



V2X Motorcycle HUD Weekly Updates

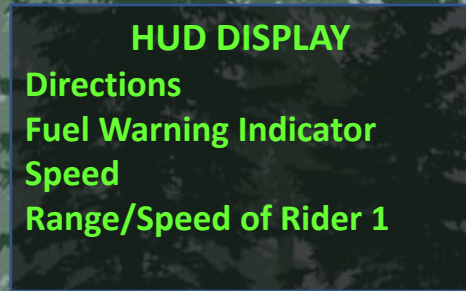
...

Jacob Nguyen, Ryan Hiser, Jorge Pacheco
UCSD MAS WES
16 April 2022

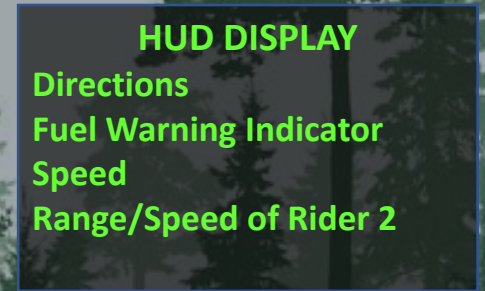
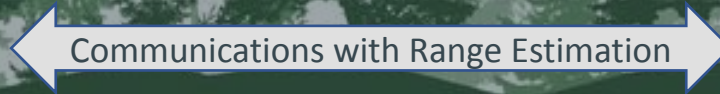


Introduction

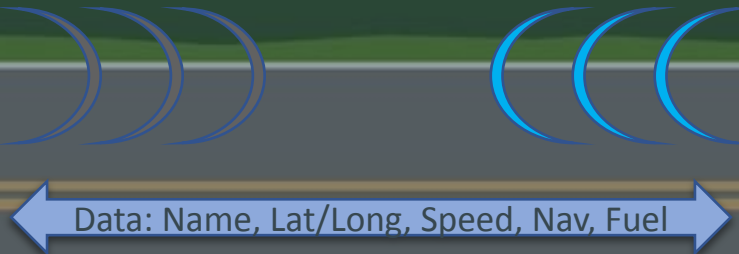
V2X Motorcycle HUD



Rider 2



Rider 1



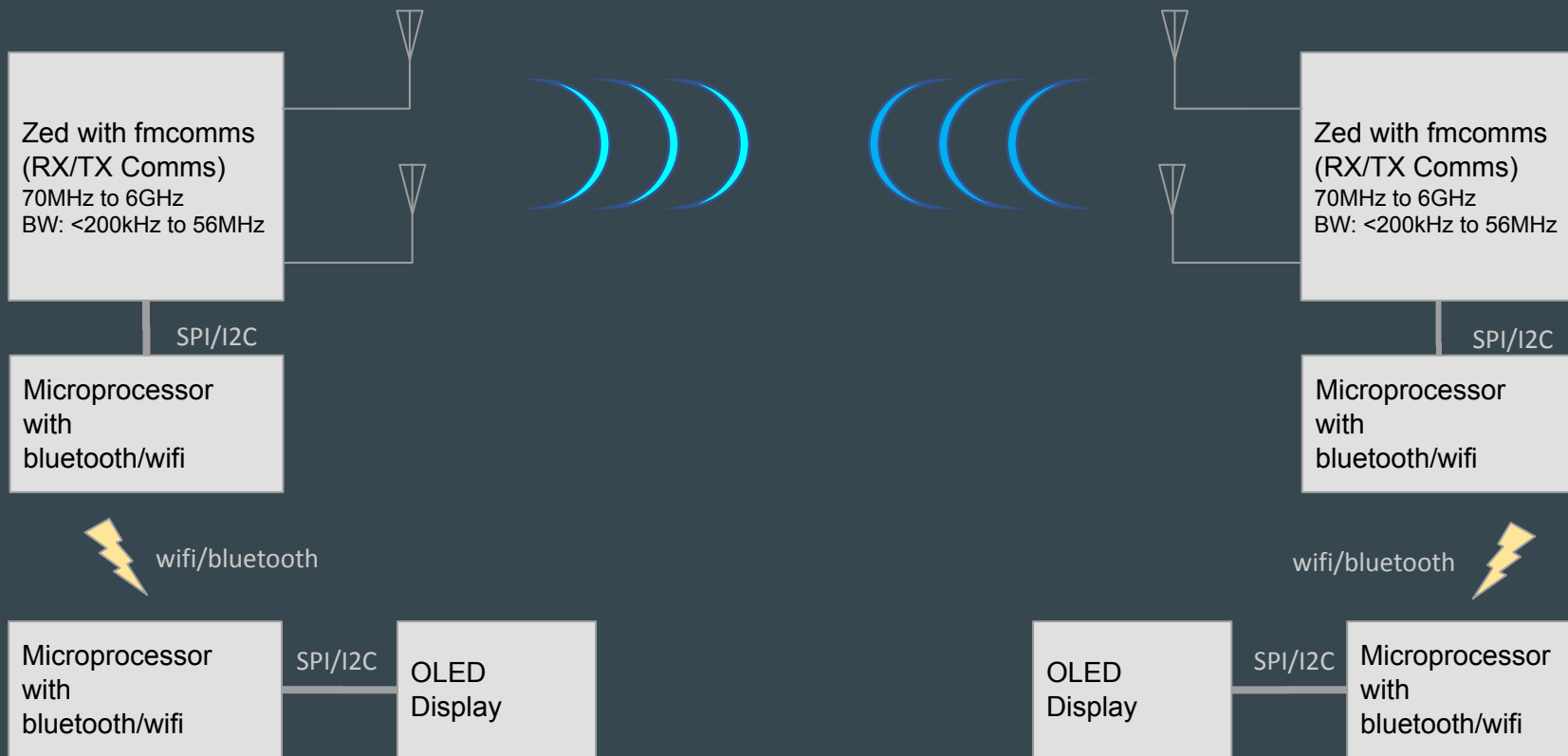


Motivation and Goal

- Our design intends to improve the motorcycle group riding experience, by providing more information to the riders.
- We intend to implement a 2-Node system capable of transmitting



Apparatus / System Diagram





Overall Progress

- Waveform Description defined
 - Data packet defined (info + audio)
 - QPSK modulation at WiFi band (2.4 GHz)
- Simulink Model of Waveform
 - Full TX/RX
 - TX Baseband, TX Modulator, RX Baseband generatable
- RX Demod being designed in HLS/IP Cores
 - MATLAB used to demo and verify components
- OLED and wireless link using arduino
 - OLED displays readable text
 - Wireless link integrated with Simulink generated code
- FMComms4 + Zedboard integration with PYNQ
 - PS to DAC link working



Current Progress



Previous Sprint

- TX
 - Update simulation to have preamble as QPSK
 - Generate data and send from PS to DAC
- RX
 - Modulator
 - Implement AGC (digital and/or fmcomms method)
 - Implement Matched Filter
 - Implement Coarse Frequency Correction
 - Test PL to PS using dummy data
 - Baseband
 - Create test vectors (IO)
 - Setup Simulink model with actual data
 - Process RX Baseband data and display on HUD
- HUD
 - Integrate HUD with PYNQ
 - Start prototype mounts

ACTIVITY	PERIODS	
	1	2
HUD		
Mount OLED		
Screen Interface/ICD		
Integrate HUD with Zed		
HUD Demo (PYNQ to HUD Data Display)		
TX		
Connect TX output (baseband + modulator) to DAC		
Modulator DEMO		
RX		
AGC		
Matched Filter		
Coarse Frequency Correction (FLL)		
Timing/Symbol Recovery		
Fine Frequency Correction (Optional)		
PL to PS interface		
Port Baseband RX processing (simulink code) to ARM		
RX Demodulator Demo		
TDMA MAC Layer		
Verification and Validation		
TWO-WAY LINK DEMO		
Audio Input/Output		
Final Demonstration (COMMS with HUD Integrated)		
	4/1/2022	4/8/2022
	4/7/2022	4/14/2022



Previous Sprint

- TX
 - ~~Update simulation to have preamble as QPSK~~
 - ~~Generate data and send from PS to DAC~~
- RX
 - Modulator
 - ~~Implement AGC (digital and/or fmc comms method)~~
 - ~~Implement Matched Filter~~
 - ~~Implement Coarse Frequency Correction~~
 - ~~Test PL to PS using dummy data~~
 - Baseband
 - ~~Create test vectors (IO)~~
 - ~~Setup Simulink model with actual data~~
 - ~~Process RX Baseband data and display on HUD~~
- HUD
 - Integrate HUD with PYNQ
 - Start prototype mounts

ACTIVITY	PERIODS	
	1	2
HUD		
Mount OLED		
Screen Interface/ICD		
Integrate HUD with Zed		
HUD Demo (PYNQ to HUD Data Display)		
TX		
Connect TX output (baseband + modulator) to DAC		
Modulator DEMO		
RX		
AGC		
Matched Filter		
Coarse Frequency Correction (FLL)		
Timing/Symbol Recovery		
Fine Frequency Correction (Optional)		
PL to PS interface		
Port Baseband RX processing (simulink code) to ARM		
RX Demodulator Demo		
TDMA MAC Layer		
Verification and Validation		
TWO-WAY LINK DEMO		
Audio Input/Output		
Final Demonstration (COMMS with HUD Integrated)		
	4/1/2022	4/8/2022
	4/7/2022	4/14/2022



Transmit

- Update simulation to have preamble as QPSK
 - Makes RX demod processing easier
- Generate data and send from PS to DAC
 - Simulink generated code (TX Baseband + Modulator, RX Baseband) compilable on Zedboard
 - Integrated Simulink generated code with LibIIO library to send TX modulated data out
 - Able to interface with data via jupyter notebook and plot

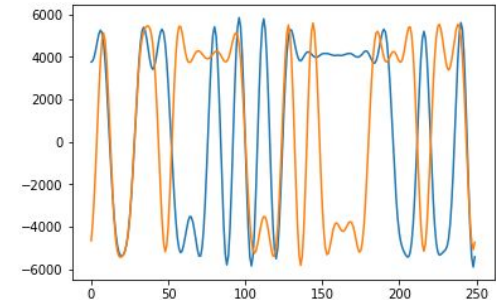
```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import os
```

```
In [9]: fid = open("fmcomms_data.dat", "rb")
data = np.fromfile(fid, dtype='int16')
```

```
In [30]: head = 400000
plot_len = 500
tail = head + plot_len

inphase = data[head:tail:2]
quadrature = data[head+1:tail:2]
```

```
In [31]: plt.plot(inphase)
plt.plot(quadrature)
plt.show()
```





Receive (Demodulator)

- AGC
 - Three options ready for testing
 - Log Based AGC
 - Cordic Amplitude Agnostic Normalizer
 - FMComms built-in AGC
- Matched Filter (SRRC)
 - Implemented using Vivado IP
- Coarse Frequency Correction
 - The Simulink Model uses an FFT approach for the Coarse Frequency Correction.
 - Investigated Band Edge (BE) Filtering
 - Preamble Frequency Estimation (Moving Forward)
- Design approach
 - Matlab Model
 - HLS Testbench -> Matlab Model for verification

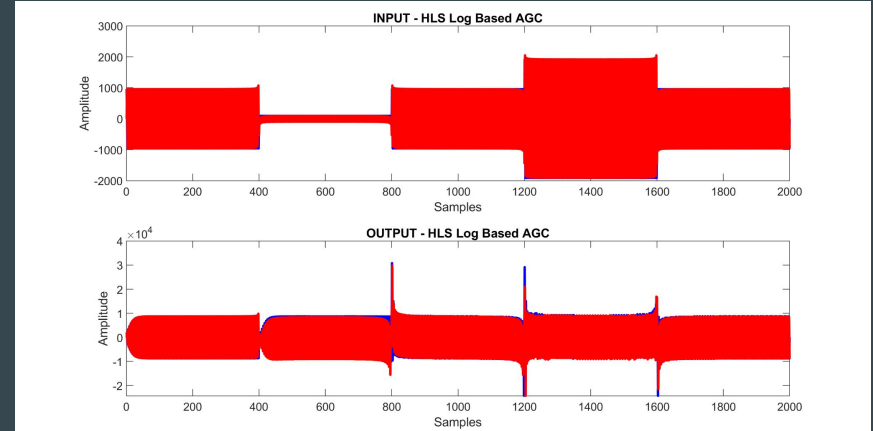


Receive (Demodulator): Log AGC

Began with a Matlab Test Script:

```
loggain = zeros(1,2001);  
loggain(1) = 0;  
mu = 0.1;  
for nn = 1:2000  
    scl(nn)= IQ(nn)*exp(loggain(nn));  
    loggain(nn+1) = loggain(nn)- mu*log(abs(scl(nn))+0.1));  
end
```

- Could use any log so chose to use `log2()` and `exp2()`.
- HLS provides and `hls::log2` function. Then we used a look-up-table (LUT) for `exp2()`.
- Added limits to gain so to control size of LUT.



	Interval	BRAM	DSP	FF	LUT
agc	11	4	2	422	1282

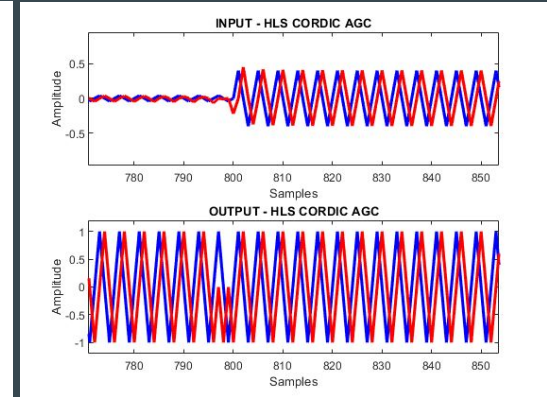
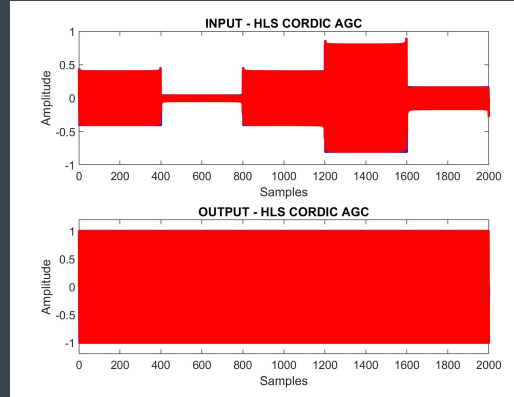


Receive (Demodulator): Normalizer

Again we began with a Matlab Test Script:

```
phase = atan2(Quadrature,InPhase);  
ylnorm = exp(1i*phase);
```

- In this approach we can use two cordics.
- Pros:
 - No Overshoot
 - No BRAM
- Cons:
 - Longer interval
 - Unsure about accuracy given phase noise and cordic precision.

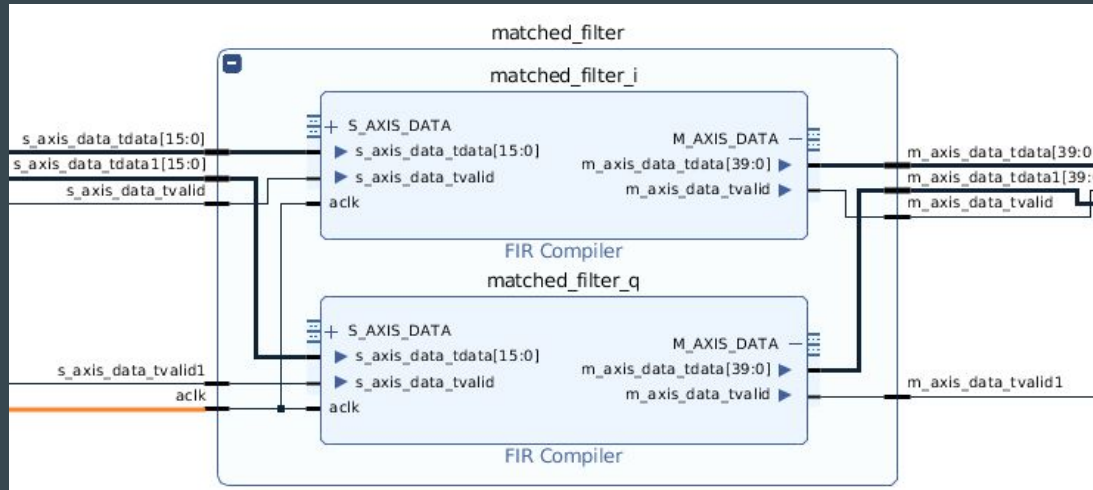


	Interval	BRAM	DSP	FF	LUT
agc	25	0	2	1952	7657

Note: We believe DSPs will be a limiting factor in our design (only 220 available)

Receive (Demodulator): Matched Filter (SRRC)

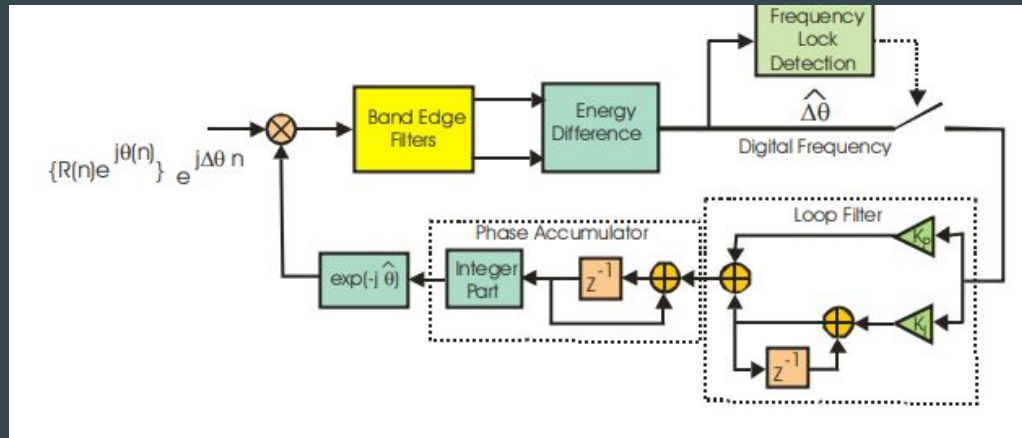
- The SRRC was easy to implement. We were able to use the Simulink Model to provide the coefficient.
- Used Xilinx FIR IP
- No complex coefficients so we can handle the complex data by doubling filters.





Receive (Demodulator): Band Edge FLL

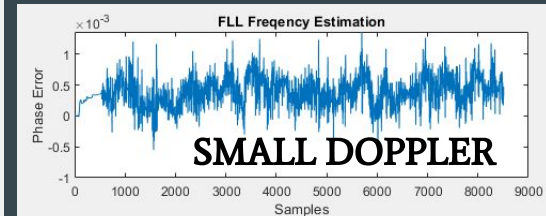
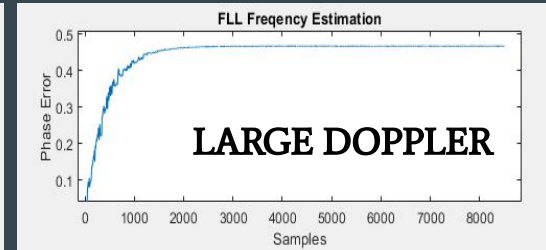
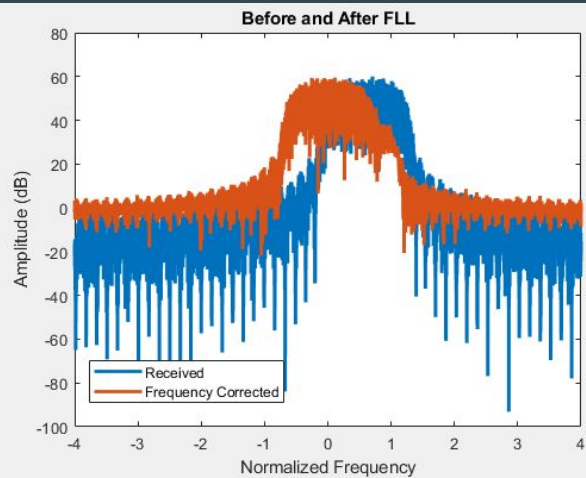
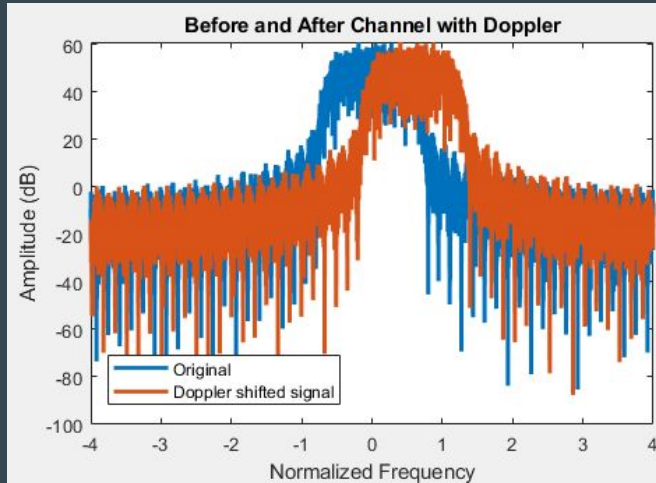
- We began research Frequency Correction Methods. This led us to fred harris' paper on System Synchronization:
 - Harris, F (2011). Let's Assume the System Is Synchronized. In: Prasad, R., Dixit, S., van Nee, R., Ojanpera, T. (eds) Globalization of Mobile and Wireless Communications. Signals and Communication Technology. Springer, Dordrecht. https://doi.org/10.1007/978-94-007-0107-6_20





Receive (Demodulator): Band Edge FLL - Investigation

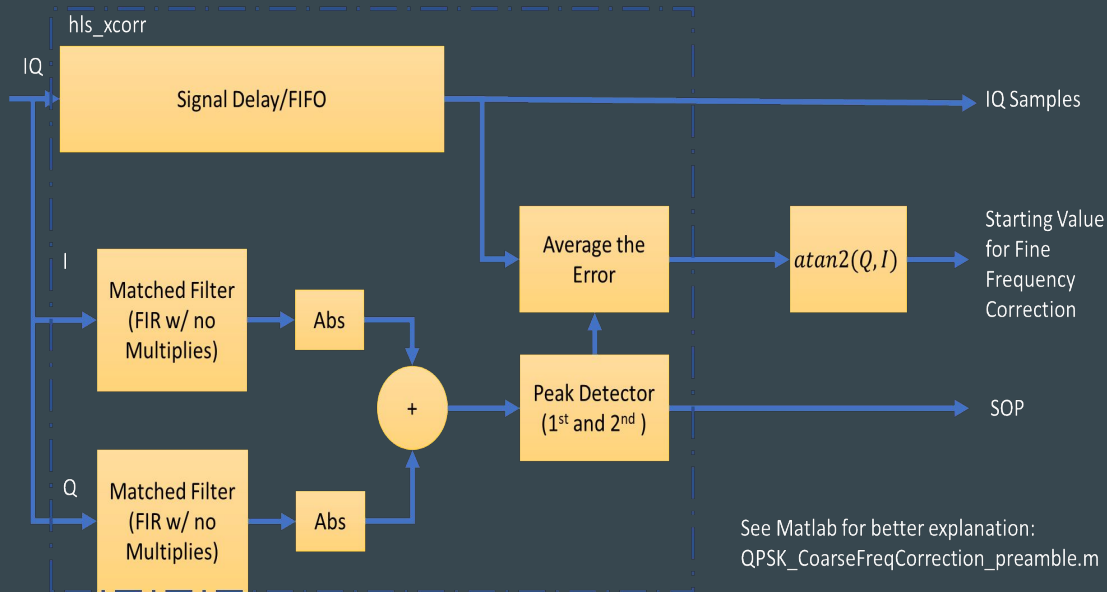
- Matlab model
 - Symbols \rightarrow Add Preamble \rightarrow SRRC \rightarrow Channel \rightarrow SRRC \rightarrow AGC \rightarrow FLL
- Modulator** **Demodulator**
- Worked great for large doppler we had issues with smaller doppler values.





Receive (Demodulator): Preamble Coarse Frequency Estimator

- After some investigating of the BE Filters we then decided to use the information we can acquire from the preamble. This data will then be fed to the fine frequency correction.



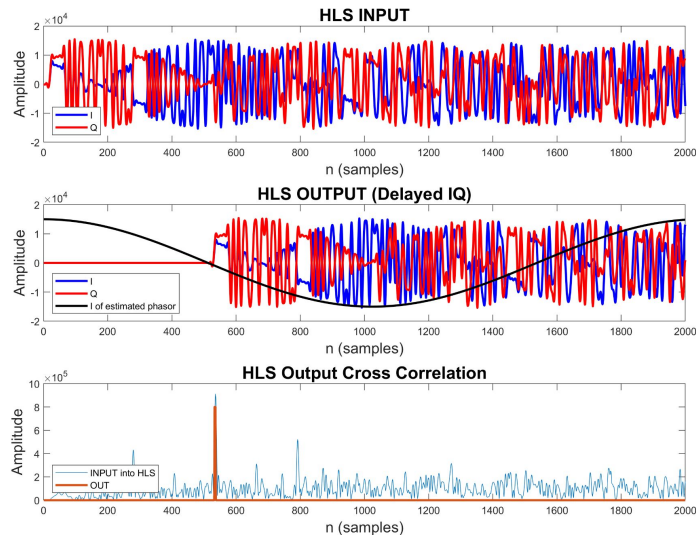
- New Matlab model for this based off BE FLL matlab script.



Receive (Demodulator): Preamble Coarse Frequency Estimator

- After some investigating of the BE Filters we then decided to use the information we can acquire from the preamble. This data will then be fed to the fine frequency correction.

- Black line is estimated correcting sinusoid
- Red spike is our IP core's peak detection.



PL to PS

- Communication via libiio is working.
- Also implemented DMA using Userspace IO (UIO) for added flexibility.
- UDMA kernel was available from github

([ikwzm/udmabuf: User space mappable dma buffer device driver for Linux. \(github.com\)](https://github.com/ikwzm/udmabuf))


- Installed the kernel modules for uio_pdrv_irq.ko and udmabuf.ko

```
root@pynq:/home/xilinx# ls /dev/
block      iio:device1  loop6      mtd2ro     null       ram4        tty0        tty20      tty32      tty44      tty56      udmabuf-rx  vcsa2
bus        iio:device2  loop7      mtd3       ptmx       ram5        tty1        tty21      tty33      tty45      tty57      udmabuf-tx  vcsa3
char       iio:device3  loop-control mtd3ro     pts        ram6        tty10       tty22      tty34      tty46      tty58      uio0        vcsa4
```



Receive (Baseband)

- Create test vectors (IO)
 - Saved test vectors in binary files for all IO except the RX demod (as it is not code generation compatible)
- Setup Simulink model with actual data
 - Data packet (name, location, etc) oscillates between the two sets of data on the right
- Process RX Baseband data and display on HUD
 - Shown in the HUD demo



v2x_rx_bb_in.bin
v2x_rx_bb_out.bin
v2x_tx_bb_in.bin
v2x_tx_bb_out.bin
v2x_tx_mod_out_imag.bin
v2x_tx_mod_out_real.bin

Name: V2X!
Location: UCSD
Speed: 60
Navigation: Dir = 3, Dist = 5.3

Name: Home
Location: San Jose, CA
Speed: 45
Navigation: Dir = 1, Dist = 0.6789



HUD - Test Vectors at Baseband

```
% Name: V2X!  
name = 'V2X!';  
name_bin_mat = dec2bin(name, 8) - '0';  
name_bin_mat2 = name_bin_mat.';  
name_bin = name_bin_mat2(:);  
  
% Location: UCSD  
pos.lat = single(32.880100);  
pos.lon = single(-117.234000);  
pos.lat_bin = (float_2_bin(pos.lat) - '0').';  
pos.lon_bin = (float_2_bin(pos.lon) - '0').';  
  
% Speed: 60  
speed = uint8(60);  
speed_bin = (dec2bin(speed, 8) - '0').';  
  
% Navigation: Directions = 3, Distance to next step = 5.3  
nav.dir = uint8(3);  
nav.dtns = single(5.3);  
nav.dir_bin = (dec2bin(nav.dir, 8) - '0').';  
nav.dtns_bin = (float_2_bin(nav.dtns) - '0').';  
  
% Concatenate array  
info_pkt = [name_bin; pos.lat_bin; pos.lon_bin; speed_bin; ...  
            nav.dir_bin; nav.dtns_bin];
```

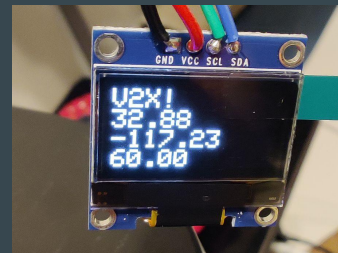
Create Matlab Test Vector and
send stream of bits

```
▼ pyld: 0x7fffffffedba0  
▼ name  
  [0]: 86 'V'  
  [1]: 50 '2'  
  [2]: 88 'X'  
  [3]: 33 '!'  
lat: 32.8801003  
lon: -117.234001  
speed: 60 '<'  
dir: 3 '\003'  
dist_next_step: 5.30000019
```

Parse data in C

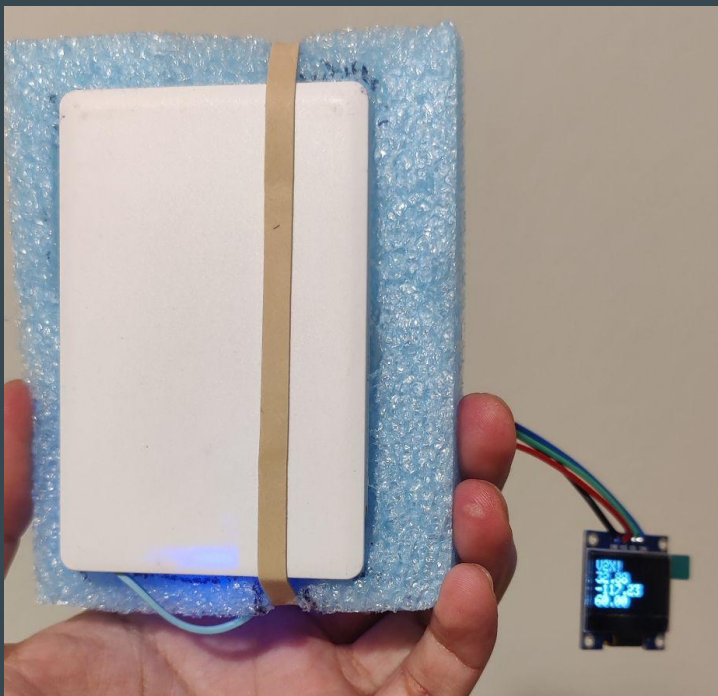
Create HTTP Client

```
"GET /name?value=V2X! HTTP/1.1\r\n\r\n"
```



Update HUD
values in real time

HUD Demo!





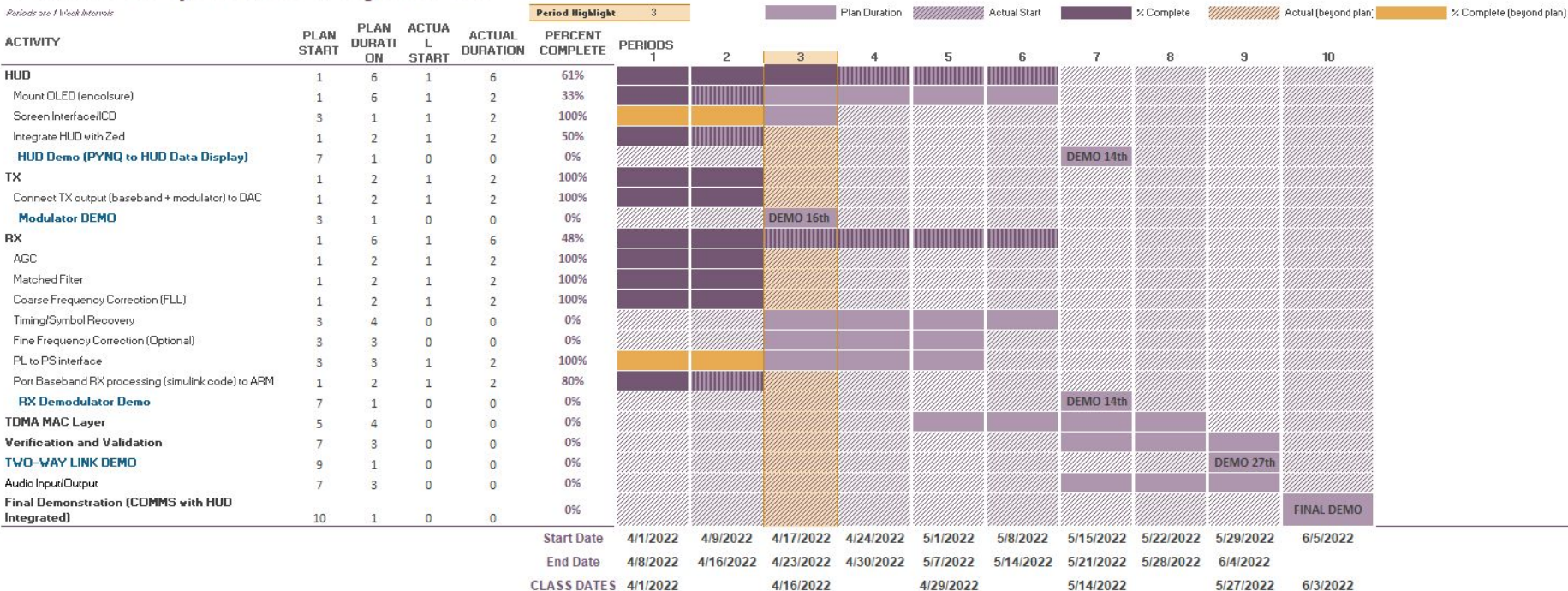
Future Plans



Gantt (timeline)

V2X Motorcycle HUD Project Plan

Periods are 1 Week Intervals





Next Sprint

- Demo
 - Create data file (name, location, etc) for demo
 - Create audio test file for final demo
 - Integrate data/audio files with TX modulator SW
- RX
 - Insert AGC, SRRC, and Coarse Frequency Correction into Vivado design.
 - Verilog Test Bench of RX Chain
 - Implement Timing Error and Symbol Recovery
 - Implement Fine Frequency Correction
 - (Start) Integrate with Baseband RX software
- HUD
 - Complete integration with Zed
 - Continue work on Mount/Enclosure

Additional Slides



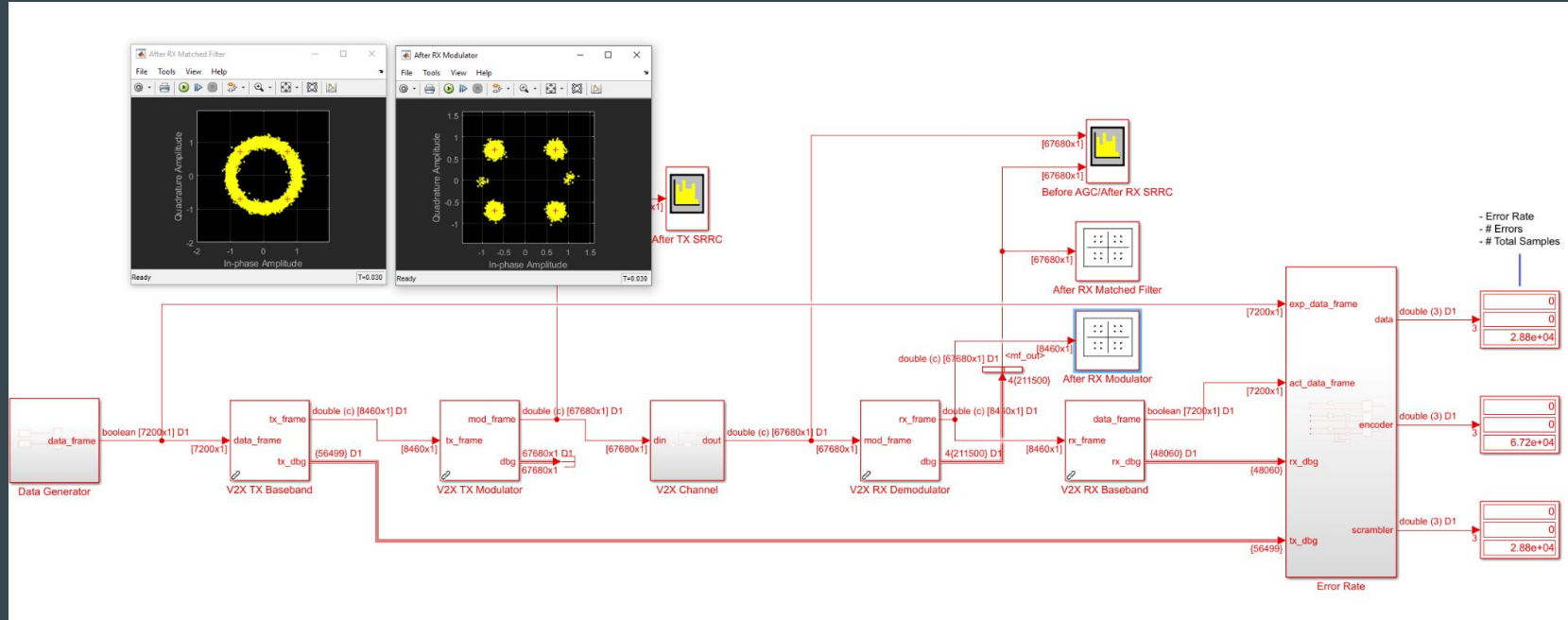
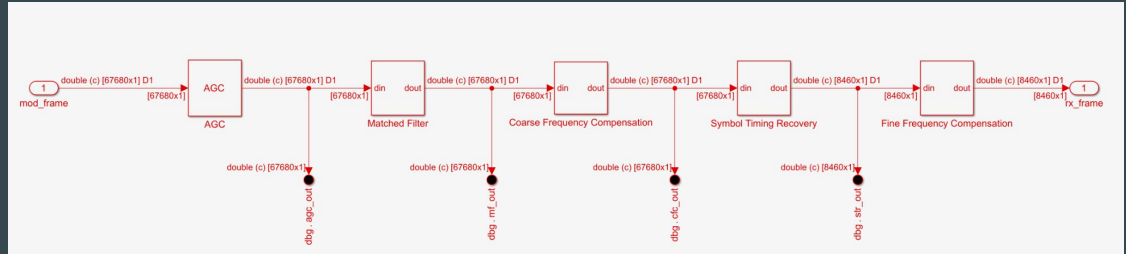
Current state of affairs

Waveform Description

- Datalength: 7200 bits (Data + 0.01s audio).
 - Throughput: $7200 \text{ bits} * (1/0.01\text{s}) = 720 \text{ kbps}$
- Minimum Required Sampling Frequency (before carrier mixer)
 - After TX processing/modulation: 67200 sym
 - $\text{Min } F_s = 67200 \text{ sym} * (1/0.01\text{s}) * 2 = 13.44 \text{ Msps}$
- Max Doppler Frequency:
 - $F_c = 2.4 \text{ GHz}$ (WiFi)
 - Max speed (x2): 200 MPH = 89.41m/s
 - $F_d = 2,400,715 \text{ Hz}$ (offset = 715 Hz)
- $\text{Max } f_e < 1/2T$
 - $1/2T = f_s/2L$
 - $f_s/2L = 100 \text{ MHz}/(2*64) = 781.25 \text{ KHz}$

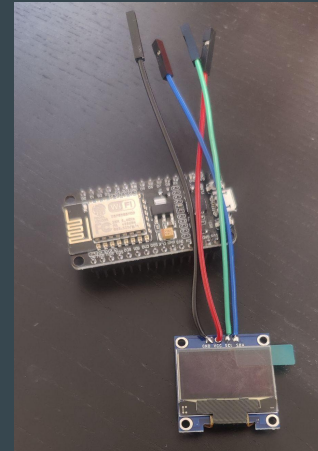
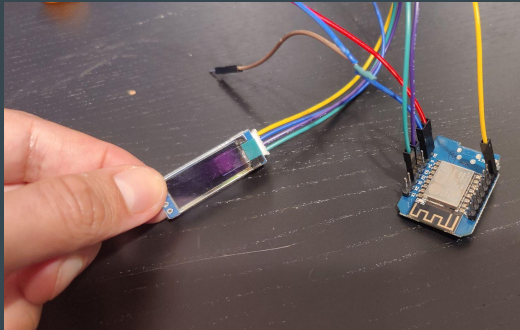
Waveform Description (cont)

QPSK sim with phase + freq offset and AWGN



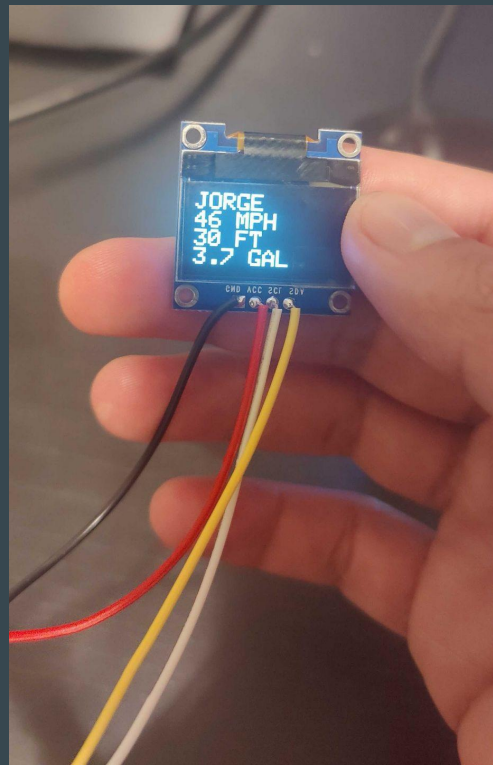
HUD Progress

- Encountered issues with first batch of components
 - Assembled all parts and soldered components
 - Created test programs on Arduino IDE to configure ESP8266 (WiFi) and SSD1306 (OLED)
 - Couldn't control OLED with SPI or I2C
 - Ordered a new set of parts with same controllers



HUD Progress (cont)

- Configured OLED to receive subset of data we plan to transmit
 - Created test programs to fiddle with WiFi server functionality
 - Connected WiFi module to local network and was able to toggle individual GPIO pins through a client connected to the network
- Next steps
 - Create a server to listen to our incoming traffic based on our requirements
 - Display text in real time on OLED
 - Work on enclosure



FMComms+Zed Board Integration

- Working SD Card PYNQ image
 - ADI provided base HDL project for FPGA.
 - Used Petalinux to generate boot files with desired FMComms4 device tree.
 - Integrated Petalinux and PYNQ Root Filesystem
 - Installed LibIIO library
 - Created a Transmit and Receive Demonstration.
 - Sends to Tx DMA → DAC
 - Reads from Tx DMA ← ADC and stores to file.
 - On our github.

```
In [25]: import numpy as np
import matplotlib.pyplot as plt
import os

In [38]: fid = open("./iio_stream/build/fmcomms_data.dat", "rb")
data = np.fromfile(fid, dtype='int16')

In [39]: inphase=data[1:40000:2]
quadrature = data[0:40000:2]

In [40]: plt.plot(inphase)
plt.plot(quadrature)
plt.show()
```

