagram above shows the intended communications data path. Each V2X system has 3 essential components to be designed: Zedboard Software(SW) for HUD/FPGA interface, Zedboard FPGA firmware, and HUD design (SW/HW interface).

Zedboard SW:

- Track local status
- Track received status
- Display HUD info
- Prepare transmit data
- Decode received data
- Determine distance between systems

Zedboard FPGA firmware:

- Modulate transmit data (from SW)
- Demodulate receive data (to SW)

HUD design:

- Display data in a digestible format
- Display info without hindering view

Waveform and System Description

This section describes the currently planned/implemented design for the V2X system. The current design has the following constraints:

- Maximum HUD refresh rate: 2Hz
- Maximum detection distance: 0.25mi (0.4km)
- Maximum vehicle speed: 100MPH (160km/h)
- Minimum navigation distance resolution: 0.5mi

Data Packet

The transmitted data packet is composed of an info packet that contains the HUD information and an audio packet.

Data Packet: 7200 bits (0.01s of audio)

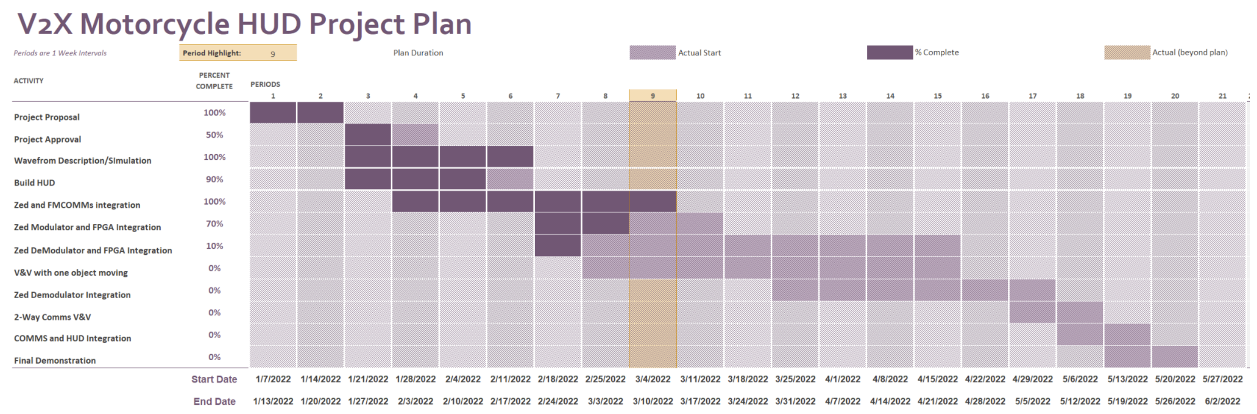
- Info Packet: 144 bits
    - Identifier (name): 32 bits (4 ASCII Characters)
    - Position (lat/long): 64 bits (float, float)
    - Speed (mph): 8 bits (255 max) mph
    - Navigation: 40 bits
        - Directions: 8 bits (left, right, straight, u-turn, arrived)
        - Distance to next step: 32 bits (float)
- Audio Packet: 7056 bits
    - 1 channel (16 bits per sample for each channel)
    - WAV format (44.1kHz)
    - 0.01 seconds of audio

System Specifications

The communications specifications for the V2X system are the following:

- Throughput: 720 kbps
- Minimum Required Sampling Frequency: 13.44 Msps
- Carrier Frequency: 2.4GHz (WiFi band)
- Maximum Doppler Frequency: 2,400,715 Hz
    - Maximum Offset Frequency: 715 Hz
- Maximum Frequency Error: 781.25 KHz

Schedule



V2X Motorcycle HUD Project Plan

Periods are 1 Week Intervals Period Highlight: 9 Plan Duration ▨ Actual Start ■ % Complete ▨ Actual (beyond plan)

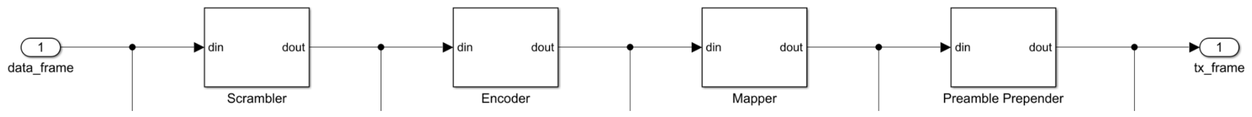
ACTIVITY	PERCENT COMPLETE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Project Proposal	100%	■	■																			
Project Approval	50%			■	▨																	
Wavefrom Description/SImulation	100%			■	■	■	■															
Build HUD	90%			■	■	■	▨															
Zed and FMCOMMs integration	100%				■	■	■	■	■	■												
Zed Modulator and FPGA Integration	70%							■	■	▨	▨											
Zed DeModulator and FPGA Integration	10%							■	▨	▨	▨	▨	▨	▨	▨	▨						
V&V with one object moving	0%								▨	▨	▨	▨	▨	▨	▨	▨						
Zed Demodulator Integration	0%												▨	▨	▨	▨	▨	▨				
2-Way Comms V&V	0%																	▨	▨			
COMMS and HUD Integration	0%																		▨	▨		
Final Demonstration	0%																			▨	▨	
	Start Date	1/7/2022	1/14/2022	1/21/2022	1/28/2022	2/4/2022	2/11/2022	2/18/2022	2/25/2022	3/4/2022	3/11/2022	3/18/2022	3/25/2022	4/1/2022	4/8/2022	4/15/2022	4/22/2022	4/29/2022	5/6/2022	5/13/2022	5/20/2022	5/27/2022
	End Date	1/13/2022	1/20/2022	1/27/2022	2/3/2022	2/10/2022	2/17/2022	2/24/2022	3/3/2022	3/10/2022	3/17/2022	3/24/2022	3/31/2022	4/7/2022	4/14/2022	4/21/2022	4/28/2022	5/5/2022	5/12/2022	5/19/2022	5/26/2022	6/2/2022

Results

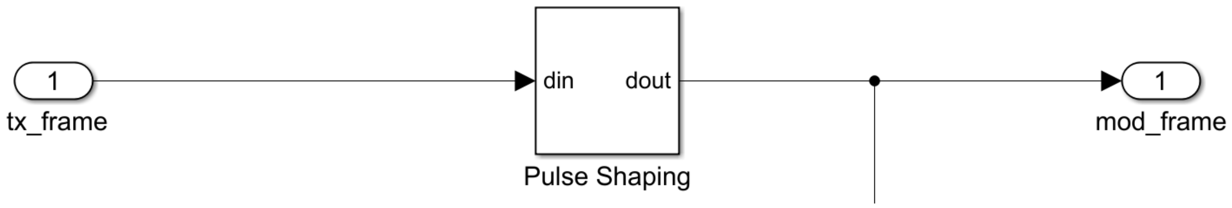
Simulink Models

The images and descriptions below show the current Simulink models for the V2X system. The Simulink models for V2X TX Baseband, TX Modulator, RX Baseband, are able to generate embedded C code, and output the same vectors as our Simulink models.

TX:

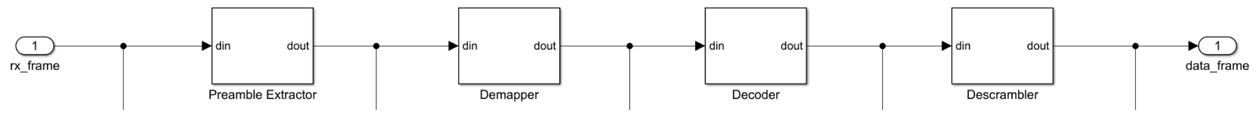


- TX Baseband
  - Scrambler
  - Encoder (Reed-Solomon (7, 3, 3))
  - Mapper (QPSK)
  - Preamble Prepender (2x sequences, N = 6 (64 bits))

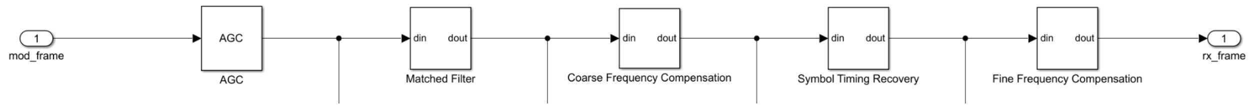


- TX Modulator
  - Pulse Shaping (SRRC at 8 sps)

RX:



- RX Demodulator
  - Automatic Gain Control (AGC)
  - Matched Filter (SRRC at 8 sps)
  - Coarse Frequency Compensation
  - Symbol Timing Recovery
  - Fine Frequency Compensation



- RX Baseband
  - Preamble Extractor
  - Demapper (QPSK)
  - Decoder (Reed-Solomon (7, 3, 3))
  - Descrambler

HUD

The Head-up display (HUD) is mainly composed of the following items:

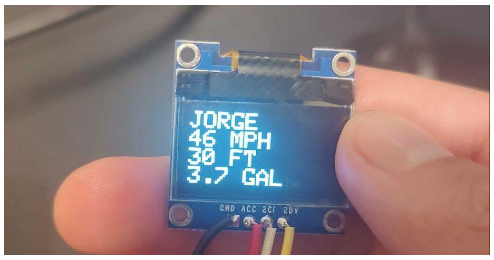
- ESP8266 WiFi hotspot and server module
- SSD1306 0.96 inch Organic light-emitting diode (OLED) display
- 3.7 Volt LiPo battery
- Jumper wires
- Other miscellaneous components

We got our main inspiration for our HUD development from the following do-it-yourself (DIY) tutorial:

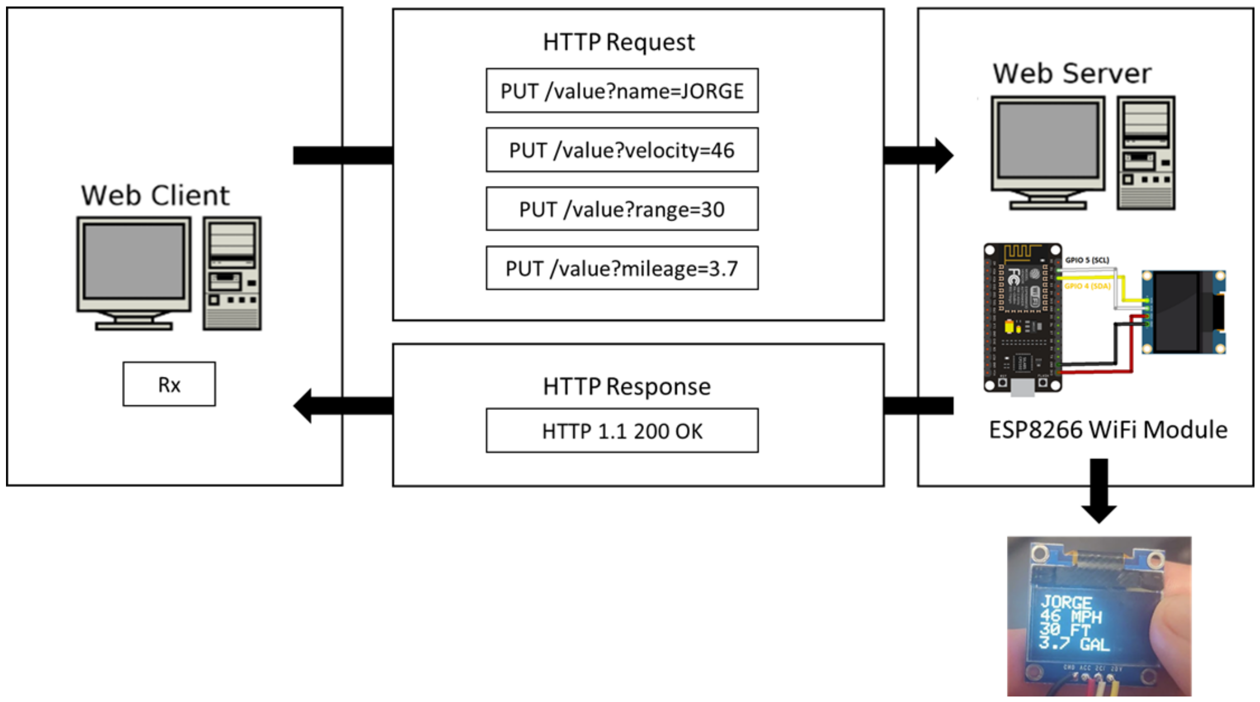
- https://www.instructables.com/DIY-Smart-Glasses-ArduinoESP/

Soon after we got our hands in the HUD equipment, we began the development of our HUD by simply following the instructions from the DIY tutorial. The team ran into countless issues trying to set up the ESP8266 module to fully control the OLED screen, using both SPI and I2C communication buses. It became apparent that the OLED screens that we originally ordered were not following the specifications of the SSD1306 controller. Instead of exhausting more time, the team decided to order a new set of equipment and start over with the HUD development.

After we received the second batch of equipment, we were able to control and configure the ESP8266 + SSD1306 combination without any major issues. The team devoted some time to investigate how to fully command the OLED display and learn how to control text positioning, text sizing, screen refresh, and other important command and control configurations. Then, we focused our attention on configuring the ESP8266 as its own WiFi hotspot in order to have clients able to control the text being displayed on the OLED screen. Then, we configured the ESP8266 to be both a WiFi hotspot and a simple Hypertext Transfer Protocol (HTTP) server so that clients could update individual metrics to be displayed in the HUD, as well as controlling what to display on the screen. The following picture is an example of how the text will look in the OLED screen:



The following diagram further describes the final setup for the ESP8266 and the SSD1306 OLED screen:



The next steps for HUD development is to insert a HTTP client after we have demodulated and received our payload in the Rx. Our plan is to create a client that can both parse individual fields from our payload and send HTTP requests to our server to update individual metrics in our HUD. Once that is done, we plan to work on the enclosure and adapt our HUD setup so that the information is clearly visible, non-obstructive, and can be fully mounted in a motorcycle helmet.

System Setup (FMComms4 and Zedboard Integration)

The base hardware for our system consists of the Zedboard from Digilent and the FMComms4 from Analog Devices. First we created a virtual machine with the following software versions:

- Ubuntu 18.04
- Vivado and Vivado HLS 2018.3
- Petalinux 2018.3

With the required software included we used the Analog Devices HDL git repository for the FMComms2 to generate the base FPGA project. WIth a working bitstream for the FPGA we began the petalinux build. Analog Devices provide the Yocto layer meta-adi, which is used in combination with Xilinx Petalinux build tools to generate the required boot files (boot.bin, and image.ub) with the appropriate hooks to the Fmcomms4. Unfortunately, FMComms4 is not natively supported by the Meta-adi layer. This required some modification to the layer definition (bbappend file and device trees). The Zedboard boots from a secure digital (SD) card. The SD card is partitioned into two. One partition stores the boot image, while the other partition holds the root file system (rootfs). For our system we desired to use the PYNQ root file system. This was the preferred choice due to the team's familiarity with the PYNQ-Z2 development board.

The following software versions/repository tags were used:

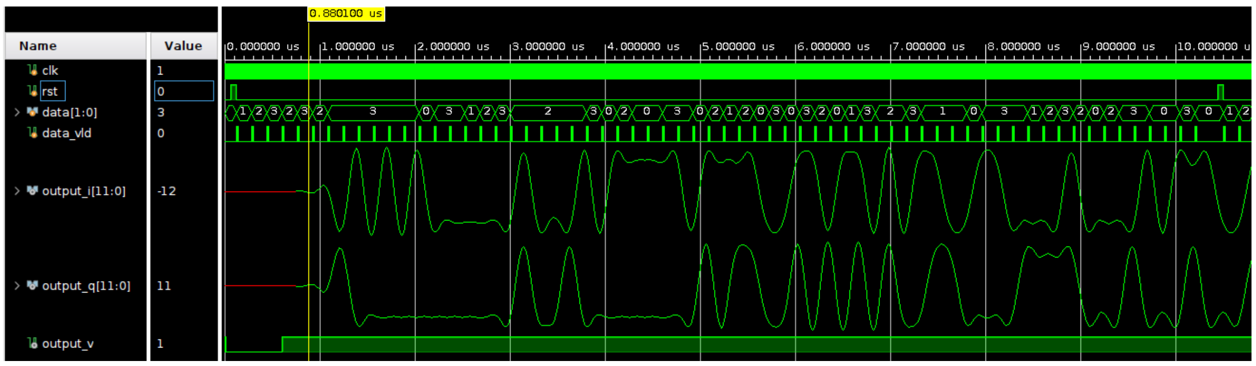
- PYNQ 2.4
- Meta-adi 2019_R1
- HDL 2019_R1
- LibIIO

A big accomplishment for this project was the successful creation of the boot images using the meta-adi layer and the PYNQ image. With a working image we were able to install the libIIO software library from Analog Devices. We were then able to read and write to registers from the FMComms4 and stream data. A simple example of reading from the FMComms4 can be found in the appendix; however for further testing, we need an attenuator to avoid damage to the FMComms4.

FPGA Implementation

We began investigating the FPGA implementation of the demodulator and modulator. For the demodulation, we implemented a polyphase filter using Vivado HLS. This will be used for the Timing Error Correction. The Polyphase filter is used to interpolate between samples allowing for fine timing correction [1].

While most of the modulation (scrambling, data creation, etc.) is planned for the ARM processor, We implemented a mapper and shaping filter using verilog. Additionally, a testbench was written to test the output of the mapper and shaping filter.



Conclusion

As to be expected, some delays were encountered due to overall system complexity. Our team worked effectively to address issues such as the unsupported software versions, lack of meta-adi support, and a defective OLED. To accomplish our goals we routinely reached out for help and continued to make progress in other areas when delays occurred.

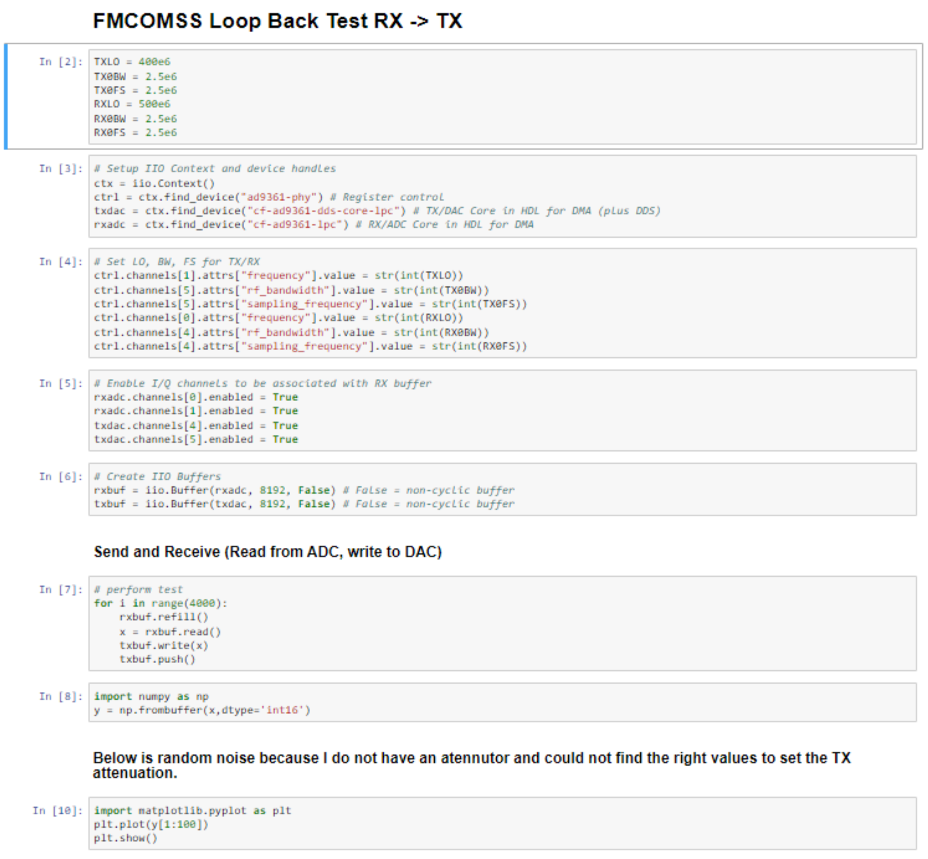
Our project is well underway with significant progress made during Winter Quarter. We were able to successfully create a Simulink Model for our QPSK waveform, implement a HTTP server/client with an OLED, and integrate the FMComms4 with the Zed board. We also were able to create our waveform definition and select a method of timing error correction. We are well positioned to complete the modulator over the break, and begin the FPGA/ARM integration of the Demodulator at the start of next quarter.

Citations

1. Harris F. (2011) Let’s Assume the System Is Synchronized. In: Prasad R., Dixit S., van Nee R., Ojanpera T. (eds) Globalization of Mobile and Wireless Communications. Signals and Communication Technology. Springer, Dordrecht. https://doi.org/10.1007/978-94-007-0107-6_20

Appendix

FMComms Loop Back Test



FMCOMSS Loop Back Test RX -> TX

```
In [2]: TXLO = 400e6
        TX0BW = 2.5e6
        TX0FS = 2.5e6
        RXLO = 500e6
        RX0BW = 2.5e6
        RX0FS = 2.5e6
```

```
In [3]: # Setup IIO Context and device handles
        ctx = iio.Context()
        ctrl = ctx.find_device("ad9361-phy") # Register control
        txdac = ctx.find_device("cf-ad9361-dds-core-lpc") # TX/DAC Core in HDL for DMA (plus DDS)
        rxadc = ctx.find_device("cf-ad9361-lpc") # RX/ADC Core in HDL for DMA
```

```
In [4]: # Set LO, BW, FS for TX/RX
        ctrl.channels[1].attrs["frequency"].value = str(int(TXLO))
        ctrl.channels[5].attrs["rf_bandwidth"].value = str(int(TX0BW))
        ctrl.channels[5].attrs["sampling_frequency"].value = str(int(TX0FS))
        ctrl.channels[0].attrs["frequency"].value = str(int(RXLO))
        ctrl.channels[4].attrs["rf_bandwidth"].value = str(int(RX0BW))
        ctrl.channels[4].attrs["sampling_frequency"].value = str(int(RX0FS))
```

```
In [5]: # Enable I/Q channels to be associated with RX buffer
        rxadc.channels[0].enabled = True
        rxadc.channels[1].enabled = True
        txdac.channels[4].enabled = True
        txdac.channels[5].enabled = True
```

```
In [6]: # Create IIO Buffers
        rxbuf = iio.Buffer(rxadc, 8192, False) # False = non-cyclic buffer
        txbuf = iio.Buffer(txdac, 8192, False) # False = non-cyclic buffer
```

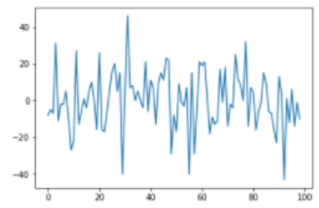
Send and Receive (Read from ADC, write to DAC)

```
In [7]: # perform test
        for i in range(4000):
            rxbuf.refill()
            x = rxbuf.read()
            txbuf.write(x)
            txbuf.push()
```

```
In [8]: import numpy as np
        y = np.frombuffer(x,dtype='int16')
```

Below is random noise because I do not have an atennutor and could not find the right values to set the TX attenuation.

```
In [10]: import matplotlib.pyplot as plt
         plt.plot(y[1:100])
         plt.show()
```


40
20
0
−20
−40
0
20
40
60
80
100