Half-duplex system design using USRP N321

5G Testbed, IIT Madras

ID4100

Milind Kumar V

EE16B025

June 14, 2020

Contents

List of Figures

1 Introduction

This technical report discusses the work done at the Indigenous 5G Testbed at IIT Madras to develop a half duplex [1] system operating at 122.88 MSPS and transmitting data generated according to 5G NR standards using a Universal Software Radio Peripheral (USRP). A half duplex system can both transmit and receive but can not do both simultaneously. Instead, the system transmits for a period of time and receives for the remaining. Therefore, the system is designed in three stages- transmitter, receiver and then integration of the two. Once designed, the USRP is expected to work in parallel with the existing Keysight systems if not replace them.

In this work, USRP N321 by Ettus Research [2] is used. The USRP is a software defined radio that uses software as a substitute for components implemented in hardware. This document details the procedures involved in setting up the USRP- the associated software, interfacing and verification of function. This document will also discuss the design of a transmitter and receiver using GNU Radio and the associated challenges. The designed systems are tested using a Vector Signal Analyzer (VSA) and Vector Signal Generator (VSG) and data that adheres to the 5G standards is generated using MATLAB.

2 USRP N321

2.1 Features

USRP N321 is a software defined radio that provides two TX and RX channels each with a bandwidth of 200 MHz- a prerequiste for transmission at the required sampling rate. The device also provides 1 GbE and 10 GbE interfaces over two SFP+ ports and 1 QSFP+ port which enable high throughput streaming of IQ data to a host computer. Figures 1 and 2 display the front and rear panels of the USRP. Further, the USRP operates only in the 3-6 GHz range i.e in FR1.

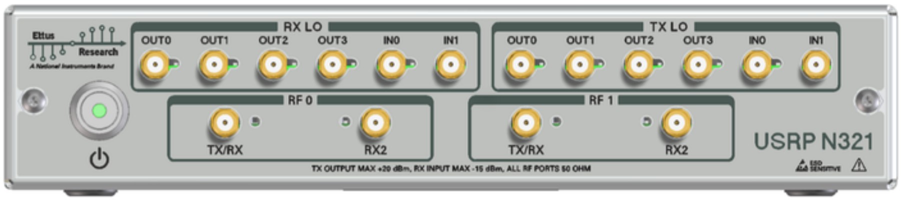


Figure 1: USRP N321 front panel

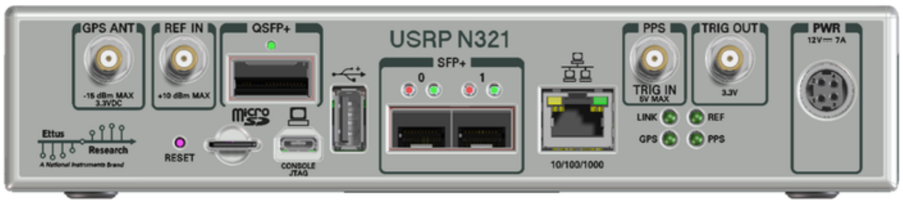


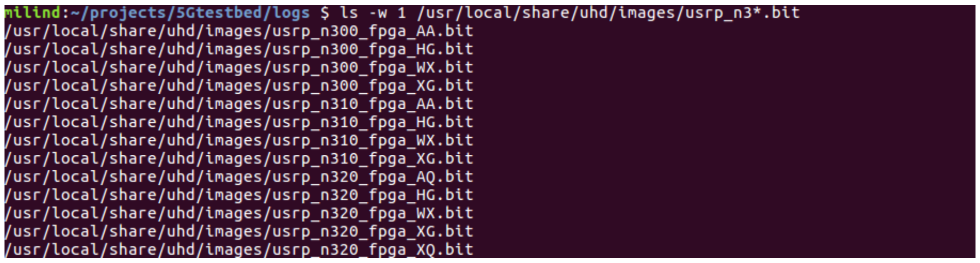
Figure 2: USRP N321 rear panel

2.2 Necessary software setup

At the 5G testbed at IIT Madras, a computer running Ubuntu 16.04 is used as the host computer. In order to operate the USRP, it is necessary to install the following- the USRP Hardware Driver (UHD) and GNU Radio, both of which are open source. The sequence of steps to be followed to correctly set up the necessary drivers and software can be found at [3].

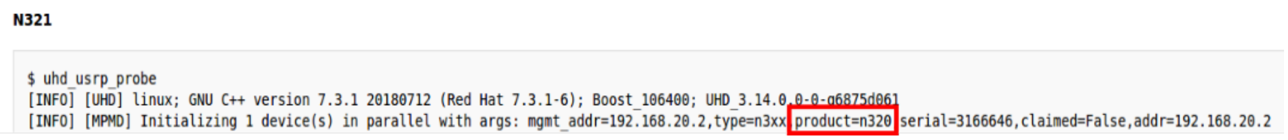
2.2.1 UHD

UHD is a software driver and API for the USRP and is created by Ettus Research. It can be used to build stand-alone applications or alongside frameworks such as GNU Radio. USRP N321 requires a UHD version 3.14.0.0 or above. Consequently,UHD version 3.14.1 (Section 2.2.3 describes how this can be determined once the installation is complete) is used in the laboratory. If an imprpoper UHD version is used, the host computer will be unable to detect the USRP device despite being connected to it. Consequently, UHD is built from source to ensure the selection of the right version of the driver. This can be done by following the instructions in the "Building and installing UHD from source code" section of [3]. It is also necessary to download the UHD FPGA images. Figure 3 provides a list of the available FPGA images. Note that while all the images appear to be necessary for n320, this does not affect the operation of USRP N321. This is made even more clear from the output of the uhd_usrp_probe command (discussed in Section 2.2.3) whose output is shown in Figure 4 Further, the 1 GbE and 10 GbE interfaces require the use of different FPGA images. The HG image supports both the interfaces and is hence used throughout this work.



```
milind:~/projects/5Gtestbed/logs $ ls -w 1 /usr/local/share/uhd/images/usrp_n3*.bit
/usr/local/share/uhd/images/usrp_n300_fpga_AA.bit
/usr/local/share/uhd/images/usrp_n300_fpga_HG.bit
/usr/local/share/uhd/images/usrp_n300_fpga_WX.bit
/usr/local/share/uhd/images/usrp_n300_fpga_XG.bit
/usr/local/share/uhd/images/usrp_n310_fpga_AA.bit
/usr/local/share/uhd/images/usrp_n310_fpga_HG.bit
/usr/local/share/uhd/images/usrp_n310_fpga_WX.bit
/usr/local/share/uhd/images/usrp_n310_fpga_XG.bit
/usr/local/share/uhd/images/usrp_n320_fpga_AQ.bit
/usr/local/share/uhd/images/usrp_n320_fpga_HG.bit
/usr/local/share/uhd/images/usrp_n320_fpga_WX.bit
/usr/local/share/uhd/images/usrp_n320_fpga_XG.bit
/usr/local/share/uhd/images/usrp_n320_fpga_XQ.bit
```

Figure 3: List of the available FPGA images for USRP N321



N321

```
$ uhd_usrp_probe
[INFO] [UHD] linux; GNU C++ version 7.3.1 20180712 (Red Hat 7.3.1-6); Boost_106400; UHD_3.14.0.0-0-g6875d061
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.20.2,type=n3xx,product=n320,serial=3166646,claimed=False,addr=192.168.20.2
```

Figure 4: Output of uhd_usrp_probe when USRP N321 is connected- product is n320

2.2.2 GNU Radio

GNU Radio is a software that provides signal processing blocks that can be used to implement software radios. GNU Radio can be built from source by following the instructions provided in the "Building and installing GNU Radio from source code" section of [3]. Version 3.7.13.4 is used in the laboratory.

2.2.3 Testing the installation

As described in the "Connect the USRP" section of [3], the USRP can be connected to the host computer via either an ethernet or SFP port (see 2.3 for more details). In the laboratory, this done by connecting the USRP to the 10 GbE lab network via optical cables using the SFP port. The availability of the USRP can be determined using the following command (for the 10 GbE connection)

```
ping 192.168.20.2
```

Correct functioning of the USRP drivers can be established by using the following command

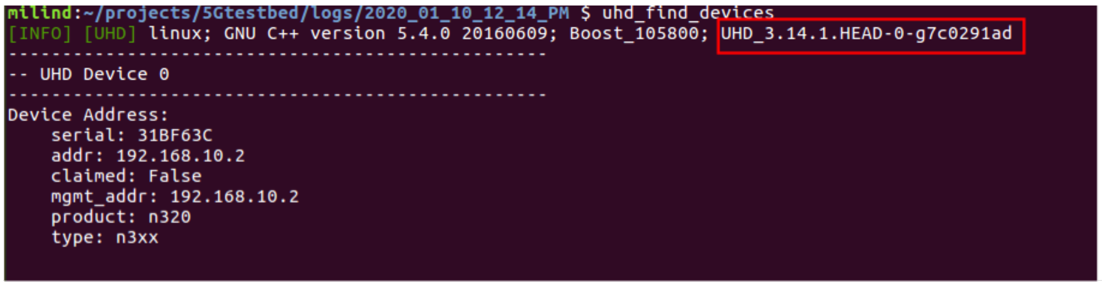
```
uhd_find_devices
```

in the console. When the installation is correctly set up, this yields the output shown in Figure 5. As is highlighted in the figure, this can also be used to determine the UHD version.

Following this, the command

```
uhd_usrp_probe
```

can be used to check UHD functionality. Another simple test to establish whether the USRP and the UHD are functioning correctly is to run one of the example programs that come with the UHD driver as described in the "ASCII Art Example" section of [4] using the command

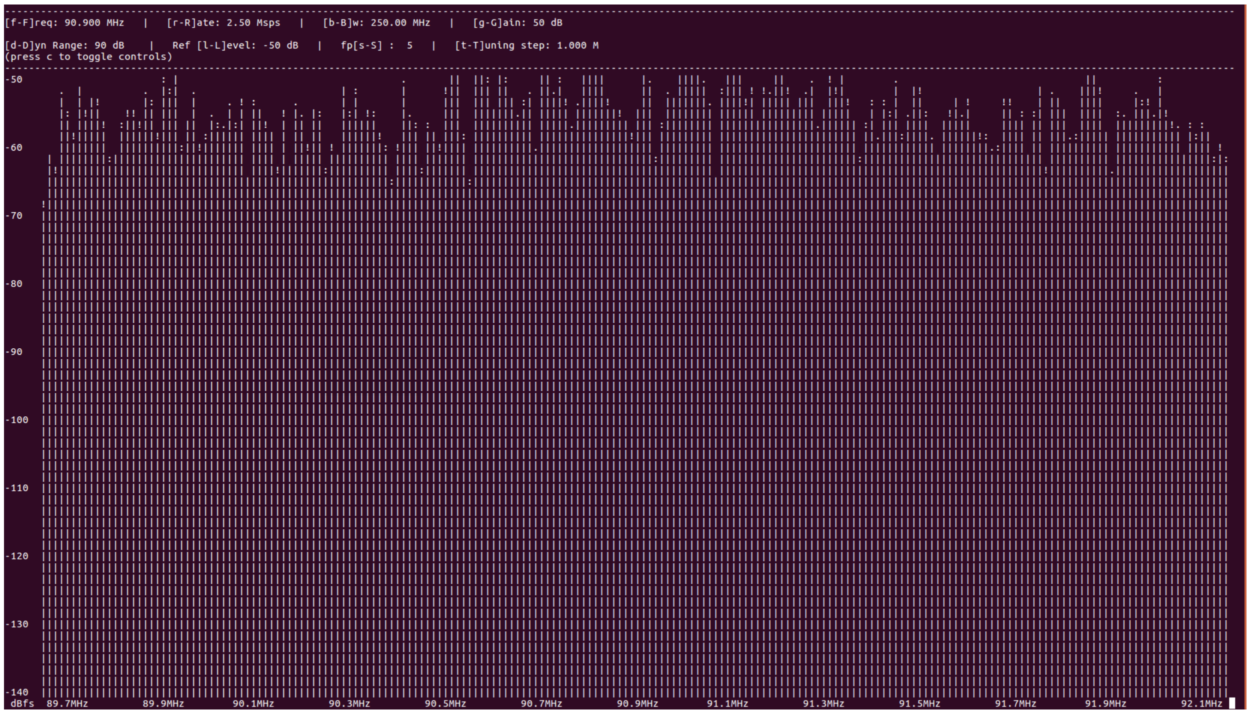


```
milind:~/projects/5Gtestbed/logs/2020_01_10_12_14_PM $ uhd_find_devices
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.14.1.HEAD-0-g7c0291ad
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    serial: 31BF63C
    addr: 192.168.10.2
    claimed: False
    mgmt_addr: 192.168.10.2
    product: n320
    type: n3xx
```

Figure 5: Output of uhd_find_devices when USRP N321 is connected

```
/usr/local/lib/uhd/examples/rx_ascii_art_dft --args "master_clock_rate=250e6,
mgmt_addr=192.168.10.2,addr=192.168.10.2" --freq 91.9e6 --rate 2.5e6 --gain
50 --ref-lvl="-50" --dyn-rng 90 --ant "RX2" --subdev "A:0"
```

where mgmt_addr and addr are obtained from Figure 5. This allows one to observe the spectrum of a commercial FM radio station in real time. The results can be seen in Figure 6. A VERT2450 antenna is used for this example.



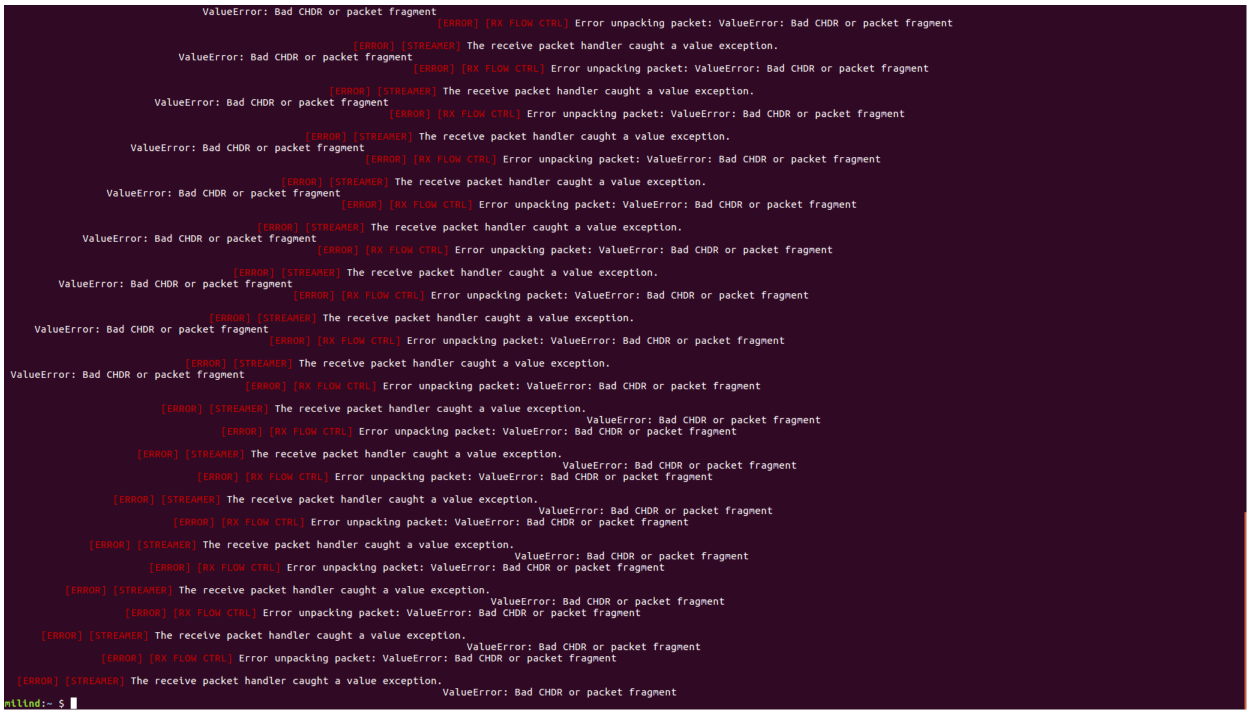
```
[f-F]req: 90.900 MHz   |   [r-R]ate: 2.50 Msps   |   [b-B]w: 250.00 MHz   |   [g-G]ain: 50 dB

[d-D]yn Range: 90 dB    |   Ref [l-L]evel: -50 dB   |   fp[s-S] :  5   |   [t-T]uning step: 1.000 M
(press c to toggle controls)
```

Figure 6: Output of the ASCII Art Example program that tests USRP and UHD functionality

2.3 Interfacing with the USRP

As discussed in Section 2.1, there are several methods to interface with the USRP. This is described in detail in [4]. From Figure 2 it is evident that a host computer can connect to the USRP using the 2 SFP ports (using optical cables), a RJ45 interface (using a cat5e cable) and serial console connection (using a USB cable). Both the SFP and RJ45 interfaces require that the host computer be configured according to the instructions in the "Setting Up a Streaming Connection" section of [4]. When using these interfaces, the USRP is accessed via SSH. Alternately, the ARM CPU can be accessed using the serial console connection.

An error encountered when running the FM spectrum analyzer described in Section 2.2.3 code using SFP0 serves as a good example of when these multiple connections can be used. By default, the MTU value of SFP0 is set to 8000. The correct value being 1500, the error shown in Figure 7 is produced.



```
                                 ValueError: Bad CHDR or packet fragment
                                                                        [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                                          [ERROR] [STREAMER] The receive packet handler caught a value exception.
                             ValueError: Bad CHDR or packet fragment
                                                                    [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                                      [ERROR] [STREAMER] The receive packet handler caught a value exception.
                         ValueError: Bad CHDR or packet fragment
                                                                [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                                  [ERROR] [STREAMER] The receive packet handler caught a value exception.
                     ValueError: Bad CHDR or packet fragment
                                                            [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                              [ERROR] [STREAMER] The receive packet handler caught a value exception.
                 ValueError: Bad CHDR or packet fragment
                                                        [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                          [ERROR] [STREAMER] The receive packet handler caught a value exception.
             ValueError: Bad CHDR or packet fragment
                                                    [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                      [ERROR] [STREAMER] The receive packet handler caught a value exception.
         ValueError: Bad CHDR or packet fragment
                                                [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                                  [ERROR] [STREAMER] The receive packet handler caught a value exception.
     ValueError: Bad CHDR or packet fragment
                                            [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                              [ERROR] [STREAMER] The receive packet handler caught a value exception.
 ValueError: Bad CHDR or packet fragment
                                        [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                          [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                                                 ValueError: Bad CHDR or packet fragment
                                    [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                      [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                                             ValueError: Bad CHDR or packet fragment
                                [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

                  [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                                         ValueError: Bad CHDR or packet fragment
                            [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

              [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                                     ValueError: Bad CHDR or packet fragment
                        [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

          [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                                 ValueError: Bad CHDR or packet fragment
                    [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

      [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                             ValueError: Bad CHDR or packet fragment
                [ERROR] [RX FLOW CTRL] Error unpacking packet: ValueError: Bad CHDR or packet fragment

  [ERROR] [STREAMER] The receive packet handler caught a value exception.
                                                                         ValueError: Bad CHDR or packet fragment
milind:~ $
```

Figure 7: Error produced due to incorrect MTU value

This can not be corrected via a connection established through SFP0 as that link must first be turned off for the MTU value to be reset. Therefore, the USRP can first be accessed using the method described in the "Setting up a Serial Console Connection" section of [4] and the MTU value can be changed using the following commands

```
ifconfig sfp0 down
ifconfig sfp0 mtu 1500
ifconfig sfp0 up
```

2.4 USRP operation procedure

Here, a brief description of the method to correctly and safely handle the USRP is provided.

- Connect to the host system before power up using one of the previously specified interfaces and plug in the power cord. Do not power on the device.

If using SFP0, first edit the MTU value using the following series of steps

- Set up a screen session using the command

  ```
  sudo screen /dev/serial/by-id/usb-Silicon_Labs_CP2105_Dual_USB_to_
  UART_Bridge_Controller_009781FB-if00-port0 115200
  ```

  This command is determined by the output of the command (see Section 2.3 regarding setting up a serial connection)

  ```
  ls /dev/serial/by-id
  ```

- Power on the USRP. Follow the steps described in Section 2.3 to change the MTU value.
- Exit the screen session using the command

  ```
  exit
  ```

If not using SFP0, power on directly. Run the commands

```
uhd_find_devices
uhd_usrp_probe
```

to verify the function of the USRP. It is likely that the latter command will produce a warning regarding buffer size whilst supplying a command to fix it.

Once the USRP is powered on, ensure to terminate any TX/RX terminals with an antenna with 50 Ω impedance to prevent damage. The necessary GNU Radio programs can be run subsequently.

The USRP can then be shut down by logging in via SSH and using the command

```
shutdown -h now
```

Simply powering off the USRP can cause severe damage and corruption of the file system.

3 Half-duplex system design

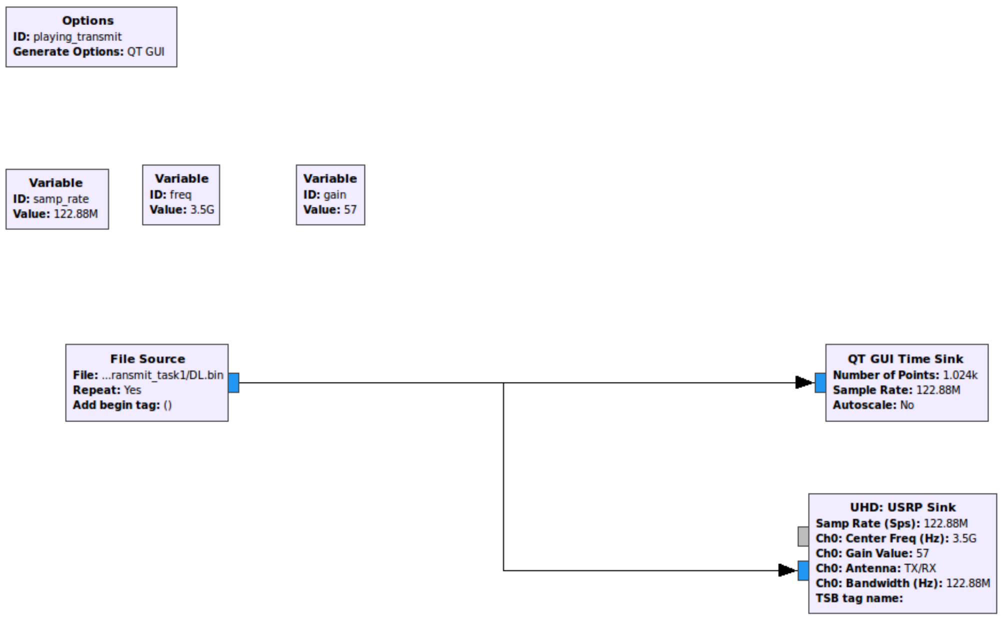
As discussed before, the USRP can be used in conjunction with GNU Radio to design a transmitter and receiver. While this document does not discuss the full design, it covers the implementation and testing of the transmitter and receiver and presents the unanswered questions remaining in each case.

3.1 Transmitter

Signal processing blocks necessary to implement the transmitter are easily found in the GNU Radio GUI. 64 QAM IQ data for four users each with a specific resource allocation is transmitted using the USRP. Transmission occurs at 3.5 GHz and 122.88 MSPS. This rate requires that the USRP be connected to the 10 GbE network.

3.1.1 Transmitter design

The transmitter is designed in GNU Radio using a "File Source" signal processing block that takes in a binary file, scales and transmits it. Figure 8 presents the design of the transmitter in GNU Radio.



Options
ID: playing_transmit
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 122.88M

Variable
ID: freq
Value: 3.5G

Variable
ID: gain
Value: 57

File Source
File: ...ransmit_task1/DL.bin
Repeat: Yes
Add begin tag: ()

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 122.88M
Autoscale: No

UHD: USRP Sink
Samp Rate (Sps): 122.88M
Ch0: Center Freq (Hz): 3.5G
Ch0: Gain Value: 57
Ch0: Antenna: TX/RX
Ch0: Bandwidth (Hz): 122.88M
TSB tag name:

Figure 8: Transmitter implemented using GNU Radio

As transmission occurs at 3.5 GHz, the VERT2450 antenna can no longer be used. Therefore, transmission was initially performed with a wired setup and wireless transmission was performed using the antenna in Figure 9.

Figure 9: Antenna (TG.55.8113) used to transmit at 3.5 GHz

A vector signal analyzer (VSA) is used to analyze the received waveform.

3.1.2 Results and discussion

The success of transmission is measured by the VSA being able to decode (and display) the constellation used and recover the transmitted waveform. The SNR is observed from the spectrum obtained. The Error Vector Magnitude (EVM) [5] measurement is also a useful metric in this scenario. Figure 10 presents the output of the VSA when 3 cables connected in series were used as the transmission medium at a transmitter gain of 16 dB. Figure 10b provides the EVM measurement for the same. Figure 11 presents VSA output for wireless transmission. Predictably, much higher transmitter gains are necessary to attain EVM values comparable to those in the wired case. The VSA first detects something as being transmitted at 25 dB (see Figure 11a).

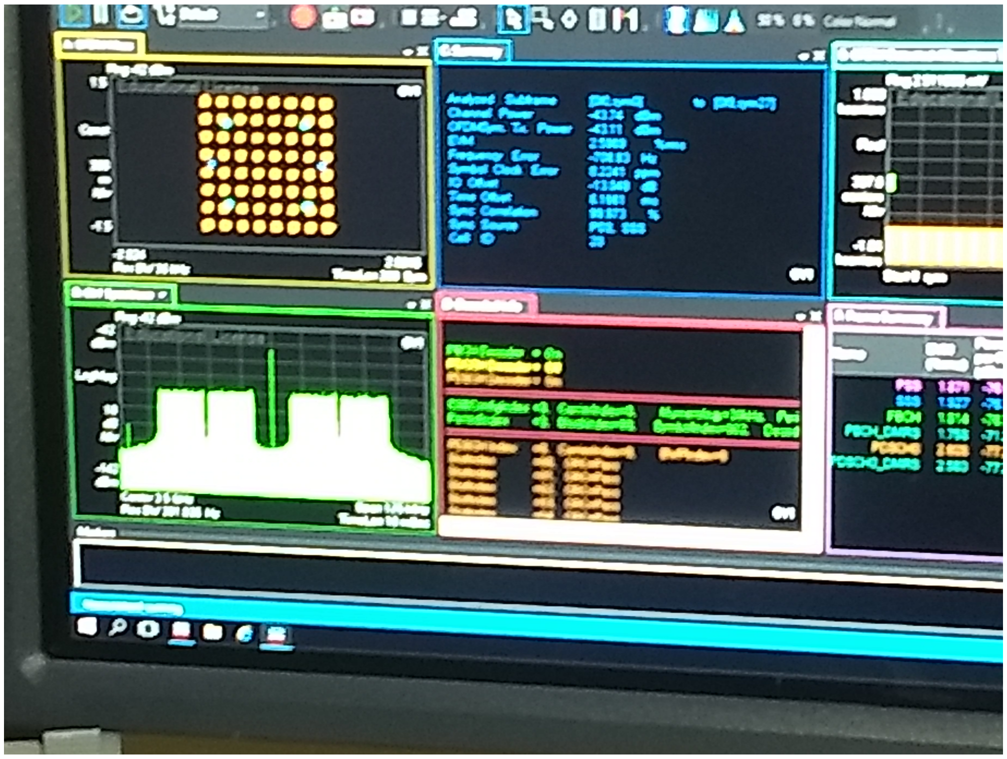
One computation that remains is to verify that the SNR at the receiver matches the net sum of gain and loss from the transmitter and channel respectively. The next step of the transmission is to use 16 QAM data instead of 64 QAM.

3.2 Receiver

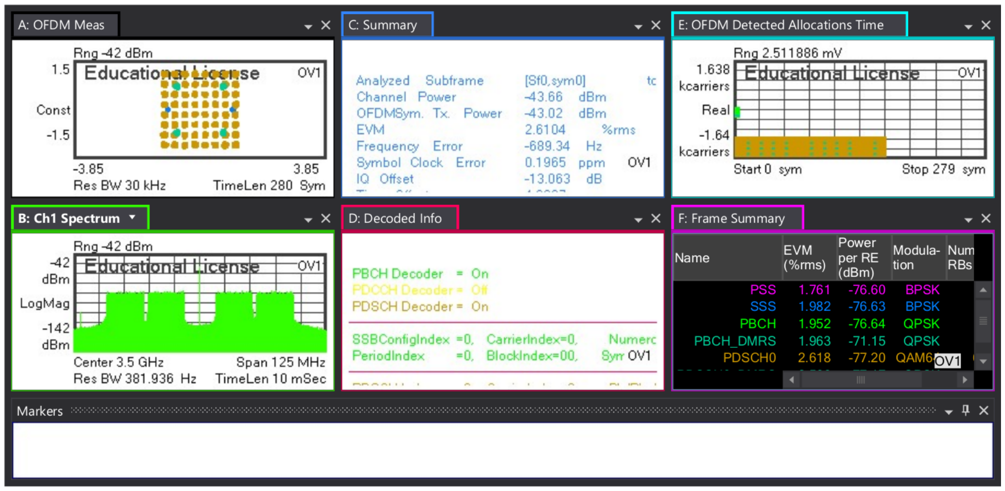
The receiver too is implemented using signal processing blocks present in GNU Radio. The receiver implementation presents far more significant challenges than does the transmitter in terms of frequency offset, timing estimation and demodulation.

3.2.1 Receiver design

Figure 12 presents the signal processing blocks and their interconnections in the receiver. At the transmitter i.e at the Vector Signal Generator (VSG), data for one frame is transmitted



(a)



C: Summary

Analyzed Subframe	[Sf0,sym0]	tc
Channel Power	-43.66	dBm
OFDMSym. Tx. Power	-43.02	dBm
EVM	2.6104	%rms
Frequency Error	-689.34	Hz
Symbol Clock Error	0.1965	ppm
IQ Offset	-13.063	dB

OV1

D: Decoded Info

PBCH Decoder = On
PDCCH Decoder = Off
PDSCH Decoder = On

SSBConfigIndex =0, CarrierIndex=0, Numerc
PeriodIndex =0, BlockIndex=00, Sym OV1

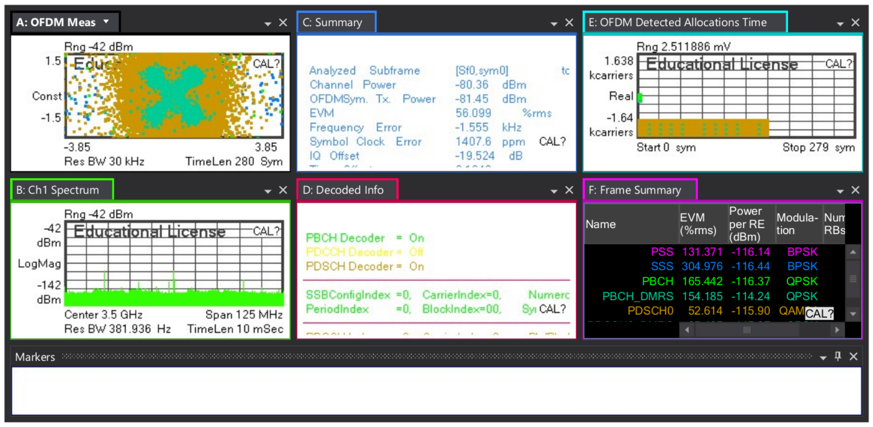
F: Frame Summary

Name	EVM (%rms)	Power per RE (dBm)	Modula-tion	Num RBs
PSS	1.761	-76.60	BPSK	
SSS	1.982	-76.63	BPSK	
PBCH	1.952	-76.64	QPSK	
PBCH_DMRS	1.963	-71.15	QPSK	
PDSCH0	2.618	-77.20	QAM6	

OV1

(b)

Figure 10: Transmission over cables at 16 dB transmitter gain- (a) VSA display (b) VSA screengrab displaying EVM



Res BW 30 kHz TimeLen 280 Sym

Analyzed Subframe	[Sf0,sym0]	
Channel Power	-80.36	dBm
OFDMSym. Tx. Power	-81.45	dBm
EVM	56.099	%rms
Frequency Error	-1.555	kHz
Symbol Clock Error	1407.6	ppm
IQ Offset	-19.524	dB

Res BW 381.936 Hz TimeLen 10 mSec

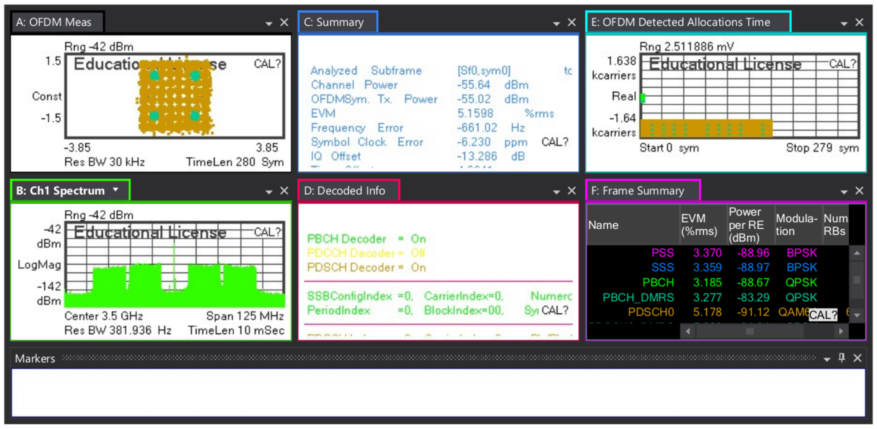
PBCH Decoder = On
PDCCH Decoder = Off
PDSCH Decoder = On

SSBConfigIndex =0, CarrierIndex=0,
PeriodIndex =0, BlockIndex=00,

Name	EVM (%rms)	Power per RE (dBm)	Modula-tion
PSS	131.371	-116.14	BPSK
SSS	304.976	-116.44	BPSK
PBCH	165.442	-116.37	QPSK
PBCH_DMRS	154.185	-114.24	QPSK
PDSCH0	52.614	-115.90	QAM

(a)

(b)



Res BW 30 kHz TimeLen 280 Sym

Analyzed Subframe	[Sf0,sym0]	
Channel Power	-55.64	dBm
OFDMSym. Tx. Power	-55.02	dBm
EVM	5.1598	%rms
Frequency Error	-661.02	Hz
Symbol Clock Error	-6.230	ppm
IQ Offset	-13.286	dB

Res BW 381.936 Hz TimeLen 10 mSec

PBCH Decoder = On
PDCCH Decoder = Off
PDSCH Decoder = On

SSBConfigIndex =0, CarrierIndex=0,
PeriodIndex =0, BlockIndex=00,

Name	EVM (%rms)	Power per RE (dBm)	Modula-tion
PSS	3.370	-88.96	BPSK
SSS	3.359	-88.97	BPSK
PBCH	3.185	-88.67	QPSK
PBCH_DMRS	3.277	-83.29	QPSK
PDSCH0	5.178	-91.12	QAM

(c)

Figure 11: Wireless transmission (a) VSA screengrab at 25 dB transmitter gain (b) VSA display at 57 dB (c) VSA screengrab at 57 dB

repeatedly. Data obtained either via wireless or wired transmission is passed on to the host computer by the USRP and then saved as a binary file which is converted to IQ data that can be processed using MATLAB scripts. These scripts first determine the slot boundaries. Samples corresponding to one frame (20 slots as SCS used is 30 kHz) or even one slot are then extracted and passed through a receiver script that plots the received constellation and calculates the BER.

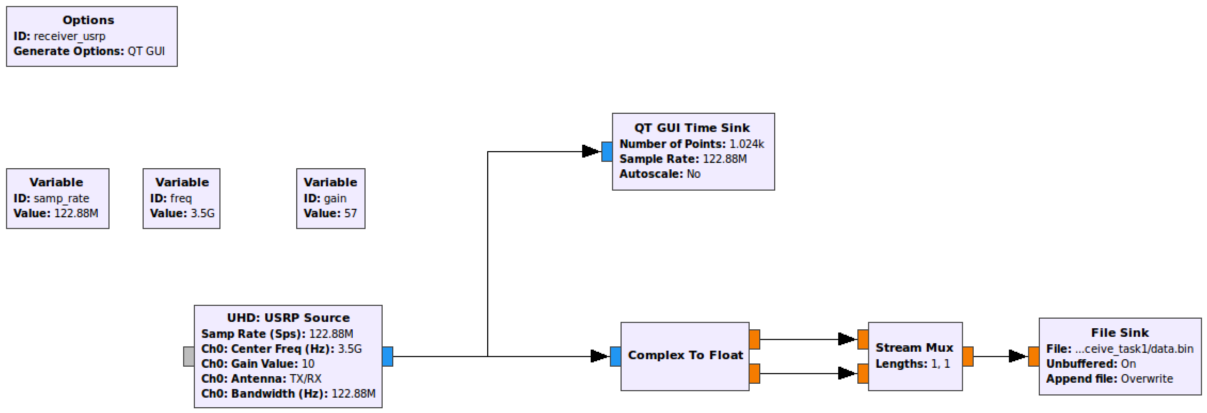


Figure 12: Receiver schematic in GNU Radio

3.2.2 Results and discussion

The power spectrum of the received data for one wired transmission is presented in Figure 13. This data corresponds to allocation of resources to four users. In fact, each user is allocated 50 resource blocks.

As discussed in Section 3.2.1, upon receiving the signal, the first task is to determine the slot boundaries so that the data corresponding to one slot can be passed through the receiver implemented in MATLAB to recover the transmitted data and measure BER. Pseudocode to determine the slot boundaries is given below

```
- Obtain complex RX_signal from saved binary file
- Correlation = zeros(1,length(RX_signal) - (288 + FFT_size))
- for i = 0:(length(Correlation)-1)
-   Correlation(i + 1) = Correlate(RX_signal(i + 1:288), RX_signal(i +
...FFT_size + 1:288))
- start = location of peak lasting (352-288) samples
- slot_data = RX_signal(start + (1:(122.88e6*0.5e-3)) - 1)
```

where 288 and 352 correspond to the CP lengths used for different symbols in a slot, 0.5e-3 corresponds to the duration of a slot and 122.88e6 is the sampling rate. This is

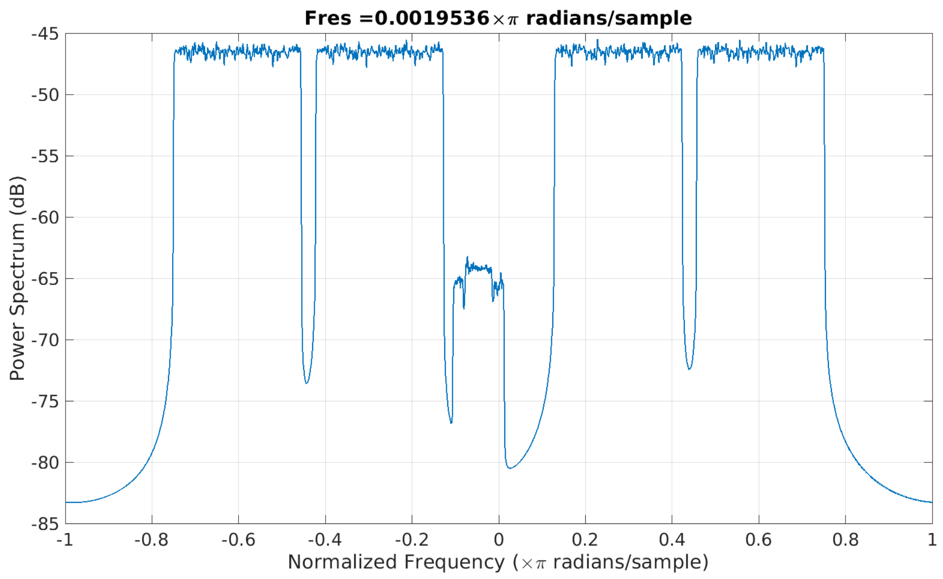


Figure 13: Power spectrum of received data

demonstrated theoretically for data which is transmitted through an AWGN channel with an SNR of 10 dB in Figures 14a and 14b.

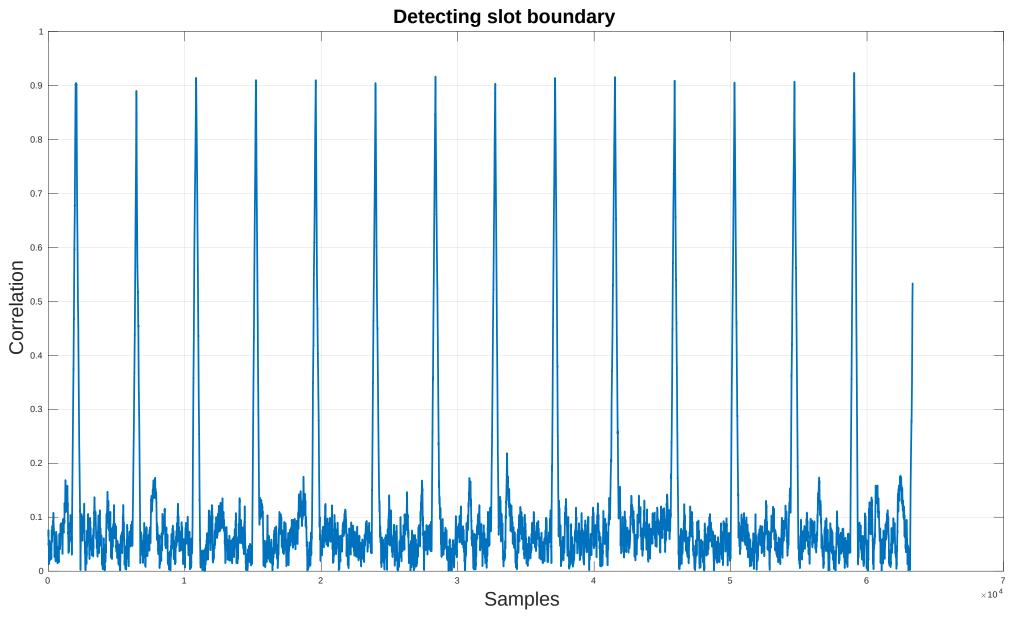
Even with data transmitted using the VSG via a wired connection to the USRP, the results are quite similar. However, on extracting the slot data and passing it through the receiver script, high BER and EVM are noted with the detected constellation being completely scrambled. This occurs irrespective of the transmitter and receiver gains. Therefore, it is hypothesized that this is due to frequency offset. Consequently, to mitigate this, synchronising both the USRP and VSG using the USRP's 10 MHz clock reference was attempted but proved ineffective.

Therefore, in the receiver implementation, the following work remains to be completed:

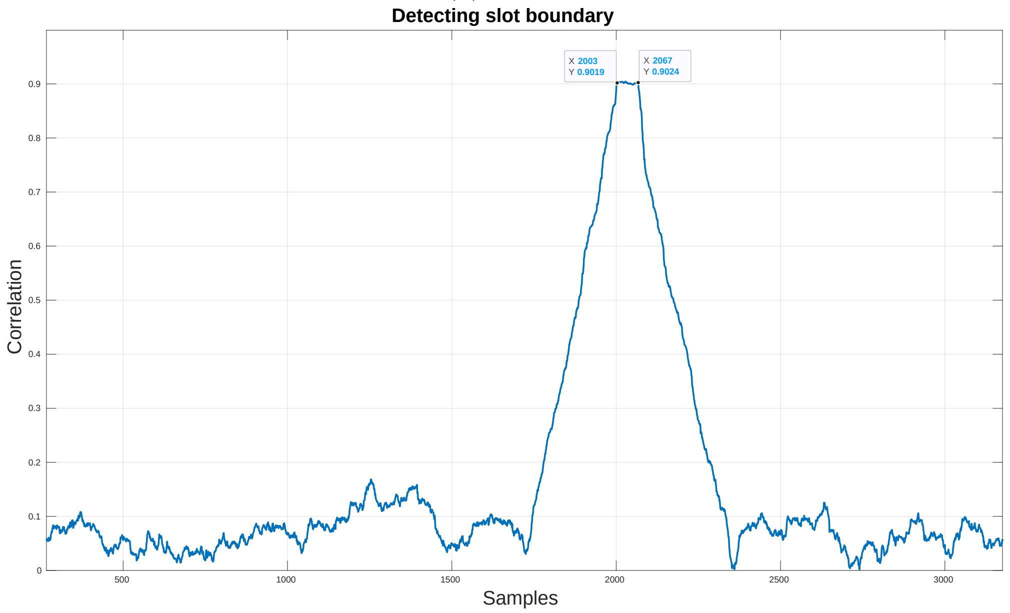
- Identifying if frequency offset does indeed exist and is the cause of the high BER. If so, estimating and correcting for the same. One method to do this is to transmit sinusoidal signals and detect the offset produced.
- Transmit over a wireless channel. Thus far, transmission has only been done over cables. Consequently, transmitting using antennas is the obvious next step.

3.3 Integration

Once the design of the receiver is complete, the transmitter and receiver can be integrated using time division to design a system that can both transmit and receive.



(a)



(b)

Figure 14: Using correlation to detect slot boundaries for data transmitted through an AWGN channel (a) 14 peaks- one for each OFDM symbol in a slot (b) Zoomed in peak for the zeroth OFDM symbol with a peak duration of 64 samples

4 Conclusion

This document looks at the operation of the USRP N321 available at the Indigenous 5G Testbed at IIT Madras and the design of a transmitter and receiver using the same. While the transmitter is about complete, both the transmitter and receiver implementations have issues that need to be resolved before full functionality. This document also substitutes theoretical information for actual data in one case (Figure 14- owing to constraints that deprive the author of access to the original data available at the testbed at the time of writing) with the guarantee that the results obtained for the actual data bear no significant dissimilarities. Some future work to complete the transmitter and receiver design is also highlighted.

References

[1] Wikipedia, "Half duplex communication." https://en.wikipedia.org/wiki/Duplex_(telecommunications)Half_duplex, 2020.

[2] Ettus Research, "USRP N321 Datasheet." https://www.ettus.com/wp-content/uploads/2019/03/USRP-N321-Datasheet-5.pdf, 2020.

[3] Ettus Research, "USRP N321 Software Setup." https://kb.ettus.com/Building_and_Installing_the_USRP_Open-Source_Toolchain_(UHD_and_GNU_Radio)_on_Linux, 2020.

[4] Ettus Research, "USRP N321 Setup Guide." https://kb.ettus.com/USRP_N300/N310/N320/N321_Getting_Started_Guide, 2020.

[5] Wikipedia, "EVM." https://en.wikipedia.org/wiki/Error_vector_magnitude, 2020.