Table of Contents

1 B205mini Getting Started Guides....1
1.1....1
1.2....1
1.3....1
1.4....1
1.5....1
1.6....1
1.7....1
1.8....2
1.9....2
1.10....2
1.11....2

2 Ettus USRP E300 Embedded Family Getting Started Guides....3
2.1....3
2.2....3
2.3....4
2.4....4
2.5....4
2.6....4
2.7....5
2.8....5
2.9....5
2.10....7
2.11....7
2.12....7
2.13....7
2.14....8
2.15....8
2.16....8
2.17....9
2.18....9
2.19....9

3 E320 Getting Started Guide....10
3.1....10
3.2....10
3.3....10
3.4....10
3.5....11
3.6....11
3.7....12
3.8....13
3.9....14
3.10....15
3.11....16
3.12....19
3.13....21
3.14....21
3.15....21
3.16....21
3.17....21

4 N210 Getting Started Guides....22
4.1....22
4.2....22
4.3....22
4.4....22
4.5....22
4.6....22
4.7....22
4.8....23
4.9....23
4.10....23
4.11....23

5 N321 Getting Started Guide....24
5.1....24
5.2....25
5.3....25
5.4....25
5.5....25
5.6....25
5.7....27
5.8....28
5.9....29
5.10....30
5.11....31
5.12....40
5.13....40
5.14....41
5.15....41
5.16....41

Table of Contents

6 X310 Getting Started Guides **42**
6.1 42
6.2 42
6.3 42
6.4 42
6.5 42
6.6 42
6.7 43
6.8 43
6.9 43
6.10 43
6.11 44
6.12 44
6.13 44
6.14 44

7 USRP-2974 Getting Started Guide **45**
7.1 45
7.2 45
7.3 45
7.4 45
7.5 45
7.6 46
7.7 46
7.8 46
7.9 46
7.10 46
7.11 46
7.12 47
7.13 47
7.14 47
7.15 47

8 X440 Getting Started Guide **48**
8.1 48
8.2 48
8.3 48
8.4 48
8.5 48
8.6 49
8.7 49
8.8 49
8.9 49
8.10 50
8.11 50
8.12 50
8.13 50
8.14 51
8.15 52
8.16 52
8.17 52
8.18 52
8.19 53
8.20 53
8.21 53
8.22 53
8.23 53
8.24 54
8.25 54
8.26 54
8.27 55
8.28 55
8.29 55
8.30 55

9 BasicRX Getting Started Guides **56**
9.1 56
9.2 56
9.3 56
9.4 56
9.5 56
9.6 57
9.7 57
9.8 57

10 CBX Getting Started Guides **58**
10.1 58
10.2 58
10.3 58
10.4 58
10.5 58
10.6 59
10.7 59
10.8 59

Table of Contents

11 LFRX Getting Started Guides **60**
11.1 60
11.2 60
11.3 60
11.4 60
11.5 60
11.6 61
11.7 61
11.8 61

12 SBX Getting Started Guides **62**
12.1 62
12.2 62
12.3 62
12.4 62
12.5 62
12.6 63
12.7 63
12.8 63

13 TwinRX Getting Started Guides **64**
13.1 64
13.2 64
13.3 64
13.4 64
13.5 64
13.6 64
13.7 65
13.8 65
13.9 65
13.10 65
13.11 65

14 UBX Getting Started Guides **66**
14.1 66
14.2 66
14.3 66
14.4 66
14.5 66
14.6 67
14.7 67
14.8 67

15 WBX Getting Started Guides **68**
15.1 68
15.2 68
15.3 68
15.4 68
15.5 68
15.6 69
15.7 69
15.8 69

16 Getting Started with RFNoC in UHD 4.0 ..... **70**
16.1 70
16.2 70
16.3 70
16.4 70
16.5 70
16.6 70
16.7 72
16.8 72
16.9 74
16.10 77
16.11 78
16.12 80

17 RFNoC 4 Migration Guide **81**

18 ..... **82**

19 ..... **83**
19.1 83

20 ..... **84**
20.1 84

21 ..... **86**

22 ..... **87**
22.1 87

23 ..... **89**
23.1 89

Table of Contents

24 ..... **91**
24.1 91

25 Getting Started with RFNoC Development **92**
25.1 92
25.2 92
25.3 92
25.4 92
25.5 92
25.6 92
25.7 97
25.8 99
25.9 111
25.10 113
25.11 115
25.12 115
25.13 116

26 Live SDR Environment Getting Started Guides **117**

27 OctoClock CDA-2990 Getting Started Guides **118**
27.1 118
27.2 118
27.3 118
27.4 118
27.5 118
27.6 118

28 Using Ethernet-Based Synchronization on the USRP? N3xx Devices **119**
28.1 119
28.2 119
28.3 119
28.4 119
28.5 119
28.6 120
28.7 122
28.8 123

29 Getting Started with DPDK and UHD **124**
29.1 124
29.2 124
29.3 124
29.4 124
29.5 124
29.6 124
29.7 124
29.8 125
29.9 125
29.10 125
29.11 126
29.12 126
29.13 127
29.14 129

30 B205mini **130**
30.1 130
30.2 130
30.3 131
30.4 131
30.5 132
30.6 132
30.7 133
30.8 133
30.9 133
30.10 133
30.11 134
30.12 135
30.13 136
30.14 136
30.15 136
30.16 136

31 Ettus USRP E300 Embedded Family Hardware Resources **138**
31.1 138
31.2 138
31.3 139
31.4 140
31.5 140
31.6 140
31.7 141
31.8 142
31.9 142
31.10 143
31.11 143
31.12 145

Table of Contents

31 Ettus USRP E300 Embedded Family Hardware Resources
31.13 ... 146
31.14 ... 146
31.15 ... 147
31.16 ... 147
31.17 ... 148
31.18 ... 148

32 E320 ... 149
32.1 ... 149
32.2 ... 149
32.3 ... 149
32.4 ... 149
32.5 ... 149
32.6 ... 150
32.7 ... 150
32.8 ... 150
32.9 ... 151
32.10 ... 151
32.11 ... 153
32.12 ... 153
32.13 ... 153
32.14 ... 154

33 N210 ... 155
33.1 ... 155
33.2 ... 155
33.3 ... 155
33.4 ... 155
33.5 ... 155
33.6 ... 156
33.7 ... 156
33.8 ... 156
33.9 ... 156
33.10 ... 156
33.11 ... 157
33.12 ... 157
33.13 ... 157
33.14 ... 157
33.15 ... 157

34 N310 ... 158
34.1 ... 158
34.2 ... 158
34.3 ... 158
34.4 ... 159
34.5 ... 160
34.6 ... 160
34.7 ... 160
34.8 ... 161
34.9 ... 161
34.10 ... 161
34.11 ... 162
34.12 ... 162
34.13 ... 170
34.14 ... 170
34.15 ... 170
34.16 ... 170
34.17 ... 171
34.18 ... 171
34.19 ... 171
34.20 ... 171
34.21 ... 171

35 N321 ... 172
35.1 ... 172
35.2 ... 172
35.3 ... 172
35.4 ... 173
35.5 ... 174
35.6 ... 174
35.7 ... 175
35.8 ... 175
35.9 ... 175
35.10 ... 175
35.11 ... 176
35.12 ... 176
35.13 ... 185
35.14 ... 185
35.15 ... 185
35.16 ... 185
35.17 ... 185
35.18 ... 186
35.19 ... 186
35.20 ... 186

Table of Contents

36 X310 ..... **187**
36.1 187
36.2 187
36.3 187
36.4 187
36.5 188
36.6 188
36.7 188
36.8 188
36.9 188
36.10 189
36.11 189
36.12 190
36.13 190
36.14 194
36.15 194
36.16 194
36.17 194
36.18 195
36.19 195
36.20 196
36.21 196
36.22 197
36.23 197
36.24 197
36.25 198
36.26 198
36.27 199
36.28 199

37 USRP-2974 ..... **201**
37.1 201
37.2 201
37.3 201
37.4 201
37.5 202
37.6 202
37.7 202
37.8 202
37.9 202
37.10 204
37.11 204
37.12 204
37.13 208
37.14 208
37.15 208
37.16 209
37.17 209
37.18 210
37.19 210
37.20 211

38 X410 ..... **213**
38.1 213
38.2 213
38.3 213
38.4 214
38.5 214
38.6 214
38.7 214
38.8 215
38.9 215
38.10 216
38.11 216
38.12 216
38.13 216

39 X440 ..... **217**
39.1 217
39.2 217
39.3 217
39.4 218
39.5 218
39.6 218
39.7 219
39.8 219
39.9 219
39.10 220
39.11 220
39.12 220
39.13 220

40 BasicRX **221**
40.1 221
40.2 221

Table of Contents

40 BasicRX
40.3 ... 221
40.4 ... 222
40.5 ... 222
40.6 ... 222
40.7 ... 222
40.8 ... 222
40.9 ... 222
40.10 ... 222
40.11 ... 222
40.12 ... 223
40.13 ... 223

41 CBX ... 224
41.1 ... 224
41.2 ... 224
41.3 ... 224
41.4 ... 224
41.5 ... 225
41.6 ... 225
41.7 ... 225
41.8 ... 225
41.9 ... 225
41.10 ... 225
41.11 ... 226
41.12 ... 226
41.13 ... 226
41.14 ... 226
41.15 ... 226
41.16 ... 226

42 LFRX ... 227
42.1 ... 227
42.2 ... 227
42.3 ... 227
42.4 ... 228
42.5 ... 228
42.6 ... 228
42.7 ... 228
42.8 ... 228
42.9 ... 228
42.10 ... 228
42.11 ... 228
42.12 ... 229

43 SBX ... 230
43.1 ... 230
43.2 ... 230
43.3 ... 230
43.4 ... 230
43.5 ... 231
43.6 ... 231
43.7 ... 231
43.8 ... 231
43.9 ... 231
43.10 ... 231
43.11 ... 232
43.12 ... 232
43.13 ... 232
43.14 ... 232
43.15 ... 232
43.16 ... 232

44 TwinRX ... 233
44.1 ... 233
44.2 ... 233
44.3 ... 234
44.4 ... 235
44.5 ... 235
44.6 ... 235
44.7 ... 236
44.8 ... 236
44.9 ... 236
44.10 ... 236
44.11 ... 236
44.12 ... 237
44.13 ... 238
44.14 ... 238
44.15 ... 238
44.16 ... 238

45 UBX ... 240
45.1 ... 240
45.2 ... 240
45.3 ... 240

Table of Contents

45 UBX
45.4 ... 240
45.5 ... 241
45.6 ... 241
45.7 ... 241
45.8 ... 241
45.9 ... 242
45.10 ... 242
45.11 ... 242
45.12 ... 242
45.13 ... 242
45.14 ... 242
45.15 ... 243
45.16 ... 243
45.17 ... 243

46 WBX ... 244
46.1 ... 244
46.2 ... 244
46.3 ... 244
46.4 ... 244
46.5 ... 245
46.6 ... 245
46.7 ... 245
46.8 ... 245
46.9 ... 245
46.10 ... 245
46.11 ... 246
46.12 ... 246
46.13 ... 246
46.14 ... 246
46.15 ... 246
46.16 ... 246

47 OctoClock CDA-2990 ... 247
47.1 ... 247
47.2 ... 247
47.3 ... 247
47.4 ... 248
47.5 ... 248
47.6 ... 248
47.7 ... 248
47.8 ... 248
47.9 ... 248
47.10 ... 248
47.11 ... 249
47.12 ... 249
47.13 ... 249
47.14 ... 249

48 GPSDO ... 251
48.1 ... 251
48.2 ... 251
48.3 ... 252
48.4 ... 253

49 Antennas ... 254
49.1 ... 254
49.2 ... 254
49.3 ... 254
49.4 ... 254
49.5 ... 254

50 UHD ... 255
50.1 ... 255
50.2 ... 256
50.3 ... 256
50.4 ... 257
50.5 ... 257
50.6 ... 258
50.7 ... 258
50.8 ... 259
50.9 ... 259
50.10 ... 261
50.11 ... 262
50.12 ... 263
50.13 ... 263
50.14 ... 263
50.15 ... 263
50.16 ... 263

51 UHD Python API ... 264
51.1 ... 264
51.2 ... 264
51.3 ... 264

Table of Contents

51 UHD Python API
51.4 ... 264
51.5 ... 264

52 Getting Started with RFNoC in UHD 4.0 ... 266
52.1 ... 266
52.2 ... 266
52.3 ... 266
52.4 ... 266
52.5 ... 266
52.6 ... 266
52.7 ... 268
52.8 ... 268
52.9 ... 270
52.10 ... 273
52.11 ... 274
52.12 ... 276

53 RFNoC ... 277

54 GNU Radio ... 278

55 LabVIEW ... 279

56 Simulink ... 280

57 OpenBTS ... 281
57.1 ... 281
57.2 ... 281

58 Eurecom OpenAirInterface (OAI) ... 282
58.1 ... 282
58.2 ... 282

59 srsUE ... 283
59.1 ... 283
59.2 ... 283

60 Gqrx ... 284

61 Fosphor ... 285

62 Multichannel RF Reference Architecture ... 286
62.1 ... 286

63 ... 287
63.1 ... 287
63.2 ... 289

64 ... 291
64.1 ... 292
64.2 ... 292
64.3 ... 299
64.4 ... 299
64.5 ... 300
64.6 ... 300

65 ... 301
65.1 ... 301
65.2 ... 301
65.3 ... 301
65.4 ... 301
65.5 ... 302
65.6 ... 302

66 ... 303
66.1 ... 303
66.2 ... 303
66.3 ... 303
66.4 ... 307

67 ... 309
67.1 ... 309
67.2 ... 309

68 ... 310
68.1 ... 310
68.2 ... 310

69 ... 311

70 Workshop Tutorial ... 312
70.1 ... 312

Table of Contents

71 Suggested Reading....313
71.1....313
71.2....313
71.3....314

72 Suggested Videos....315
72.1....315
72.2....315

73 CGRAN....316

74 SDR Events....317

75 GNU Radio Conference....318
75.1....318
75.2....318
75.3....318
75.4....318
75.5....318

76 NEWSDR....319
76.1....319
76.2....319
76.3....319
76.4....319
76.5....319

77 FOSDEM....320
77.1....320
77.2....320
77.3....320
77.4....320
77.5....320
77.6....320
77.7....320

78 Cyberspectrum....321
78.1....321

79 Email....327

80 Mailing Lists....328
80.1....328

81 Matrix....329
81.1....329

82 SDR Boston Slack....330
82.1....330

83 StackExchange....331
83.1....331

84 NI SRM....332
84.1....332

85 Technical FAQ....333
85.1....333
85.2....333
85.3....333
85.4....333
85.5....333

86 Licensing FAQ....334
86.1....334

87 USRP1....335
87.1....335
87.2....335
87.3....335
87.4....335
87.5....335
87.6....335
87.7....335

88 USRP2....336
88.1....336
88.2....336
88.3....336
88.4....336
88.5....336
88.6....336
88.7....337
88.8....337

Table of Contents

89 E110 ..... **338**
89.1 338
89.2 338
89.3 338
89.4 338
89.5 338
89.6 339
89.7 339
89.8 339
89.9 339
89.10 339

90 B100 ..... **340**
90.1 340
90.2 340
90.3 340
90.4 340
90.5 340
90.6 341
90.7 341
90.8 341

91 DBSRX2 ..... **342**
91.1 342
91.2 342
91.3 342
91.4 342
91.5 342
91.6 342

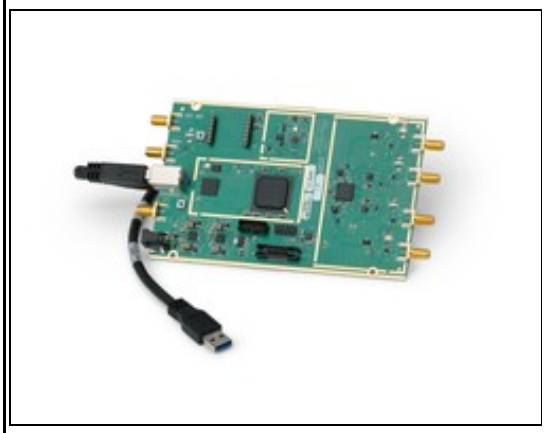
92 TVRX2 ..... **343**
92.1 343
92.2 343
92.3 343
92.4 343
92.5 343
92.6 343
92.7 343

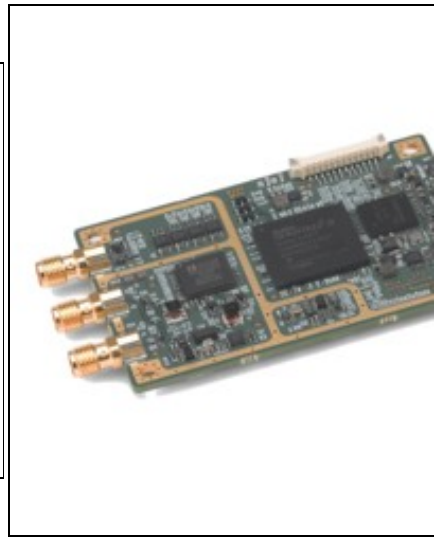
93 XCVR2450 ..... **344**
93.1 344
93.2 344
93.3 344
93.4 344
93.5 344
93.6 344
93.7 345
93.8 345
93.9 345

1 B205mini Getting Started Guides

1.1

- USRP B200 / B210 / B200mini / B205mini
- USB 3.0 Cable
- Universal power supply (B210 only)





1.2

Make sure that your kit contains all the items listed above. If any items are missing, please contact your sales agent or Ettus Research Technical support immediately.

1.3

- A host computer with an available USB 2.0 or 3.0 port

1.4

All Ettus Research products are individually tested before shipment. The USRP? is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP? can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

1.5

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on your host computer. A step-by-step guide for doing this is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows Application Notes. Release 3.8.4 or later of the USRP Hardware Driver, UHD, is required. It is recommended to use the latest stable version of UHD that is available.

If you have a USB stick with the Live SDR Environment installed on it, then you may boot your host computer from that. The LiveUSB SDR Environment does not require anything to be installed on your host computer, and contains a Linux-based environment with the UHD software and the GNU Radio framework already installed. More information about the Live SDR Environment is available at the Live SDR Environment Getting Started Guides page.

1.6

The included USB 3.0 cable provides power and data connectivity for the USRP Bus Series. The host-side of the cable must be plugged into either a USB 2.0 or 3.0 port. Note that the USB 2.0 link provides less bandwidth than the USB 3.0 link. Also note that an external DC power supply must be connected if using a GPSDO (B200/B210 only).

1.7

Once the software tools are installed on the host computer, or using the Live SDR Environment, verify the correct operation of the USRP by running the utility programs on the host computer. More information is available at the Verifying the Operation of the USRP Using UHD and GNU Radio Application Note.

1.8

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

1.9

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

1.10

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

1.11

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

2 Ettus USRP E300 Embedded Family Getting Started Guides

2.1

2.1.1

- E310 USRP
- Power supply
- 2x SMB-to-SMA adapter
- 1 Gigabit Ethernet cable
- USB2-to-microUSB cable
- Imaged microSD card
- Getting started guide

2.1.2

- E312 USRP
- Power supply
- 2x SMB-to-SMA adapter
- 1 Gigabit Ethernet cable
- USB2-to-microUSB cable
- Imaged microSD card
- Getting started guide

2.1.3

- Protective caps for input ports
- Surface mounting accessory
- Pole mounting accessory
- End conduit interface for USB devices
- Waterproof sleeve for DC power connector
- Waterproof sleeve for PoE (RJ45) connector
- Torx T-20 key
- Mounting accessory assembly guide
- Imaged microSD card

The USRP E313 is a fully assembled device that includes an USRP E310.

2.2

Make sure that your kit contains all the items listed above. If any items are missing, please contact your sales agent or Ettus Research Technical support immediately.

2.3

- A host computer with an available USB 2.0 or 3.0 port

2.4

All Ettus Research products are individually tested before shipment. The USRP? is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP? can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than 0 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

2.5

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on your host computer. A step-by-step guide for doing this is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows Application Notes. See the Hardware Specifications section of the USRP Embedded Series Hardware Resources for additional details on which version of the USRP Hardware Driver, UHD, is required. It is recommended to use the latest stable version of UHD that is available.

If you have a USB stick with the Live SDR Environment installed on it, then you may boot your host computer from that. The LiveUSB SDR Environment does not require anything to be installed on your host computer, and contains a Linux-based environment with the UHD software and the GNU Radio framework already installed. More information about the Live SDR Environment is available at the Live SDR Environment Getting Started Guides page.

2.6

2.6.1

With older USRP E31x devices running Firmware version 1, connecting the AC power supply to the device will cause the unit to turn on and boot–. By default with Firmware 2.0 and newer the device no longer turns on when AC power is plugged in. To determine the firmware version, once the device is fully booted, login to it and execute

```
$ dmesg | grep -i "firmware version"
```

If this is the first time powering on a USRP E312 (with battery), allow the battery to fully charge before disconnecting the AC power source.

Once the device has completed the boot process, you are ready to start using the device over your preferred method of connectivity (Serial Console, Network, or USB peripherals)!

2.6.2

You can power the device on and off by pressing the power button. To power the device off, hold the power button down until the button?s LED turns off -- this will take a couple of seconds, and then another 15 seconds for the device to fully shut down. To turn the device on, hold down the power button until the LED turns on.

To avoid damaging the file system and causing any corruption, do not turn the device off with the power button without first shutting down the system. Use this command to cleanly and properly shut the system down:

```
$ shutdown --h now
```

2.6.3

The USRP E31x can be configured to power on and boot automatically when power is applied. If the firmware is older than 2.0 then it will *probably* need to be updated for autoboot to work reliably; email support@ettus.com for more information.

To control autoboot on the USRP E31x, first determine the version of UHD, for example by running

```
$ uhd_config_info --version
```

on the device. The UHD version determines the filesystem location where the `autoboot` file is located.

- For UHD 4 and newer

To enable autoboot:

```
$ echo 1 > /sys/devices/soc0/fpga-full/fpga-full:pmu/autoboot
```

To disable autoboot:

```
$ echo 0 > /sys/devices/soc0/fpga-full/fpga-full:pmu/autoboot
```

- For UHD 3.15 and older:

To enable autoboot:

```
$ echo 1 > /sys/devices/axi_pmu.3/autoboot
```

To disable autoboot:

```
$ echo 0 > /sys/devices/axi_pmu.3/autoboot
```

Settings take place immediately; no reboot is required.

2.6.4

The default user is `root` and the password is empty (no password).

It is recommended to update the `root` password, which can be done with the command `passwd`:

Example Output:

```
root@ni-e3x0-SERIAL:~# passwd
Changing password for root
New password:
Re-enter new password:
passwd: password changed.
```

2.7

The easiest way to first communicate with your E31x device is by using the USB Serial Console. Connect a micro−USB cable to the Serial Console port on the E31x and connect the other end to a PC. The console will appear as an ?FTDI Serial Device? – thus, it will likely appear as a ttyUSB device in Linux or a COM port on Windows. In Windows, you will need to edit the properties of the device in ?Device Manager? and ?Enable VCP?. On the PC, open a serial terminal to the E31x using the following parameters: B?aud Rate:?115200, D?ata:?8−bit, P?arity: None, S?top:?1−bit, F?low Control:?None.

On Linux, the following command will typically handle the serial connection:

```
sudo screen /dev/ttyUSB0 115200
```

You may have to change the device name.

For additional information about using the serial console and instructions for communicating with the device over other methods (such as connecting with SSH over the network or using and LCD screen, keyboard, and mouse), please refer to the UHD Manual online: https://files.ettus.com/manual/page_usrp_e3xx.html

2.8

By default, the E31x device will run a DHCP client on its 1 Gigabit Ethernet port. Assuming your network resolves hostnames (depends on your routers / switches), if you connect the device to your network, you should see it appear with the hostname e?ttus−e300.?You can then access the device over SSH.

If the hostname does not resolve, you can discover the IP address by logging into the device over the serial connection, or checking your network?s DHCP tables.

Once you have logged in to the device, you can reconfigure the network settings (e.g., you could configure it for a static IP address, if you wish).

2.9

Before operating the device, it is? ?strongly? recommended to update to the latest version of the Embedded Linux file system. If you are operating the device in Network Mode, the version of UHD running on the host machine and E310 USRP must match.

There are two ways to update the file system for the E310 USRP:

1. Mender, which is available starting with UHD 4.0.0.0 release only. If you are using UHD 3.15 or prior you'll need to update the microSD card to UHD4 before being able to use Mender to do updates.

2. Physically remove microSD card from device and write a new file system to the microSD card.

NOTE: File System Partition Layout

The SD Card is divided into four partitions. There are two root file system partitions, a "boot" partition and a "data" partition.

Any data you would like to preserve through Mender updates should be saved to the "data" partition, which is mounted at `/data`.

2.9.1

Mender is third-party software that enables remote updating of the root file system without physically accessing the device (see also the Mender website https://mender.io/). Mender can be executed locally on the device, or a Mender server can be set up which can be used to remotely update an arbitrary number of USRP devices. Users can host their own local Mender server, or use servers hosted by Mender as a paid service; contact Mender for more information.

2.9.1.1

When updating the file system using Mender, the tool will overwrite the root file system partition that is not currently mounted. Any data stored in the root partitions will be permanently lost with a Mender update.

After updating a partition with Mender, it will reboot into the newly updated partition. Only if the update is confirmed by the user, the update will be made permanent. This means that if an update fails, the device will be always able to reboot into the partition from which the update was originally launched, which presumably is in a working state. Another update can be launched now to correct the previous, failed update, until it works.

The USRP E31x release images come in two varieties, sg1 and sg1. The variety that you will need depends on the product number of your E31x, which is printed on the bottom of the device. You must use the appropriate files for your specific device. Incorrect files will not work, and will only boot as far as the U-Boot boot loader before stopping.

For the E310, the product number will be `156333X-01L`, where X is a letter from A to Z. For devices where X is A, B, C, D, use the `sg1` files. For devices where X is E or later, use the `sg3` files.

For the E312, the product number will be `140605X-01L`, where X is a letter from A to Z. All E312 USRPs use the `sg3` files.

To obtain the file system Mender image (these are files with a `.mender` suffix), run the following command on the host computer with Internet access:

```
$ sudo uhd_images_downloader -t mender -t e310 -t sg# --yes
```

where "sg#" is the correct file type as found above, with "#" being either 1 or 3. Example Output:

```
$ sudo uhd_images_downloader -t mender -t e310 -t sg3 --yes
[INFO] Using base URL: https://files.ettus.com/binaries/cache/
[INFO] Images destination: /usr/local/share/uhd/images
292014 kB / 292014 kB (100%) e3xx_e310_sg3_mender_default-v4.0.0.0.zip
[INFO] Images download complete.
```

NOTE: In the output of the command, the folder destination where the images are saved is printed out.

NOTE: Regardless of which file type is specified, the extracted mender file will have the same name: `usrp_e310_fs.mender`.

Next, you will need to copy this Mender file system image to the USRP E310. This can be done with the Linux utility `scp`.

Example code to execute:

```
$ scp /usr/local/share/uhd/images/usrp_e310_fs.mender root@192.168.1.51:~/.
```

Note: The path and IP may different for your configuration, the command above assumes you're using the default installation path of `/usr/local` and that the E310's IP is `192.168.1.51`.

After copying the Mender file system image to the E310, connect to the E310 using either the Serial Console, or via SSH to gain shell access.

On the E310, run `mender install /path/to/latest.mender` to update the file system:

```
root@ni-e310-serial:~# mender install /home/root/usrp_e310_fs.mender
```

Example Output:

```
root@ni-e310-serial:~# mender install /home/root/usrp_e310_fs.mender
INFO[0000] Start updating from local image file: [/home/root/usrp_e310_fs.mender]  module=rootfs
Installing update from the artifact of size 399640064
INFO[0000] opening device /dev/mmcblk0p3 for writing     module=block_device
INFO[0000] partition /dev/mmcblk0p3 size: 2046820352     module=block_device
................................   0% 1024 KiB
................................   0% 2048 KiB
................................   0% 3072 KiB
[truncated for readability]
................................  99% 389120 KiB
................................  99% 390144 KiB
................................ 100% 390273 KiB
INFO[0740] wrote 2046820352/2046820352 bytes of update to device /dev/mmcblk0p3  module=device
INFO[0744] Enabling partition with new image installed to be a boot candidate: 3  module=device
```

The artifact can also be stored on a remote server:

```
$ mender install <http://server.name/path/to/latest.mender>
```

This procedure will take a few minutes to complete. After mender has logged a successful update, reboot the device:

```
$ reboot
```

If the reboot worked, and the device seems functional, commit the changes so that the boot loader knows to permanently boot into this partition:

```
$ mender -commit
```

To identify the currently installed Mender artifact from the command line, the following file can be queried on the E310:

```
$ cat /etc/mender/artifact_info
```

If you are using a Mender server, the updates can be initiated from a web dashboard. From there, you can start the updates without having to log into the device, and you can update groups of USRPs with a few clicks in a web GUI. The dashboard can also be used to inspect the state of USRPs. This is a simple way to update groups of rack-mounted USRPs with custom file systems.

For more information on updating the file-system, refer to the UHD Manual?.

2.9.1.1.1

When updating an E31x USRP using mender, it is possible that the update will not apply. For example, if the E31x v3.15.0.0 bootloader is misconfigured then it will not boot into an upgraded mender image. There are 2 solutions to this:

A. Reimage SD card with full sdimg using dd or bmaptool

This is the recommended solution. Follow the steps to update using the sdimg. E31x v4.0.0.0 and later contains the bootloader fix to enable future mender updates.

B. Manually reconfigure bootloader

This solution requires some effort, but isn't too difficult. That said, because the SD card is easily accessible on most E31x this is not the recommended solution.

1) Connect to the E31x via serial

2) Boot the device and quickly enter "noautoboot" into the serial console. It can be helpful to have "noautoboot" copied to the clipboard. If completed successfully, you should have a prompt like this:

```
Automatic boot in 3s...
Enter 'noautoboot' to enter prompt without timeout
ni-e31x-uboot>
```

If you don't get this prompt, restart the device and try again.

3) Configure the bootargs to support mender updates

```
setenv bootargs 'root=${???????mender_kernel_root}??????? rw rootwait uio_pdrv_genirq.of_id=usrp-uio'
saveenv
```

4) Reboot device and apply mender image

NOTE: This solution may brick the E31x USRP if done incorrectly. To unbrick the USRP, follow the solution of overwriting the full SD card image.

2.9.2

The microSD card is accessible directly on the Board-only version of the E310 USRP. The E310 Full Enclosure version must be opened with the included Torx wrench.

NOTE: This method will overwrite all data saved on the microSD card, including any data saved to the `/data` partition.

Please see this application note for step-by-step instructions on writing the file system image to the microSD card.

2.10

2.10.1

The USRP E31x contains 2 channels, each represented on the front panel as `TRX-A / RX2-A` and `TRX-B / RX2-B`. Below is the `subdev` mapping of RF

2.10.1.1

- TRX-A / RX2-A = A:0
- TRX-B / RX2-B = B:0

2.10.1.2

- TRX-A / RX2-A = A:0
- TRX-B / RX2-B = A:1

Additional details of UHD Subdevice Specifications can be found here in the UHD Manual:
http://files.ettus.com/manual/page_configuration.html#config_subdev

2.11

The UHD driver includes several example programs, which may serve as test programs or the basis for your application program. These example programs are already installed on the E31x device, and the source code can be obtained from the UHD repository on GitHub at:
https://github.com/EttusResearch/uhd/tree/master/host/examples

2.12

You can quickly verify the operation of your USRP E31x by running the `rx_ascii_art_dft` UHD example program. The `rx_ascii_art_dft` utility is a simple console –based, real–time FFT display tool. It is not graphical in nature, so it can be easily run over an SSH connection within a terminal window, and does not need any graphical capability, such as X Windows, to be installed. It can also be run over a serial console connection, although this is not recommended, as the formatting may not render correctly.

You can run a simple test of the E31x device by connecting an antenna and observing the spectrum of a commercial FM radio station in real–time. Please follow the steps listed below.

1. Attach an antenna to the RX2–A antenna port of the E31x.

2. Log into the E31x from an external host computer over Ethernet using an SSH client.

3. At a terminal prompt running on the E31x, run:

```
/usr/lib/uhd/examples/rx_ascii_art_dft ----freq 88.1e6 ----rate 400e3 ----gain 30 ----ref--lvl --30
```

4. Modify the command–line argument "`freq`" ?above to specify a tuning frequency for a strong local FM radio station.

5. You should see a real–time FFT display of 400 KHz of spectrum, centered at the specified tuning frequency.

6. Type "`Q`" or `Ctrl--C` to stop the program and to return to the Linux command line.

7. You can adjust the size of your terminal window and then re–run the command to enlarge or shrink the FFT display.

8. You can run with the "?––help"?option to see a description of all available command–line options.

Additional information is available at the Verifying the Operation of the USRP Using UHD and GNU Radio Application Note.

2.13

The USRP E312 is equipped with an integrated 3.7V, 3200mAh lithium–ion battery cell. After unboxing the USRP E312 , plug in the power adapter to an AC power source and fully charge the battery. This process with take approximately 2 hours. Do not leave the USRP E312 unit plugged in for more than 24 hours.

The status LED in the power button indicates the power and charge status of the battery:

Off: Indicates device is off and not charging.

- Slow Blinking Green: Indicates device is off and charging.
- Fast Blinking Green: Indicates device is on and charging.
- Solid Green: Indicates device is on and not charging (Battery is finished charging).
- Solid Orange: Indicates device is on and discharging.
- Fast Blinking Orange: Indicates device is on, discharging, and charge is below 10% charge.

- Fast Blinking Red: Indicates an error code:

1. Low Voltage Error
2. Regulator Low Voltage Error
3. FPGA Power Error
4. DRAM Power Error
5. 1.8V Power Rail Error
6. 3.3V Power Rail Error
7. Daughterboard / TX Power Error
8. Charger Error
9. Charger Temperature Error
10. Battery Low Error
11. Fuel Gauge Temperature Error
12. Global (Enclosure) Temperature Error

The battery life of the USRP E312 in idle mode is approximately 5 1/2 hours. The battery will enable the USRP E312 to operate for approximately 2 hours 20 minutes, when transmitting and receiving on both channels (2x2 MIMO), with maximum gain settings, at 5 GHz center frequency, and 1 MS/s sample rate. When the power button status LED is in the ?Fast Blinking Orange? mode, plug the USRP E312 into an AC power source as soon as possible to recharge the battery.

If the power button status LED indicates a ?Low Voltage Error? (codes 1, 2, 3, 4, 5, 6, 7) or a ?Battery Low Error? (code 10), plug the USRP E312 into an AC power source as soon as possible to recharge the battery.

When the power button status LED indicates at ?Temperature Error? or ?Charger Error? (codes 8, 9, 11, or 12), power off the USRP E312 unit and allow it to cool down to room temperature. Then, plug in the USRP E312 to and AC power source and fully charge the battery.

If error codes persist after cooling down and/or recharging the USRP E312, please contact support@ettus.com.

You can purchase a replacement battery for the E312 at https://www.ettus.com/product/details/E312-battery.

An Application Note covering the replacement of the E312 battery can be found at USRP E312 Battery Replacement Instructions.

2.14

In order for the battery gauge to give a usable indication of remaining charge it needs to be calibrated. The procedure for calibration is as follows:

1. Completely charge the battery.
2. Type: `?echo 3200000 >/sys/class/power_supply/BAT/charge_now`
3. Unplug AC power.
4. Replug AC power, and wait until charging completes.

2.15

To ensure proper use of the battery, please read the the battery specification sheet. This document is available at: Media:34118 datasheet.pdf

Because batteries utilize a chemical reaction, battery performance will deteriorate over time even if stored for a long period of time without being used. In addition, if the various usage conditions such as charge, discharge, ambient temperature, etc. are not maintained within the specified ranges, the life expectancy of the battery may be shortened or the device in which the battery is used may be damaged by electrolyte leakage.

2.15.1

- Do not expose the battery to flame or dispose of it in a fire.
- Do not put the battery in a charger or equipment with the wrong terminals connected.
- Do not short circuit the battery.
- Avoid excessive physical shock or vibration.
- Do not disassemble or deform the battery.
- Do not immerse in water.
- Do not use the battery mixed with other different make, type, or model batteries.
- Keep out of the reach of children.
- Do not use the battery if it appears damaged.

2.15.2

- Always charge the battery while it is installed in the USRP E312 and only use the DC power supply provided in the USRP E312 kit.
- Do not leave the battery charging for longer than 24 hours.
- Never use a modified or damaged USRP E312 DC power supply to charge the battery.

2.15.3

- Store the battery in a cool, dry, and well–-ventilated area.

2.15.4

- Regulations vary for different countries. Dispose of the battery in accordance with local regulations.

2.16

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

2.17

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

2.18

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

2.19

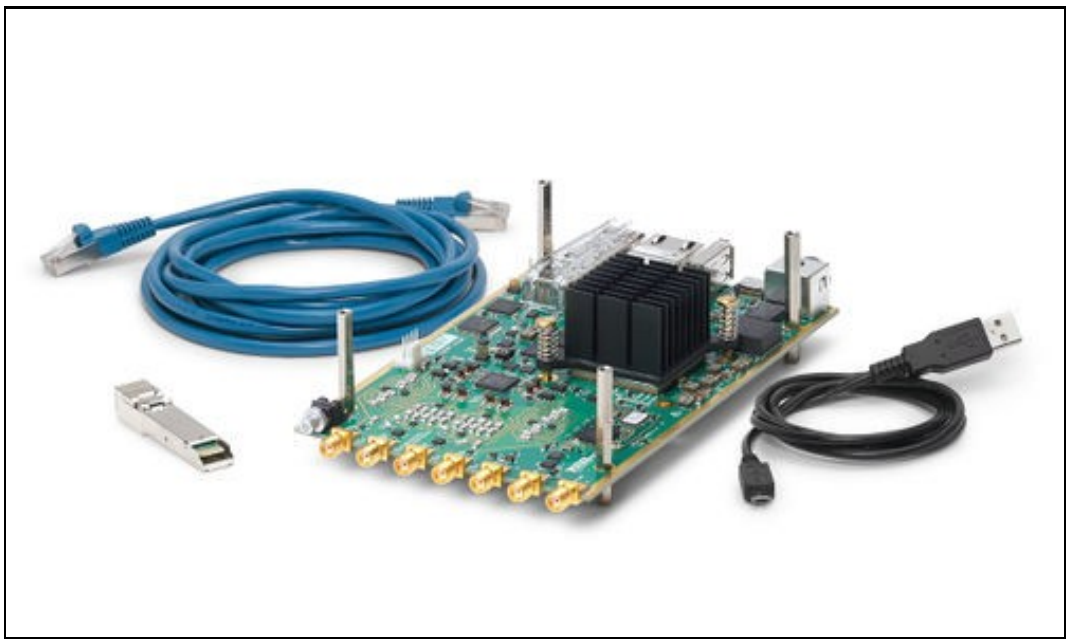
Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

3 E320 Getting Started Guide

3.1

3.1.1

- USRP E320
- Power connector (assembly required)
- 4 M3x0.5, M3x5 Standoffs
- 1 Gb Ethernet Cat-5e Cable (3m)
- USB-A to Micro USB-B Cable (1m)
- 1 Gb SFP+ to RJ45 Adapter
- Getting Started Guide
- Ettus Research Sticker



3.1.2

- USRP E320 in enclosure
- DC Power Supply (12V, 7A)
- 1 Gb Ethernet Cat-5e Cable (3m)
- USB-A to Micro USB-B Cable (1m)
- 1 Gb SFP+ to RJ45 Adapter
- Getting Started Guide
- Ettus Research Sticker
- T8 Torx Wrench

3.2

Ensure that your kit contains all the items listed above. If any items are missing, please contact sales@ettus.com? immediately.

3.3

- For Network Mode: A host computer with an 1 or 10 Gb Ethernet interface. If operating with the 10 Gb Ethernet interface, the "XG" FPGA image must be loaded before the SFP+ port will operate at 10 Gb speeds. Optionally a second 1 Gb Ethernet interface can be used to connect to the onboard ARM CPU for remote management.

- For Embedded Mode: A host computer is only required for initial device configuration, remote control and management, or data visualization. The host computer can connect to the RJ45 1 Gb port or Serial Console port to remotely access the Open Embedded Linux operating system running on the ARM CPU. Once configured, the USRP E320 can operate as a stand-alone device without a connection to a remote host computer.

- For Board-only Version: A third-party 10-14V/3A power supply, which requires assembly with the power connect components included in the kit. An assembled power supply can be purchased here: https://www.ettus.com/product/details/12V-PWR

3.4

All Ettus Research products are individually tested before shipment. The USRP is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP can cause the device to become non-functional. Take the following precautions to prevent damage to the unit.

- Never allow anything especially metal objects to touch the board while it is powered on.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water or condensing moisture to come into contact with the device.
- Always use caution with FPGA, firmware, or software modifications.
- Never touch the circuit board or heatsink while the device is powered on.
- All connections should be made/removed while is device is powered off.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

3.5

To use your Universal Software Radio Peripheral (USRP?), you must have software tools correctly installed and configured on your host computer. Step-by-step guides for these software tools are found in the Application Notes for Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows.

The USRP E320 requires UHD version 3.13.0.2 or later. It is strongly? recommended to use the latest stable release of UHD on both the host computer and the USRP via the filesystem on the SD card. If this release fails to work in some way, then try the maintenance branch of the latest stable version. If you are operating the device in Network Mode, the version of UHD running on the host machine and E320 USRP must match to within the same maintenance release and branch. See the UHD GitHub repository for the latest release and maintenance branch.

3.6

3.6.1

Listed below are the interfaces to connect to the USRP E320. Each interface has specific functionality, limitations and purpose.

Serial Console

The Serial Console provides a low-level interface to the ARM CPU and STM32 microcontroller, typically used for debugging. The serial console can also be used as a JTAG connection to the FPGA.

1 Gb RJ45 Connection

The 1 Gb RJ45 Connection interfaces with the on-board ARM CPU. When operated in "Network mode", this interface can optionally be used for remote control and management traffic. Regardless of the operation mode (Host vs Embedded) this interface can be used to connect to the ARM via SSH. By default, the 1 Gb RJ45 connection is configured to use a DHCP assigned IP address.

SFP+ Connection

The SFP+ Connection supports multiple interfaces for streaming high-speed, low-latency data, depending upon which FPGA image is loaded.

3.6.2

It is possible to gain shell access to the device using a serial terminal emulator via the Serial Console port. Most Linux, OS X, or other Unix based operating systems have a utility called `screen` which can be used for this purpose.

If you do not have `screen` installed, it can be installed via your distribution's package manager. For Ubuntu/Debian based operating systems it can be installed with the package manager `apt` such as:

```
sudo apt install screen
```

The default Baud Rate for the Serial Console is: `115200`

The exact device node you should attach to depends on your operating system's driver and other USB devices that might already be connected. Modern Linux systems offer alternatives to simply trying device nodes; instead, the OS might have a directory of symlinks under `/dev/serial/by-id`:

```
$ ls /dev/serial/by-id
usb-FTDI_Dual_RS232-HS-if00-port0
usb-FTDI_Dual_RS232-HS-if01-port0
usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6A69-if00-port0
usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6A69-if01-port0
```

NOTE: Exact names depend on the host operating system version and may differ.

Every E320 series device connected to USB will by default show up as four different devices. The devices labeled `"USB_to_UART_Bridge_Controller"` are the devices that offer a serial prompt. The first (with the `if00` suffix) connects to the `STM32 Microcontroller`, whereas the second connects to the `ARM CPU`.

If you have multiple E320 Serial Consoles connected to a single host, you may have to empirically test nodes.

Connecting to the ARM CPU can be performed with the command:

```
$ sudo screen  /dev/serial/by-id/usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6A69-if01-port0 115200
```

Upon starting the USRP E320, boot messages will appear and rapidly update. Once the boot process successfully completes, a login prompt like the following should appear:

```
Alchemy 2018.04 ni-e320-serial ttyPS0
ni-e320-serial login:
```

Enter the username: ?`root`?

By default, the `root` user's password is left blank. Press the `Enter` key when prompted for a password.

You should now be presented with a shell prompt similar to the following:

```
root@ni-e320-<motherboard serial #>:~#
```

Using the default configuration, the serial console will show all kernel log messages (which are not available when using SSH) and give access to the boot loader (U-boot prompt). This can be used to debug kernel or boot-loader issues more efficiently than when logged in via SSH.

3.6.3

Using the Serial Console interface, it is possible to connect to the STM32 microcontroller with the command below. The STM32 controls the power sequencing and several other low-level device operations.

```
$ sudo screen  /dev/serial/by-id/usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6A69-if00-port0 115200
```

The STM32 interface provides a very simple prompt. The command `help` will list all available commands. A direct connection to the microcontroller can be used to hard-reset the device without physically accessing it (i.e., emulating a power button press) and other low-level diagnostics.

3.6.4

By default, the RJ45 1 Gb management interface is configured to be assigned a DHCP IP address.

If you have access to a network which provides a DHCP server (such as a common router's LAN), attach the RJ45 1 Gb port to this network. Details vary by vendor, however, most router management interfaces will provide a list of attached devices to the LAN including their IP address.

Without access to a router management interface, you can identify the IP address by connecting to the ARM CPU via Serial Console as detailed in the section above and running the command `ip a`:

Example Output:

```
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.151/24 brd 192.168.1.255 scope global dynamic eth0
       valid_lft 42865sec preferred_lft 42865sec
3: sfp0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8000 qdisc pfifo_fast qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.2/24 brd 192.168.10.255 scope global sfp0
       valid_lft forever preferred_lft forever
```

If you do not have access to a network with a DHCP server, you can create one using the Linux utility `dnsmasq`:

```
$ sudo dnsmasq -i <ETHERNET_ADAPTER_NAME> --dhcp-range=192.168.1.50,192.168.1.100 --except-interface=lo --bind-dynamic --no-daemon
```

NOTE: Modify the value `<ETHERNET_ADAPTER_NAME>` to match the interface you would like to create a DHCP server on.

After the device has obtained an IP address, you can remotely log into it from a Linux or macOS systems with SSH, as shown below:

```
$ ssh root@192.168.1.51
```

NOTE: The IP address may vary depending on your network setup.

NOTE: The `root` password is empty/blank.

On Microsoft Windows, the SSH connection can be established using the third-party program, such as ?PuTTY.

After logging in, you should be presented with a shell prompt like the following:

```
root@ni-e320-<motherboard serial #>:~#
```

3.7

Before operating the device, it is? ?strongly? recommended to update to the latest version of the Embedded Linux file system. If you are operating the device in Network Mode, the version of UHD running on the host machine and E320 USRP must match.

There is two ways to update the file system for the E320 USRP:

1. Mender

2. Physically remove microSD card from device and write a new file system to the microSD card.

3.7.1

The SD Card is divided into four partitions. There are two root file system partitions, a "boot" partition and a "data" partition.

Any data you would like to preserve through Mender updates should be saved to the "data" partition, which is mounted at `/data`.

3.7.2

Mender is third-party software that enables remote updating of the root file system without physically accessing the device (see also the Mender website https://mender.io). Mender can be executed locally on the device, or a Mender server can be set up which can be used to remotely update an arbitrary number of USRP devices. Users can host their own local Mender server, or use servers hosted by Mender as a paid service; contact Mender for more information.

3.7.2.1

When updating the file system using Mender, the tool will overwrite the root file system partition that is not currently mounted. Any data stored in the root partitions will be permanently lost with a Mender update.

After updating a partition with Mender, it will reboot into the newly updated partition. Only if the update is confirmed by the user, the update will be made permanent. This means that if an update fails, the device will be always able to reboot into the partition from which the update was originally launched, which presumably is in a working state. Another update can be launched now to correct the previous, failed update, until it works.

To obtain the file system Mender image (these are files with a `.mender` suffix), run the following command on the host computer with Internet access:

```
$ sudo uhd_images_downloader -t mender -t e320 --yes
```

Example Output:

```
[INFO] Using base URL: https://files.ettus.com/binaries/cache/
[INFO] Images destination: /usr/local/share/uhd/images
[INFO] No inventory file found at /usr/local/share/uhd/images/inventory.json. Creating an empty one.
301483 kB / 301483 kB (100%) e3xx_e320_mender_default-v4.4.0.0.zip
[INFO] Images download complete.
```

NOTE: In the output of the command, the folder destination where the images are saved is printed out.

Next, you will need to copy this Mender file system image to the USRP E320. This can be done with the Linux utility `scp`.

```
$ scp /usr/local/share/uhd/images/usrp_e320_fs.mender root@192.168.1.51:~/.
```

Note: The path and IP may different for your configuration, the command above assumes you're using the default installation path of `/usr/local` and that the E320's IP is `192.168.1.51`.

After copying the Mender file system image to the E320, connect to the E320 using either the Serial Console, or via SSH to gain shell access.

On the E320, run `mender install /path/to/latest.mender` to update the file system:

```
root@ni-e320-serial:~# mender install /home/root/usrp_e320_fs.mender
```

Example Output:

```
root@ni-e320-316E375:~# mender install /home/root/usrp_e320_fs.mender
INFO[0000] Start updating from local image file: [/home/root/usrp_e320_fs.mender]  module=rootfs
Installing update from the artifact of size 399640064
INFO[0000] opening device /dev/mmcblk0p3 for writing     module=block_device
INFO[0000] partition /dev/mmcblk0p3 size: 2046820352     module=block_device
................................   0% 1024 KiB
................................   0% 2048 KiB
................................   0% 3072 KiB
[truncated for readability]
................................  99% 389120 KiB
................................  99% 390144 KiB
................................ 100% 390273 KiB
INFO[0740] wrote 2046820352/2046820352 bytes of update to device /dev/mmcblk0p3  module=device
INFO[0744] Enabling partition with new image installed to be a boot candidate: 3  module=device
```

The artifact can also be stored on a remote server:

```
$ mender install <http://server.name/path/to/latest.mender>
```

This procedure will take a few minutes to complete. After mender has logged a successful update, reboot the device:

```
$ reboot
```

If the reboot worked, and the device seems functional, commit the changes so that the boot loader knows to permanently boot into this partition:

```
$ mender -commit
```

To identify the currently installed Mender artifact from the command line, the following file can be queried on the E320:

```
$ cat /etc/mender/artifact_info
```

If you are using a Mender server, the updates can be initiated from a web dashboard. From there, you can start the updates without having to log into the device, and you can update groups of USRPs with a few clicks in a web GUI. The dashboard can also be used to inspect the state of USRPs. This is a simple way to update groups of rack-mounted USRPs with custom file systems.

For more information on updating the file-system, refer to the E3xx page in the Devices section of the UHD Manual at ?http://uhd.ettus.com.

3.7.3

The microSD card is accessible directly on the Board-only version of the E320 USRP. The E320 Full Enclosure version must be opened with the included Torx wrench.

NOTE: This method will overwrite all data saved on the microSD card, including any data saved to the `/data` partition.

Please see the separate application note, Writing the USRP File System Disk Image to a SD Card, for step-by-step instructions on writing the file system image to the microSD card.

3.8

The USRP E320 systemd network configuration files are located either at: `/etc/systemd/network/`

```
# ls /etc/systemd/network/
eth0.network  sfp0.network
```

or for newer versions of the file system: `/data/network/`

```
# ls /data/network/
eth0.network  int0.network  sfp0.network
```

For details on configuration please refer to the systemd-networkd manual pages.

The factory settings are as follows:

```
eth0 (DHCP):

    [Match]
    Name=eth0

    [Network]
    DHCP=v4

    [DHCPv4]
    UseHostname=false

sfp0 (static):

    [Match]
    Name=sfp0

    [Network]
    Address=192.168.10.2/24

    [Link]
    MTUBytes=8000
```

Additional notes on networking:

- Care needs to be taken when editing these files on the device, since `vi` / `vim` sometimes generates undo files (e.g. `/data/network/sfp0.network~`), that `systemd-networkd` might accidentally pick up.
- Temporarily setting the IP addresses or MTU sizes via `ifconfig` or other command line tools will only change the value until the next reboot or reload of the FPGA image.
- If the MTU of the device and host computers differ, streaming issues can occur.
- Streaming via SFP0 at 1 Gb rates requires a MTU of `1500`
- Streaming via SFP0 at 10 Gb rates requires a MTU of `8000`

For addition details on network configuration here: https://files.ettus.com/manual/page_usrp_e320.html#e320_network_configuration

3.9

3.9.1

The FPGA image should match the version of UHD installed on the host computer when operated in Network mode.

Network mode FPGA image updates must be made through the RJ45 management interface.

To obtain all the FPGA images for your installed version of UHD, run the following command on the host computer with internet access:

```
$ sudo uhd_images_downloader -t e320 -t fpga
```

Example Output:

```
$ uhd_images_downloader -t e320 -t fpga
[INFO] Images destination: /usr/local/share/uhd/images
[INFO] No inventory file found at /usr/local/share/uhd/images/inventory.json. Creating an empty one.
05920 kB / 05920 kB (100%) e3xx_e320_fpga_default-g494ae8bb.zip
[INFO] Images download complete.
```

There is two versions of the E320 FPGA images shipped with UHD:

- `1G` for 1 Gb rates on the SFP+ port (default image)

- `XG` for 10 Gb rates on the SFP+ port

In this example, we load the `XG` variant of the FPGA image.

```
$ uhd_image_loader --args "type=e3xx,mgmt_addr=<E320_RJ45_IP_ADDR>,fpga=XG"
```

Example Output:

```
$ uhd_image_loader --args "mgmt_addr=192.168.1.51,type=e3xx,fpga=XG"
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.13.1.0-1-gd3b7e90a
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.1.51,type=e3xx,product=e320,serial=316E375,claimed=False,s
[INFO] [MPMD] Claimed device without full initialization.
[INFO] [MPMD IMAGE LOADER] Starting update. This may take a while.
[INFO] [MPM.PeriphManager] Updating component `fpga'
[INFO] [MPM.PeriphManager] Updating component `dts'
[INFO] [MPM.RPCServer] Resetting peripheral manager.
[INFO] [MPM.PeriphManager] Device serial number: 316E375
[INFO] [MPMD IMAGE LOADER] Update component function succeeded.
[INFO] [MPM.PeriphManager] Found 1 daughterboard(s).
```

The FPGA is immediately updated, and this FPGA image will continue to be used. The device does not need to be power cycled to use the new image.

To load a different FPGA image (i.e. `1G`), modify the device argument `fpga=` to a value of `fpga=1G`.

To specify the path to a custom FPGA image, use the ?`--fpga-path`? argument.

```
$ uhd_image_loader --args "type=e3xx,mgmt_addr=<E320_RJ45_IP_ADDR>" --fpga-path=/path/to/custom/fpga.bit
```

The Verilog code for the FPGA in the USRP E320 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

3.9.2

It is possible to update the FPGA image when operated in Embedded mode. Connect to the ARM CPU via Serial Console or via SSH. It is generally recommend to use SSH over the RJ45 interface for remote management.

Run the command `uhd_images_downloader` to download the FPGA images to the device's file system:

NOTE: The 1 Gb RJ45 management interface will require Internet access for this next step.

```
root@ni-e320-serial:~# python3 /usr/bin/uhd_images_downloader -t e320 -t fpga
[INFO] Images destination: /usr/share/uhd/images
[INFO] No inventory file found at /usr/share/uhd/images/inventory.json. Creating an empty one.
05920 kB / 05920 kB (100%) e3xx_e320_fpga_default-g494ae8bb.zip
[INFO] Images download complete.
```

NOTE: The default UHD FPGA Images destination within the E320's file-system is `/usr/share/uhd/images`. The default UHD FPGA Images destination on a typical host installation is `/usr/local/share/uhd/images`.

Updating the FPGA image from the ARM CPU is the same as detailed above for a Network mode update:

```
root@ni-e320-serial:~# uhd_image_loader --args "type=e3xx,fpga=1G"
[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_106600; UHD_3.13.1.0-0-unknown
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=127.0.0.1,type=e3xx,product=e320,serial=316E375,claimed=False,skip_in
[INFO] [MPM.PeriphManager.UDP] No CHDR interfaces found!
[INFO] [MPM.PeriphManager.UDP] No CHDR interfaces found!
[INFO] [MPMD] Claimed device without full initialization.
[INFO] [MPMD IMAGE LOADER] Starting update. This may take a while.
[INFO] [MPM.PeriphManager] Updating component `fpga'
[INFO] [MPM.PeriphManager] Updating component `dts'
[INFO] [MPM.RPCServer] Resetting peripheral manager.
[INFO] [MPM.PeriphManager] Device serial number: 316E375
[INFO] [MPMD IMAGE LOADER] Update component function succeeded.
[INFO] [MPM.PeriphManager] Found 1 daughterboard(s).
```

For more information on updating the FPGA image, refer to the UHD Manual at http://uhd.ettus.com.

3.10

The device supports multiple high-speed, low-latency interfaces on the SFP+ port for streaming samples to the host computer.

3.10.1

Complete the steps below to set up a streaming connection over the 1 Gb Ethernet interface on the `SFP+ Port`.

NOTE: The `1G` FPGA image must be loaded for the `SFP+ Port` to operate at 1 Gb speeds. If the `XG` image is loaded, the port will be unresponsive at 1Gb speeds.

1. Configure your Host's 1 Gb Ethernet interface as shown below. This interface should be separate from the 1 Gb NIC/network which is connected to the 1 Gb RJ45 management interface.

```
IP Address: 192.168.10.1
Subnet Mask: 255.255.255.0
Gateway: 0.0.0.0
MTU: 1500
```

NOTE: When operating the `SFP+ Port` at 1 Gb speeds, it is important to set a MTU of `1500` and not a value of `automatic`. Mismatched MTU values on either the Host or E320 may cause flow control errors. Your computer may need to be restarted for the MTU value to take effect.

2. Insert the RJ45-to-SFP+ adapter ?into the? `SFP+ Port`?.

3. Connect the SFP+ adapter on the device to an Ethernet port on the host computer using a standard Ethernet cable.

The ? Green LED? above the ?`SFP+ Port`? should illuminate.

4. To test the connection,? ?`ping`? the device at address `192.168.10.2?` from the host, as shown below:

```
$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=1.06 ms
^C
--- 192.168.10.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.065/1.065/1.065/0.000 ms
```

Press `CTRL+C` to stop the ping program.

5. Verify your MTU is set correctly for 1 Gb speeds on the E320. See the section Updating the Network Configurations for additional details.

Proceed to the next section Verifying Device Operation.

3.10.2

Load the `XG` FPGA image for 10 Gb streaming as detailed in the section Updating the FPGA Image. You will need to use a 10 GigE cable that can be plugged in directly to the SFP+ connector on the board.

NOTE: The `XG` FPGA image must be loaded for the `SFP+ Port` to operate at 10 Gb speeds. If the `1G` image is loaded, the port will be unresponsive at 10 Gb speeds. Mismatched MTU values on either the Host or E320 may cause flow control errors.

1. Configure your Host's 10 Gb Ethernet interface as shown below.

```
IP Address: 192.168.10.1
Subnet Mask: 255.255.255.0
```

```
Gateway: 0.0.0.0
MTU: 8000
```

NOTE: When operating the `SFP+ Port` at 10 Gb speeds, it is important to set a MTU of `8000` and not a value of `automatic`. Mismatched MTU values on either the Host or E320 may cause flow control errors. Your computer may need to be restarted for the MTU value to take effect.

2. Connect the SFP+ port on the device to an Ethernet port on the host computer using a 10 Gb SFP+ copper or fiber cable.

The ? Green LED? above the ?`SFP+ Port`? should illuminate.

4. To test the connection,? ?`ping`? the device at address `192.168.10.2?` from the host, as shown below:

```
$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=1.06 ms
^C
--- 192.168.10.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.065/1.065/1.065/0.000 ms
```

Press `CTRL+C` to stop the ping program.

5. Verify your MTU is set correctly for 10 Gb speeds on the E320. See the section Updating the Network Configurations for additional details.

Proceed to the next section Verifying Device Operation.

3.11

Once you have successfully setup a management interface and streaming interface, you can now verify the devices operation using the include UHD utilities.

3.11.1

The USRP E320 contains 2 channels, each represented on the front panel as `RF A` and `RF B`. Below is the `subdev` mapping of RF Ports.

E320

- RF A = A:0
- RF B = A:1

Additional details of UHD Subdevice Specifications can be found here in the UHD Manual:
http://files.ettus.com/manual/page_configuration.html#config_subdev

3.11.2

The USRP E320 supports master clock rate from 200 kHz to 61.44 MHz and can be changed by adding `master_clock_rate=<rate>` to the default UHD args. The default master clock rate is 16 MHz.

Sample rates as delivered to/from the host computer for USRP devices are constrained to follow several important rules.

It is important to understand that strictly-integer decimation and interpolation are used within USRP hardware to meet the requested sample rate requirements of the application at hand. That means that the desired sample rate must meet the requirement that master-clock-rate/desired-sample-rate be an integer ratio. Further, it is strongly desirable for that ratio to be even. This ratio is the decimation (down-conversion) or interpolation (up-conversion) factor. The decimation or interpolation factor may be between 1 and 1024. There are further constraints on the decimation or interpolation factor. If the decimation or interpolation factor exceeds 128, then it must be evenly divisible by 2. If the decimation or interpolation factor exceeds 256, then it must be evenly divisible by 4.

Additional information on Sample Rates can be found here in the UHD Manual:
http://files.ettus.com/manual/page_general.html#general_sampleratenotes

3.11.3

The UHD utility `uhd_usrp_probe` provides detailed information of the USRP device.

From your host computer, run the command `uhd_usrp_probe`:

```
$ uhd_usrp_probe --args "addr=192.168.10.2"
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.13.1.0-1-gd3b7e90a
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.10.2,type=e3xx,product=e320,serial=316E375,claimed=False,addr
[INFO] [MPM.PeriphManager] init() called with device args `product=e320,mgmt_addr=192.168.10.2'.
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000000)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1343 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1335 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000003320)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000002)
[INFO] [0/Radio_0] Performing CODEC loopback test...
[INFO] [0/Radio_0] CODEC loopback test passed
[INFO] [0/Radio_0] Performing CODEC loopback test...
[INFO] [0/Radio_0] CODEC loopback test passed
  ___________________________________________________
 /
|       Device: E300-Series Device
|     ___________________________________________________
|    /
|   |       Mboard: ni-e320-316E375
|   |   eeprom_version: 2
|   |   mpm_version: 3.13.1.0-gd3b7e90a
|   |   pid: 58144
|   |   product: e320
|   |   rev: 2
|   |   rpc_connection: remote
|   |   serial: 316E375
|   |   type: e3xx
```

```
|   |   MPM Version: 1.2
|   |   FPGA Version: 3.0
|   |   RFNoC capable: Yes
|   |
|   |   Time sources:  internal, external, gpsdo
|   |   Clock sources: external, internal, gpsdo
|   |   Sensors: gps_locked, temp_main_power, ref_locked, temp_rf_channelA, temp_fpga, gps_sky, temp_rf_channelB, fan, temp_internal, gps_tpv
|   |     ____________________________________________________
|   |    /
|   |   |       RX Dboard: A
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: Neon
|   |   |   |   Antennas: RX2, TX/RX
|   |   |   |   Sensors: lo_locked, ad9361_temperature, rssi, lo_lock
|   |   |   |   Freq range: 70.000 to 6000.000 MHz
|   |   |   |   Gain range PGA: 0.0 to 76.0 step 1.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 40000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 1
|   |   |   |   Name: Neon
|   |   |   |   Antennas: RX2, TX/RX
|   |   |   |   Sensors: lo_locked, ad9361_temperature, rssi, lo_lock
|   |   |   |   Freq range: 70.000 to 6000.000 MHz
|   |   |   |   Gain range PGA: 0.0 to 76.0 step 1.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 40000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Codec: A
|   |   |   |   Name: AD9361 Dual ADC
|   |   |   |   Gain Elements: None
|   |     ____________________________________________________
|   |    /
|   |   |       TX Dboard: A
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: Neon
|   |   |   |   Antennas: TX/RX
|   |   |   |   Sensors: lo_locked, ad9361_temperature
|   |   |   |   Freq range: 47.000 to 6000.000 MHz
|   |   |   |   Gain range PGA: 0.0 to 89.8 step 0.2 dB
|   |   |   |   Bandwidth range: 20000000.0 to 40000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Frontend: 1
|   |   |   |   Name: Neon
|   |   |   |   Antennas: TX/RX
|   |   |   |   Sensors: lo_locked, ad9361_temperature
|   |   |   |   Freq range: 47.000 to 6000.000 MHz
|   |   |   |   Gain range PGA: 0.0 to 89.8 step 0.2 dB
|   |   |   |   Bandwidth range: 20000000.0 to 40000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Codec: A
|   |   |   |   Name: AD9361 Dual DAC
|   |   |   |   Gain Elements: None
|   |     ____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * DDC_0
|   |   |   * DUC_0
```

3.11.4

The UHD driver includes several example programs, which may serve as test programs or the basis for your application program. The source code can be obtained from the UHD repository on github at: https://github.com/EttusResearch/uhd/tree/master/host/examples

You can quickly verify the operation of your USRP E320 by running the `rx_ascii_art_dft` UHD example program.

The `rx_ascii_art_dft` utility is a simple console –based, real-time FFT display tool. It is not graphical in nature, so it can be easily run over an SSH connection within a terminal window, and does not need any graphical capability, such as X Windows, to be installed. It can also be run over a serial console connection, although this is not recommended, as the formatting may not render correctly.

You can run a simple test of the E320 USRP by connecting an antenna and observing the spectrum of a commercial FM radio station in real-time, following the steps below:

1. Attach an antenna to the `RF A / RX2`– antenna port of the E320.

2. From your host computer, run the command:

```
$ /usr/local/lib/uhd/examples/rx_ascii_art_dft \
--args "addr=192.168.10.2" \
--freq 98.5e6 \
--rate 2e6 \
--gain 40 \
--ref-lvl="-30" \
```

```
   --dyn-rng 90 \
   --ant "RX2" \
   --subdev "A:0"
```

NOTE: Modify the command–line argument `freq` ?above to specify a tuning frequency for a strong local FM radio station. You will also need to update the IP Address to match your device IP.

3. You should see a real-time FFT display of 2 MHz of spectrum, centered at the specified tuning frequency.

4. Type "`Q`" to stop the program and to return to the Linux command line.

5. You can run with the `?----help` ?argument to see a description of all available command-line options.

Example Output:

```
$ ./rx_ascii_art_dft --args "addr=192.168.10.2" --freq 98.5e6 --rate 2e6 --gain 40 --ref-lvl="-30" --dyn-rng 90 --ant "RX2" --subdev "A:0"

Creating the usrp device with: addr=192.168.10.2...
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.13.1.0-1-gd3b7e90a
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.10.2,type=e3xx,product=e320,serial=316E375,claimed=False,addr
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000000)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1334 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1325 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000003320)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000002)
[INFO] [MPM.PeriphManager] init() called with device args `product=e320,mgmt_addr=192.168.10.2'.
[INFO] [0/Radio_0] Performing CODEC loopback test...
[INFO] [0/Radio_0] CODEC loopback test passed
[INFO] [0/Radio_0] Performing CODEC loopback test...
[INFO] [0/Radio_0] CODEC loopback test passed
Using Device: Single USRP:
  Device: E300-Series Device
  Mboard 0: ni-e320-316E375
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: Neon
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: Neon
  TX Channel: 1
    TX DSP: 1
    TX Dboard: A
    TX Subdev: Neon

Setting RX Rate: 2.000000 Msps...
Actual RX Rate: 2.000000 Msps...

Setting RX Freq: 98.500000 MHz...
Actual RX Freq: 98.500000 MHz...

Setting RX Gain: 40.000000 dB...
Actual RX Gain: 40.000000 dB...

Checking RX: all_los: locked ...

Done!
```

3.11.5

Included with the UHD driver example programs is a utility, `benchmark_rate` to benchmark the transport link of the system.

A system's maximum performance is dependent upon many factors. `benchmark_rate` will exercise the transport link and CPU of the system.

3.11.5.1

NOTE: This example requires the `1G` FPGA image to be loaded.

This example will test one full-duplex stream using "RFA/A:0", at a rate of 2 MS/s, for 60 seconds:

```
   /usr/local/lib/uhd/examples/benchmark_rate  \
   --args "addr=192.168.10.2" \
   --duration 60 \
   --channels "0" \
   --rx_rate 2e6 \
   --rx_subdev "A:0" \
   --tx_rate 2e6 \
   --tx_subdev "A:0"
```

This example will test two full-duplex streams at 2 MS/s, for 60 seconds:

```
   /usr/local/lib/uhd/examples/benchmark_rate  \
   --args "addr=192.168.10.2" \
   --duration 60 \
   --channels "0,1" \
   --rx_rate 2e6 \
   --rx_subdev "A:0 A:1" \
   --tx_rate 2e6 \
   --tx_subdev "A:0 A:1"
```

This example will test two full-duplex streams at 12.5 MS/s, for 60 seconds:

```
   /usr/local/lib/uhd/examples/benchmark_rate  \
   --args "addr=192.168.10.2,master_clock_rate=25e6" \
   --duration 60 \
   --channels "0,1" \
   --rx_rate 12.5e6 \
   --rx_subdev "A:0 A:1" \
   --tx_rate 12.5e6 \
```

```
--tx_subdev "A:0 A:1"
```

When streaming samples over a 1 Gb transport link, the maximum accumulative rate for all channels is 25 MS/s with a `sc16` OTW format. To achieve higher streaming rates, it is recommended to use the 10 Gb interfaces.

3.11.5.2

NOTE: These examples require the `XG` FPGA image to be loaded.

This example will test one full-duplex stream using "RFA/A:0", at a rate of 61.44 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "addr=192.168.10.2,master_clock_rate=61.44e6" \
--duration 60 \
--channels "0" \
--rx_rate 61.44e6 \
--rx_subdev "A:0" \
--tx_rate 61.44e6 \
--tx_subdev "A:0"
```

This example will test two full-duplex stream, at a rate of 30.72 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "addr=192.168.10.2,master_clock_rate=61.44e6" \
--duration 60 \
--channels "0,1" \
--rx_rate 30.72e6 \
--rx_subdev "A:0 A:1" \
--tx_rate 30.72e6 \
--tx_subdev "A:0 A:1"
```

3.12

3.12.1

To avoid damaging the file system and causing any corruption, do not turn the device off with the power button without first shutting down the system. Use this command to cleanly and properly shut the system down:

```
shutdown --h now
```

3.12.2

The USRP E320 can be configured to power on and boot automatically when power is applied. By default, autoboot is disabled on all USRPs that support it. To control autoboot on the USRP E320, first determine the current value for `MCU_FLAGS[0]` by running `eeprom-dump`; the meaning of the `MCU_FLAGS[0]` is found in the UHD manual. The least significant bit when `MCU_FLAGS[0]` is viewed as a binary value controls the autoboot.

For example

```
root@ni-e320-XXXXXXX:~# eeprom-dump
-- PID/REV: e320 0002
-- MCU_FLAGS[0]: 00000008
-- MCU_FLAGS[1]: 00000000
-- MCU_FLAGS[2]: 00000000
-- MCU_FLAGS[3]: 00000000
-- Serial: XXXXXXX
-- eth_addr0: XX:XX:XX:XX:XX:XX
-- eth_addr1: XX:XX:XX:XX:XX:XX
-- eth_addr2: XX:XX:XX:XX:XX:XX
-- DT-Compat/MCU-Compat: 0000 0002
-- CRC: cbd79a61 (matches)
```

shows `-- MCU_FLAGS[0]: 00000008`; `0x08` (`0b00001000` in binary) indicates that autoboot is disabled. If this value were `0x09` (`0b00001001` in binary) it would indicate that autoboot is enabled because least significant bit is 1; same would be true if this value is `0x01` (`0b00000001` in binary).

To enable or disable autoboot, copy the existing value of `MCU_FLAGS[0]` retrieved by `eeprom-dump` into `<MCU_FLAGS[0]>` below and run the command:

- Disable autoboot on USRP E320 (sets least significant bit to 0), regardless of whether currently enabled or disabled:

```
root@ni-e320-XXXXXXX:~# eeprom-set-flags $((0x<MCU_FLAGS[0]> & ~0x1))
```

Thus, for the value noted above (autoboot is already disabled, so this command doesn't actually change anything):

```
root@ni-e320-XXXXXXX:~# eeprom-set-flags $((0x00000008 & ~0x1))
-- PID/REV: e320 0002
-- MCU_FLAGS[0]: 00000008
-- MCU_FLAGS[1]: 00000000
-- MCU_FLAGS[2]: 00000000
-- MCU_FLAGS[3]: 00000000
-- Serial: XXXXXXX
-- eth_addr0: XX:XX:XX:XX:XX:XX
-- eth_addr1: XX:XX:XX:XX:XX:XX
-- eth_addr2: XX:XX:XX:XX:XX:XX
-- DT-Compat/MCU-Compat: 0000 0002
-- CRC: cbd79a61 (matches)
-- Reading back
-- PID/REV: e320 0002
-- MCU_FLAGS[0]: 00000008
-- MCU_FLAGS[1]: 00000000
-- MCU_FLAGS[2]: 00000000
-- MCU_FLAGS[3]: 00000000
-- Serial: XXXXXXX
-- eth_addr0: XX:XX:XX:XX:XX:XX
-- eth_addr1: XX:XX:XX:XX:XX:XX
-- eth_addr2: XX:XX:XX:XX:XX:XX
-- DT-Compat/MCU-Compat: 0000 0002
-- CRC: 448fb572 (matches)
```

- Enable autoboot on USRP E320 (sets least significant bit to 1), regardless of whether currently enabled or disabled. For example when changing from autoboot disabled to enabled:

```
root@ni-e320-XXXXXXX:~# eeprom-set-flags $((0x<MCU_FLAGS[0]> | 0x1))
```

Thus, for the value noted above:

```
root@ni-e320-XXXXXXX:~# eeprom-set-flags $((0x00000008 | 0x1))
-- PID/REV: e320 0002
-- MCU_FLAGS[0]: 00000008
-- MCU_FLAGS[1]: 00000000
-- MCU_FLAGS[2]: 00000000
-- MCU_FLAGS[3]: 00000000
-- Serial: XXXXXXX
-- eth_addr0: XX:XX:XX:XX:XX:XX
-- eth_addr1: XX:XX:XX:XX:XX:XX
-- eth_addr2: XX:XX:XX:XX:XX:XX
-- DT-Compat/MCU-Compat: 0000 0002
-- CRC: cbd79a61 (matches)
-- Reading back
-- PID/REV: e320 0002
-- MCU_FLAGS[0]: 00000009
-- MCU_FLAGS[1]: 00000000
-- MCU_FLAGS[2]: 00000000
-- MCU_FLAGS[3]: 00000000
-- Serial: XXXXXXX
-- eth_addr0: XX:XX:XX:XX:XX:XX
-- eth_addr1: XX:XX:XX:XX:XX:XX
-- eth_addr2: XX:XX:XX:XX:XX:XX
-- DT-Compat/MCU-Compat: 0000 0002
-- CRC: 448fb572 (matches)
```

If setting this flag *does not* allow autoboot control on the USRP E320, then the device boot firmware needs to be updated. This update is accomplished via the following instructions.

On the USRP E320 via ssh or serial terminal, download the update MCU firmware and extract it:

```
root@ni-e320-XXXXXXX:~# curl https://files.ettus.com/binaries/misc/upgrade_mcu_neon_v1.1.7358-a190641-musl-glibc-rev3-7.tar.gz | tar zxf -
```

This will create a directory `upgrade_mcu_neon_v1.1.7358-a190641-musl-glibc-rev3-7`. Go into this directory and run the firmware flash script:

```
root@ni-e320-XXXXXXX:~# cd upgrade_mcu_neon_v1.1.7358-a190641-musl-glibc-rev3-7
root@ni-e320-XXXXXXX:~/upgrade_mcu_neon_v1.1.7358-a190641-musl-glibc-rev3-7# ./flash-firmware.sh
This script updates the microcontroller firmware (RO part). The change is
persistent across power cycles. Incorrect updates can only fixed be a manual
process which requires opening the enclosure.

Updating the microcontroller firmware (RO part) is only required if the Ettus
Research support told you to do so.

Press "y" to continue
```

At the prompt, press the `y` key to continue. Pressing any other key aborts the procedure:

```
Press "y" to continue n

aborting
root@ni-e320-317F9BF:~/upgrade_mcu_neon_v1.1.7358-a190641-musl-glibc-rev3-7#
```

Pressing the `y` key:

```
Press "y" to continue y

This script will flash ec-neon-rev3.RO.flat to the device
old RO version:    neon_vX.X.XXXX-XXXXXXX
new RO version:    neon_v1.1.7358-a190641

Press "y" to continue
```

At the prompt, press the `y` key *again* to continue. Pressing any other key aborts the procedure as before.

```
Press "y" to continue y

./ectool --interface=dev reboot_ec RW
./ectool --interface=dev flashread 0x0 65536 ec-neon-rev3.RO.flat.old
Reading 65536 bytes at offset 0...
done.
./ectool --interface=dev flasherase 0x0 65536
Erasing 65536 bytes at offset 0...
done.
./ectool --interface=dev flashwrite 0x0 ec-neon-rev3.RO.flat
Reading 49592 bytes from ec-neon-rev3.RO.flat...
Writing to offset 0...
Write size 112...
done.

copying new firmware files
'ec-neon-rev3.bin' -> '/lib/firmware/ni/ec-neon-rev3.bin'
'ec-neon-rev3.RW.bin' -> '/lib/firmware/ni/ec-neon-rev3.RW.bin'
root@ni-e320-317F9BF:~/upgrade_mcu_neon_v1.1.7358-a190641-musl-glibc-rev3-7#
```

Once the script is done, reboot the USRP (e.g., `shutdown -r now`), and when it comes up the autoboot flag should now work as desired. If these instructions *do not* work, then email support@ettus.com and ask for alternative instructions on how to update the USRP E320 RO and RW boot firmware such that this EEPROM flag setting is honored.

3.12.3

The default user is `root` and the password is empty (no password).

It is recommended to update the `root` password, which can be done with the command `passwd`:

Example Output:

```
root@ni-e320-serial:~# passwd
Changing password for root
New password:
Re-enter new password:
passwd: password changed.
```

3.13

3.13.1

In some streaming modes, the Intel I219-LM NIC can produce flow control and sequence errors. It is recommended to use a USB3 to 1 Gb Ethernet Adapter for hosts which have an I219-LM NIC.

3.14

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

3.15

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

3.16

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

3.17

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

4 N210 Getting Started Guides

4.1

- USRP N200/N210
- 1 Gigabit Ethernet Cable
- Power Supply
- 2 SMA-Bulkhead Cables

4.2

Make sure that your kit contains all the items listed above. If any items are missing, please contact your sales agent or Ettus Research Technical support immediately.

4.3

- A host computer with an available 1-GigE port

4.4

All Ettus Research products are individually tested before shipment. The USRP? is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP? can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

4.5

The USRP N200/N210 and compatible RF daughterboards are shipped separately. To operate the device you will need to install the RF daughterboards and supplied bulkhead cables. This can be accomplished by removing the top plate of the USRP N200/N210, which is secured with two screws. After installation, the daughterboards and cables should be secured with the hardware provided. The device must be powered off when installing daughterboards to avoid potential damage. For detailed step-by-step instructions on assembling up your USRP N200/N210, see the USRP N Series Quick Start (Daughterboard Installation) page.

4.6

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on your host computer. A step-by-step guide for doing this is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows Application Notes. Release 3.8.4 or later of the USRP Hardware Driver, UHD, is required. It is recommended to use the latest stable version of UHD that is available.

If you have a USB stick with the Live SDR Environment installed on it, then you may boot your host computer from that. The LiveUSB SDR Environment does not require anything to be installed on your host computer, and contains a Linux-based environment with the UHD software and the GNU Radio framework already installed. More information about the Live SDR Environment is available at the Live SDR Environment Getting Started Guides page.

4.7

Once the software tools are installed on the host computer, or using the Live SDR Environment, verify the correct operation of the USRP by running the utility programs on the host computer. More information is available at the Verifying the Operation of the USRP Using UHD and GNU Radio Application Note.

4.8

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

4.9

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

4.10

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

4.11

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

5 N321 Getting Started Guide

5.1

5.1.1

- USRP N300
- DC Power Supply (12V, 7A)
- 1 RJ45 ? SFP+ Adapter
- 1 Gigabit Ethernet Cat-5e Cable (3m)
- USB-A to Micro USB-B Cable (1m)
- Getting Started Guide
- Ettus Research Sticker

5.1.2

- USRP N310
- DC Power Supply (12V, 7A)
- 1 RJ45 ? SFP+ Adapter
- 1 Gigabit Ethernet Cat-5e Cable (3m)
- USB-A to Micro USB-B Cable (1m)
- Getting Started Guide
- Ettus Research Sticker

5.1.3

- USRP N320
- DC Power Supply (12V, 7A)
- 1 RJ45 ? SFP+ Adapter
- 1 Gigabit Ethernet Cat-5e Cable (3m)
- USB-A to Micro USB-B Cable (1m)
- Getting Started Guide
- Ettus Research Sticker

5.1.4

- USRP N321
- DC Power Supply (12V, 7A)
- 1 RJ45 ? SFP+ Adapter
- 1 Gigabit Ethernet Cat-5e Cable (3m)
- USB-A to Micro USB-B Cable (1m)
- Getting Started Guide
- Ettus Research Sticker

5.2

Ensure that your kit contains all the items listed above. If any items are missing, please contact sales@ettus.com? immediately.

5.3

- microSD Card Writer
- For Network Mode: A host computer with an available 1 or 10 Gigabit Ethernet interface for sample streaming. In addition to the Ethernet interface used for sampling streaming, your host computer will require a separate 1 Gigabit Ethernet interface for command and control streaming.
- For Stand-Alone Embedded Mode: A host computer with an available 1 Gigabit Ethernet port or a USB 2.0 port to remotely access the embedded Linux operating system running on ARM CPU.

5.4

All Ettus Research products are individually tested before shipment. The USRP is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP can cause the device to become non-functional. Take the following precautions to prevent damage to the unit.

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water or condensing moisture to come into contact with the device.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

5.5

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on your host computer. A step-by-step guide for doing this is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows Application Notes.

To find the latest release of UHD, see the UHD repository at https://github.com/EttusResearch/uhd.

The USRP N310 requires UHD version 3.11.0.0 or later.

The USRP N300 requires UHD version 3.12.0.0 or later.

The USRP N320/N321 requires UHD version 3.14.0.0 or later.

White Rabbit Ethernet-Based Synchronization of the N3xx USRP requires UHD version 3.12.0.0 or later. For additional details on White Rabbit Ethernet-Based Synchronization, please see the application note, Using Ethernet-Based Synchronization on the USRP? N3xx Devices.

When you receive a brand-new device, it is strongly recommended that you download the most recent filesystem image from the Ettus Research website and write it to the SD card that comes with the unit. It is not recommended that you use the SD card from the factory as-is. Instructions on downloading the latest filesystem image and writing it to the SD card are listed below.

Note that if you are operating the device in Network Mode, the version of UHD running on the host computer and the USRP N3xx must match.

5.6

5.6.1

Listed below are the interfaces to connect to the USRP N3xx. Each interface has specific functionality, limitations and purpose.

Serial Console

The Serial Console provides a low level interface to the device typically used for debugging.

1 Gigabit RJ45 Connection

The 1 Gigabit RJ45 Connection interfaces with the on-board ARM CPU. When operated in "Network mode", this interface can optionally be used for UHD management traffic. Regardless of the operation mode (Network vs Embedded) this interface can be used to connect to the ARM via SSH. By default, the 1Gb RJ45 connection is configured to use a DHCP assigned IP address.

Dual SFP+ Connections

The Dual SFP+ Connections support multiple configurations for streaming high-speed, low-latency data, depending upon the FPGA image which is loaded.

QSFP+ Connection (N320/ N321 Only)

The QSFP+ Connection supports 2 x 10Gb lanes for streaming high-speed, low-latency data, while the onboard SFP0 connection is used for White Rabbit Ethernet-Based Synchronization.

5.6.2

It is possible to gain shell access to the device using a serial terminal emulator via the Serial Console port. Most Linux, OSX, or other Unix based operating systems have a tool called `screen` which can be used for this purpose.

If you do not have `screen` installed, it can be installed via your package manager. For Ubuntu/Debian based operating systems it can be installed with `apt` such as:

```
sudo apt install screen
```

The default Baud Rate for the Serial Console is: `115200`

The exact device node you should attach to depends on your operating system's driver and other USB devices that might already be connected. Modern Linux systems offer alternatives to simply trying device nodes; instead, the OS might have a directory of symlinks under `/dev/serial/by-id`:

```
$ ls /dev/serial/by-id
usb-Digilent_Digilent_USB_Device_25163511FE00-if00-port0
usb-Digilent_Digilent_USB_Device_25163511FE00-if01-port0
usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6CB5-if00-port0
usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6CB5-if01-port0
```

NOTE: Exact names depend on the host operating system version and may differ.

Every N3XX series device connected to USB will by default show up as four different devices. The devices labeled `"USB_to_UART_Bridge_Controller"` are the devices that offer a serial prompt. The first (with the `if00` suffix) connects to the `ARM CPU`, whereas the second connects to the `STM32 Microcontroller`.

If you have multiple N3xx Serial Consoles connected to a single host, you may have to empirically test nodes.

Connecting to the ARM CPU can be performed with the command:

```
$ sudo screen /dev/serial/by-id/usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6CB5-if00-port0 115200
```

Upon starting the USRP N3xx, boot messages will appear and rapidly update. Once the boot process successfully completes, a login prompt like the following should appear:

```
OpenEmbedded test ni-n3xx-313ABDA ttyPS0

ni-n3xx-313ABDA login:
```

Enter the username: ?`root`?

By default, the `root` user's password is left blank. Press the `Enter` key when prompted for a password.

You should now be presented with a shell prompt similar to the following:

```
root@ni-n3xx-<motherboard serial #>:~#
```

Using the default configuration, the serial console will show all kernel log messages (which are not available when using SSH), and give access to the boot loader (U-boot prompt). This can be used to debug kernel or boot-loader issues more efficiently than when logged in via SSH.

5.6.2.1

Using the Serial Console interface, it is possible to connect to the STM32 microcontroller with the command below. The STM32 controls the power sequencing and several other low level device operations.

```
$ sudo screen /dev/serial/by-id/usb-Silicon_Labs_CP2105_Dual_USB_to_UART_Bridge_Controller_007F6CB5-if01-port0 115200
```

The STM32 interface provides a very simple prompt. The command `help` will list all available commands. A direct connection to the microcontroller can be used to hard-reset the device without physically accessing it (i.e., emulating a power button press) and other low-level diagnostics.

5.6.3

By default, the RJ45 1Gb management interface is configured to be assigned a DHCP IP address.

If you have access to a network which provides a DHCP server (such as a common router's LAN), attach the RJ45 1Gb port to this network. Details vary by vendor, however, most router management interfaces will provide a list of attached devices to the LAN including their IP address.

Without access to a router management interface, you can identify the IP address by connecting to the ARM CPU via Serial Console as detailed in the section above and running the command `ip a`:

Example Output:

```
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.151/24 brd 192.168.1.255 scope global dynamic eth0
       valid_lft 42865sec preferred_lft 42865sec
3: sfp0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc pfifo_fast qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.2/24 brd 192.168.10.255 scope global sfp0
       valid_lft forever preferred_lft forever
4: sfp1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 9000 qdisc pfifo_fast qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
```

If you do not have access to a network with a DHCP server, you can create one using the Linux utility `dnsmasq`:

```
$ sudo dnsmasq -i <ETHERNET_ADAPTER_NAME> --dhcp-range=192.168.1.151,192.168.1.254 --except-interface=lo --bind-dynamic --no-daemon
```

NOTE: Modify the value `<ETHERNET_ADAPTER_NAME>` to match the interface you would like to create a DHCP server on.

After the device has obtained an IP address, you can remotely log into it from a Linux or macOS system with SSH, as shown below:

```
$ ssh root@192.168.1.151
```

NOTE: The IP address may vary depending on your network setup.

NOTE: The `root` password default password is empty/blank.

On Microsoft Windows, the SSH connection can be established using the third-party program ?Putty?.

After logging in, you should be presented with a shell like the following:

```
root@ni-n3xx-<motherboard serial #>:~#
```

5.7

Before operating the device, it is? ?strongly? recommended to update to the latest version of the Embedded Linux file system. If you are operating the device in Network Mode, the version of UHD running on the host machine and N3xx USRP must match.

There is two ways to update the file system for the N3xx USRP:

1. Mender

2. Physically remove microSD card from device and write a new file system to the microSD card.

5.7.1

The SD Card is divided into four partitions. There is two root file system partitions, a boot partition and a data partition.

Any data you would like to preserve through Mender updates should be saved to the `data` partition, which is mounted at `/data`.

5.7.2

Mender is third-party software that enables remote updating of the root file system without physically accessing the device (see also the Mender website https://mender.io). Mender can be executed locally on the device, or a Mender server can be set up which can be used to remotely update an arbitrary number of USRP devices. Users can host their own local Mender server, or use servers hosted by Mender as a paid service; contact Mender for more information.

5.7.2.1

When updating the file system using Mender, the tool will overwrite the root file system partition that is not currently mounted. Any data stored in the root partitions will be permanently lost with a Mender update.

After updating a partition with Mender, it will reboot into the newly updated partition. Only if the update is confirmed by the user, the update will be made permanent. This means that if an update fails, the device will be always able to reboot into the partition from which the update was originally launched, which presumably is in a working state. Another update can be launched now to correct the previous, failed update, until it works.

To obtain the file system Mender image (these are files with a `.mender` suffix), run the following command on the host computer with Internet access:

```
$ sudo uhd_images_downloader -t mender -t n3xx --yes
```

Example Output:

```
[INFO] Using base URL: https://files.ettus.com/binaries/cache/
[INFO] Images destination: /usr/local/share/uhd/images
365684 kB / 365684 kB (100%) n3xx_common_mender_default-v4.6.0.0.zip
[INFO] Images download complete.
```

The downloaded "zip" archive is extracted into the `Images destination` directory with the filename `usrp_n3xx_fs.mender`. Next, you will need to copy this Mender file system image from the `Images destination` directory to the USRP N3xx to have its filesystem changed. This can be done with the Linux utility `scp`, for example as follows:

```
$ scp /usr/local/share/uhd/images/usrp_n3xx_fs.mender root@192.168.1.51:~/.
```

Note: The path and IP may be different for your configuration; the command above assumes you're using the default UHD installation path of `/usr/local` and that the N3xx's IP is `192.168.1.51`.

After copying the Mender file system image to the N3xx, connect to the N3xx to gain shell access via either the Serial Console or SSH.

On the N3xx, you first need to determine the version of UHD currently running on the USRP; an easy way to do this is via the command

```
# uhd_config_info --version
```

Example output:

```
UHD 3.14.1.1-0-g0347a6d8
```

The mender command to execute is different for UHD version 4.0 or newer versus prior to version 4.0. For the former use `mender install` followed by the mender file; for the latter use `mender -f -rootfs` followed by the mender file. Starting with UHD version 4.0 one can use mender to upgrade or downgrade the UHD filesystem version between any UHD v4 versions (e.g., 4.1 to 4.6; 4.6 to 4.1). The following commands assume that the UHD filesystem is version 4; if not then substitute the other mender command.

Run `mender install /path/to/latest.mender` to update the file system, e.g.:

```
# mender install usrp_n3xx_fs.mender
```

The artifact can also be stored on a remote server:

```
# mender install http://server.name/path/to/latest.mender
```

This procedure will take a quite a few minutes to complete. While executing, mender will show progress, e.g.:

```
................................   0% 1024 KiB
...
................................   1% 4096 KiB
...
................................  99% 386048 KiB
............                     100% 386458 KiB
INFO[3865] wrote 7851737088/7851737088 bytes of update to device /dev/mmcblk0p3  module=device
INFO[3871] Enabling partition with new image installed to be a boot candidate: 3  module=device
```

After mender has logged a successful update, reboot the device:

```
# reboot
```

Upon reboot log back in to the USRP N3xx and run the command `uhd_find_devices` to verify that the UHD version is as desired and that the command runs successfully -- it should find at a minimum the USRP it is being executed on and will find more if other USRPs are on the same network.

If upon reboot the device is not working or the UHD version is not as desired, then the easiest way forward is to overwrite the sdcard filesystem manually with the desired UHD version.

If upon reboot everything is as desired and the device seems functional, then commit the changes so that the boot loader knows to permanently boot into this partition:

```
# mender commit
```

To identify the currently installed Mender artifact from the command line, the following file can be queried on the N3xx:

```
# cat /etc/mender/artifact_info
```

If you are using a Mender server, the updates can be initiated from a web dashboard. From there, you can start the updates without having to log into the device, and you can update groups of USRPs with a few clicks in a web GUI. The dashboard can also be used to inspect the state of USRPs. This is a simple way to update groups of rack-mounted USRPs with custom file systems.

For more information on updating the filesystem, refer to the UHD Manual?.

5.7.3

Please see the separate application note, Writing the USRP File System Disk Image to a SD Card, for step-by-step instructions on writing the file system image to the SD card.

5.8

The USRP N3xx systemd network configuration files are located at: `/data/network/`

```
# ls /data/network/
eth0.network  int0.network  sfp0.network  sfp1.network
```

or for older versions of the file system: `/etc/systemd/network/`

```
# ls /etc/systemd/network/
eth0.network  sfp0.network  sfp1.network
```

For details on configuration please refer to the systemd-networkd manual pages.

The factory settings are as follows:

```
eth0 (DHCP):

    [Match]
    Name=eth0

    [Network]
    DHCP=ipv4

    [DHCP]
    UseHostname=false

sfp0 (static):

    [Match]
    Name=sfp0

    [Network]
    Address=192.168.10.2/24
```

```
    [Link]
    MTUBytes=9000

sfp1 (static):

    [Match]
    Name=sfp1

    [Network]
    Address=192.168.20.2/24

    [Link]
    MTUBytes=9000
```

Additional notes on networking:

- Care needs to be taken when editing these files on the device, since `vi` / `vim` sometimes generates undo files (e.g. `/etc/systemd/network/sfp0.network~`), that `systemd-networkd` might accidentally pick up.
- Temporarily setting the IP addresses or MTU sizes via `ifconfig` or other command line tools will only change the value until the next reboot or reload of the FPGA image.
- If the MTU of the device and host computers differ, streaming issues can occur.
- Streaming via SFP0 at 1 Gb rates requires a MTU of `1500`
- Streaming via SFP0 at 10 Gb rates requires a MTU of `9000`

For addition details on network configuration here: https://files.ettus.com/manual/page_usrp_n3xx.html#n3xx_network_configuration

5.9

5.9.1

The FPGA image should match the version of UHD installed on the host computer, when operated in Network mode. Connect the device to the host computer using either the RJ45 or SFP+ port, refer to the section above for detailed instructions.

To obtain all the FPGA images for a specific version of UHD, run the following command on the host computer with internet access:

```
$ sudo uhd_images_downloader
```

Example Output:

```
$ sudo uhd_images_downloader
[INFO] Images destination: /usr/local/share/uhd/images
00006 kB / 00006 kB (100%) usrp1_b100_fw_default-g6bea23d.zip
19810 kB / 19810 kB (100%) x3xx_x310_fpga_default-gf1ba32fe.zip
02757 kB / 02757 kB (100%) usrp2_n210_fpga_default-g6bea23d.zip
02123 kB / 02123 kB (100%) n230_n230_fpga_default-ge57dfe0.zip
00522 kB / 00522 kB (100%) usrp1_b100_fpga_default-g6bea23d.zip
00491 kB / 00491 kB (100%) b2xx_b200_fpga_default-ge57dfe0.zip
02415 kB / 02415 kB (100%) usrp2_n200_fpga_default-g6bea23d.zip
08988 kB / 08988 kB (100%) e3xx_e320_fpga_default-g3de8954a.zip
23045 kB / 23045 kB (100%) n3xx_n310_fpga_default-g3de8954a.zip
00523 kB / 00523 kB (100%) b2xx_b205mini_fpga_default-ge57dfe0.zip
18937 kB / 18937 kB (100%) x3xx_x300_fpga_default-gf1ba32fe.zip
00017 kB / 00017 kB (100%) octoclock_octoclock_fw_default-g14000041.zip
00007 kB / 00007 kB (100%) usrp2_usrp2_fw_default-g6bea23d.zip
00009 kB / 00009 kB (100%) usrp2_n200_fw_default-g6bea23d.zip
00450 kB / 00450 kB (100%) usrp2_usrp2_fpga_default-g6bea23d.zip
00144 kB / 00144 kB (100%) b2xx_common_fw_default-ga69ab0c.zip
25107 kB / 25107 kB (100%) n3xx_n320_fpga_default-g3de8954a.zip
00464 kB / 00464 kB (100%) b2xx_b200mini_fpga_default-ge57dfe0.zip
00319 kB / 00319 kB (100%) usrp1_usrp1_fpga_default-g6bea23d.zip
04839 kB / 04839 kB (100%) usb_common_windrv_default-g14000041.zip
00009 kB / 00009 kB (100%) usrp2_n210_fw_default-g6bea23d.zip
16065 kB / 16065 kB (100%) n3xx_n300_fpga_default-g3de8954a.zip
05578 kB / 05578 kB (100%) e3xx_e310_fpga_default-g4bc2c6f.zip
00885 kB / 00885 kB (100%) b2xx_b210_fpga_default-ge57dfe0.zip
[INFO] Images download complete.
```

NOTE: In the above example output, the Images Destination folder is printed:

```
[INFO] Images destination: /usr/local/share/uhd/images
```

To list the N3xx FPGA images with a full path, run the command:

```
$ 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
```

To update the default `HG` variant of FPGA image, run the command:

```
$ uhd_image_loader --args "type=n3xx,addr=<N3xx_IP_ADDR>,fpga=HG"
```

Example Output:

```
uhd_image_loader --args "type=n3xx,addr=192.168.1.151,fpga=HG"
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.11.1.HEAD-0-gad6b0935
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.1.151,type=n3xx,product=n310,serial=313ABDA,claimed=False,
```

```
[INFO] [MPM.main] Launching USRP/MPM, version: 3.11.1.0-gunknown
[INFO] [MPM.main] Spawning RPC process...
[INFO] [MPM.PeriphManager] Device serial number: 313ABDA
[INFO] [MPM.PeriphManager] Found 2 daughterboard(s).
[INFO] [MPM.PeriphManager.UDP] No CHDR interfaces found!
[INFO] [MPM.PeriphManager.UDP] No CHDR interfaces found!
[INFO] [MPM.RPCServer] RPC server ready!
[INFO] [MPM.RPCServer] Spawning watchdog task...
[INFO] [MPM.PeriphManager.UDP] No CHDR interfaces found!
[INFO] [MPMD] Claimed device without full initialization.
[INFO] [MPMD IMAGE LOADER] Starting update. This may take a while.
[INFO] [MPM.PeriphManager] Updating component `fpga'
[INFO] [MPM.PeriphManager] Updating component `dts'
[INFO] [MPM.RPCServer] Resetting peripheral manager.
[INFO] [MPM.PeriphManager] Device serial number: 313ABDA
[INFO] [MPM.PeriphManager] Found 2 daughterboard(s).
[INFO] [MPMD IMAGE LOADER] Update component function succeeded.
```

To load a different default FPGA image (i.e. `XG`, `WG`), modify the device argument `fpga=` to a value of `fpga=XG` or `fpga=WG`.

To specify the path to a custom FPGA image, use the ?`--fpga-path`? argument.

```
$ uhd_image_loader --args "type=n3xx,addr=<N3xx_IP_ADDR>" --fpga-path=/path/to/custom/fpga.bit
```

The Verilog code for the FPGA in the USRP N3xx is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

5.9.2

It is possible to update the FPGA image when operated in Embedded mode. Connect to the ARM CPU via Serial Console or SSH as detailed in the section above.

Updating the FPGA image from the ARM CPU is the same as detailed above for a Network mode update, except it is not required to provide an `addr` device argument.

```
uhd_image_loader --args "type=n3xx,fpga=HG"
```

```
root@ni-n3xx-313ABDA:~# uhd_image_loader --args "type=n3xx,fpga=HG"
[INFO] [UHD] linux; GNU C++ version 7.2.0; Boost_106400; UHD_3.11.1.0-0-unknown
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=127.0.0.1,type=n3xx,product=n310,serial=313ABDA,claimed=False,skip_in
[INFO] [MPMD] Claimed device without full initialization.
[INFO] [MPMD IMAGE LOADER] Starting update. This may take a while.
[INFO] [MPM.PeriphManager] Updating component `fpga'
[INFO] [MPM.PeriphManager] Updating component `dts'
[INFO] [MPM.RPCServer] Resetting peripheral manager.
[INFO] [MPM.PeriphManager] Device serial number: 313ABDA
[INFO] [MPM.PeriphManager] Found 2 daughterboard(s).
[INFO] [MPMD IMAGE LOADER] Update component function succeeded.
```

For more information on updating the FPGA image, refer to the UHD Manual at http://uhd.ettus.com .

5.10

The device supports multiple, high-speed, low-latency interfaces on the SFP+ ports for streaming samples to the host computer.

5.10.1

Complete the steps below to set up a streaming connection over the 1 Gigabit Ethernet interface on `SFP Port 0`.

When streaming via SFP Port 0 at 1 Gb speeds, it is important that the connection is direct between the Host and USRP. Placing a switch or other network gear between the Host and USRP can reduce throughput of the transport link. It is also generally recommended to avoid using USB to Ethernet Adapters for the high speed streaming interface, as they may limit performance or cause periodic flow control errors.

NOTE: The `HG` FPGA image must be loaded for `SFP Port 0` to operate at 1Gb speeds. If the `XG` image is loaded, the port will be unresponsive at 1Gb speeds.

1. Configure your Host's Ethernet adapter as shown below. This interface should be separate from the 1Gb NIC/network which is connected to the 1Gb RJ45 management interface.

```
IP Address: 192.168.10.1
Subnet Mask: 255.255.255.0
Gateway: 0.0.0.0
MTU: 1500
```

NOTE: When operating `SFP Port 0` at 1Gb speeds, it is important to set a MTU of `1500` and not a value of `automatic`.

2. Insert the ? RJ45 ? SFP+ adapter ?into? `SFP Port 0`? .

3. Connect the adapter to a host computer using the Ethernet cable to SFP0.

The ? Green LED? above ?`SFP Port 0`? should illuminate.

4. To test the connection,? ?`ping`? the device at address `192.168.10.2?` from the host, as shown below:

```
$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=1.06 ms
^C
--- 192.168.10.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.065/1.065/1.065/0.000 ms
```

Press `CTRL+C` to stop the ping program.

Proceed to the next section "Verifying Device Operation".

5.10.2

Complete the steps below to set up a streaming connection over the 10 Gigabit Ethernet interface on `SFP Port 1`.

NOTE: Both the `HG` and `XG` FPGA images support 10Gb speeds over SFP Port 1.

1. Configure your Host's 10Gb Ethernet adapter as shown below.

```
IP Address: 192.168.20.1
Subnet Mask: 255.255.255.0
Gateway: 0.0.0.0
MTU: 9000
```

NOTE: When operating at 10Gb speeds, it is important to set a MTU of `9000` and not a value of `automatic`.

2. Connect the USRP to a host computer using either a 10Gb SFP or Fiber cable to `SFP Port 1`.

The ? Green LED? above ?`SFP Port 1`? should illuminate.

3. To test the connection,? ?`ping`? the device at address `192.168.20.2?` from the host, as shown below:

```
$ ping 192.168.20.2
```

Press `CTRL+C` to stop the ping program.

Proceed to the next section "Verifying Device Operation".

5.10.3

Complete the steps below to set up a streaming connections over the Dual 10 Gigabit Ethernet interface on `SFP Ports 0/1`.

NOTE: The `XG` FPGA image must be loaded for `SFP Port 0` to operate at 10 Gb speeds. If the `HG` image is loaded, the port will be unresponsive at 10 Gb speeds.

1. Configure your Host's #1 10Gb Ethernet adapter as shown below.

```
IP Address: 192.168.10.1
Subnet Mask: 255.255.255.0
Gateway: 0.0.0.0
MTU: 9000
```

2. Configure your Host's #2 10Gb Ethernet adapter as shown below.

```
IP Address: 192.168.20.1
Subnet Mask: 255.255.255.0
Gateway: 0.0.0.0
MTU: 9000
```

NOTE: When operating at 10Gb speeds, it is important to set a MTU of `9000` and not a value of `automatic`.

3. Connect the USRP to a host computer using either a 10Gb SFP or Fiber cables to `SFP Ports 0/1`.

The ?Green LEDs? above ?`SFP Ports 0/1`? should illuminate.

4. To test the `SFP Port 0` connection,? ?`ping`? the device at address `192.168.10.2?` from the host, as shown below:

```
$ ping 192.168.10.2
```

Press `CTRL+C` to stop the ping program.

5. To test the `SFP Port 1` connection,? ?`ping`? the device at address `192.168.20.2?` from the host, as shown below:

```
$ ping 192.168.20.2
```

Press `CTRL+C` to stop the ping program.

Proceed to the next section "Verifying Device Operation".

For more details on Network Setup and Configuration, please see the ?Interfaces and Connectivity? section on the N300/N310 or N320/N321 hardware resources pages.

5.11

Once you have successfully setup a management interface and streaming interface, you can now verify the devices operation using the included UHD utilities.

5.11.1

5.11.1.1

The USRP N300 contains 2 channels, each represented on the front panel as `RF0-1`. Below is the `subdev` mapping of RF Ports.

- RF0 = A:0
- RF1 = A:1

5.11.1.2

The USRP N310 contains 4 channels, each represented on the front panel as `RF0-3`. Below is the `subdev` mapping of RF Ports.

5.11.1.2.1

- RF0 = A:0
- RF1 = B:0
- RF2 = C:0
- RF3 = D:0

5.11.1.2.2

- RF0 = A:0
- RF1 = A:1
- RF2 = B:0
- RF3 = B:1

5.11.1.3

The USRP N320 contains 2 channels, each represented on the front panel as `RF0-1`. Below is the `subdev` mapping of RF Ports.

- RF0 = A:0
- RF1 = B:0

5.11.1.4

The USRP N321 contains 2 channels, each represented on the front panel as `RF0-1`. Below is the `subdev` mapping of RF Ports.

- RF0 = A:0
- RF1 = B:0

Additional details of UHD Subdevice Specifications can be found here in the UHD Manual:
http://files.ettus.com/manual/page_configuration.html#config_subdev

5.11.2

The USRP N300/N310 supports the three fixed Master Clock Rates listed below.

- 122.88 MHz
- 125.00 MHz
- 153.60 MHz

The USRP N320/N321 supports the three fixed Master Clock Rates listed below.

- 200.00 MHz
- 245.76 MHz
- 250.00 MHz

Sample rates as delivered to/from the host computer for USRP devices are constrained to follow several important rules.

It is important to understand that strictly-integer decimation and interpolation are used within USRP hardware to meet the requested sample rate requirements of the application at hand. That means that the desired sample rate must meet the requirement that master-clock-rate/desired-sample-rate be an integer ratio. Further, it is strongly desirable for that ratio to be even. This ratio is the decimation (down-conversion) or interpolation (up-conversion) factor. The decimation or interpolation factor may be between 1 and 1024. There are further constraints on the decimation or interpolation factor. If the decimation or interpolation factor exceeds 128, then it must be evenly divisible by 2. If the decimation or interpolation factor exceeds 256, then it must be evenly divisible by 4.

5.11.2.1

Listed below are common sample rates for the given master clock rates. This is not a complete listing of the supported sample rates.

Master Clock Rate	**Decimation / Interpolation Rate Host Sample Rate [Msps]**														
1	2	4	6	8	10	12	14	16	18	20	30	32	64	10
122.88e6	61.44e6	30.72e6	20.48e6	15.36e6	12.288e6	10.24e6	8.7771e6	7.68e6	6.8267e6	6.144e6	4.096e6	3.84e6	1.92e6	1.2
125e6	62.5e6	31.25e6	20.833e6	15.625e6	12.5e6	10.417e6	8.9286e6	7.8125e6	6.9444e6	6.25e6	4.1667e6	3.90625e6	1.953125e6	1.2
153.6e6	76.8e6	38.4e6	25.6e6	19.2e6	15.36e6	12.8e6	10.971e6	9.6e6	8.5333e6	7.68e6	5.12e6	4.8e6	2.4e6	1.5

5.11.2.2

Listed below are common sample rates for the given master clock rates. This is not a complete listing of the supported sample rates.

Master Clock Rate	**Decimation / Interpolation Rate Host Sample Rate [Msps]**													
1	2	4	6	8	10	12	14	16	18	20	30	32	64	100
200e6	100e6	50e6	33.33e6	25e6	20e6	16.66e6	14.2857e6	12.5e6	11.11e6	10e6	6.667e6	6.25e6	3.125e6	2e6
245.76e6	122.88e6	61.44e6	30.72e6	20.48e6	15.36e6	12.288e6	10.24e6	8.7771e6	7.68e6	6.8267e6	6.144e6	4.096e6	3.84e6	1.92
250e6	125e6	62.5e6	31.25e6	20.833e6	15.625e6	12.5e6	10.417e6	8.9286e6	7.8125e6	6.9444e6	6.25e6	4.1667e6	3.90625e6	1.95

Additional information on Sample Rates can be found here in the UHD Manual:
http://files.ettus.com/manual/page_general.html#general_sampleratenotes

5.11.3

5.11.3.1

The UHD utility `uhd_usrp_probe` provides detailed information of the USRP device.

From your host computer, run the command `uhd_usrp_probe`:

```
$  uhd_usrp_probe
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.13.1.HEAD-0-ga0a71d10
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.10.2,type=n3xx,product=n310,serial=313ABDA,claimed=False,addr
[INFO] [MPM.main] Launching USRP/MPM, version: 3.13.1.0-gd3b7e90a
[INFO] [MPM.main] Spawning RPC process...
[INFO] [MPM.PeriphManager] Device serial number: 313ABDA
[INFO] [MPM.PeriphManager] Initialized 2 daughterboard(s).
[INFO] [MPM.PeriphManager] init() called with device args `time_source=internal,clock_source=internal'.
[INFO] [MPM.RPCServer] RPC server ready!
[INFO] [MPM.RPCServer] Spawning watchdog task...
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000004)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1355 MB/s)
[INFO] [MPM.PeriphManager] init() called with device args `mgmt_addr=192.168.10.2,clock_source=internal,time_source=internal,product=n310'.
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1358 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1355 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1345 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000011312)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000011312)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000002)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000002)
  ____________________________________________________
 /
|       Device: N300-Series Device
|     ______________________________________________________
|    /
|   |       Mboard: ni-n3xx-313ABDA
|   |   eeprom_version: 1
|   |   mpm_version: 3.13.1.0-gd3b7e90a
|   |   pid: 16962
|   |   product: n310
|   |   rev: 3
|   |   rpc_connection: remote
|   |   serial: 313ABDA
|   |   type: n3xx
|   |   MPM Version: 1.2
|   |   FPGA Version: 5.2
|   |   RFNoC capable: Yes
|   |
|   |   Time sources:  internal, external, gpsdo, sfp0
|   |   Clock sources: external, internal, gpsdo
|   |   Sensors: gps_tpv, ref_locked, gps_time, gps_locked, temp, gps_sky, fan
|   |     ____________________________________________________
|   |    /
|   |   |       RX Dboard: A
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX, RX2, CAL, LOCAL
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 75.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 1
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX, RX2, CAL, LOCAL
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 75.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Codec: A
|   |   |   |   Name: AD9371 Dual ADC
|   |   |   |   Gain Elements: None
|   |   |
|   |     ____________________________________________________
|   |    /
|   |   |       RX Dboard: B
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX, RX2, CAL, LOCAL
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 75.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 1
|   |   |   |   Name: Magnesium
```

```
|   |   |   |   Antennas: TX/RX, RX2, CAL, LOCAL
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 75.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |   |
|   |   |   |
|   |   |   |       RX Codec: B
|   |   |   |   Name: AD9371 Dual ADC
|   |   |   |   Gain Elements: None
|   |   |
|   |   |       TX Dboard: A
|   |   |
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 65.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |   |
|   |   |   |       TX Frontend: 1
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 65.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |   |
|   |   |   |       TX Codec: A
|   |   |   |   Name: AD9371 Dual DAC
|   |   |   |   Gain Elements: None
|   |   |
|   |   |       TX Dboard: B
|   |   |
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 65.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |   |
|   |   |   |       TX Frontend: 1
|   |   |   |   Name: Magnesium
|   |   |   |   Antennas: TX/RX
|   |   |   |   Sensors: lo_locked, ad9371_lo_locked, lowband_lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 65.0 step 0.5 dB
|   |   |   |   Gain range rfic: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range dsa: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Gain range amp: 0.0 to 0.0 step 0.0 dB
|   |   |   |   Bandwidth range: 20000000.0 to 100000000.0 step 0.0 Hz
|   |   |   |   Connection Type: IQ
|   |   |   |   Uses LO offset: No
|   |   |   |
|   |   |   |       TX Codec: B
|   |   |   |   Name: AD9371 Dual DAC
|   |   |   |   Gain Elements: None
|   |   |
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * DDC_0
|   |   |   * DDC_1
|   |   |   * DUC_0
|   |   |   * DUC_1
```

5.11.3.2

```
$ uhd_usrp_probe
[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_106600; UHD_3.14.0.0-0-g6875d061
```

```
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=127.0.0.1,type=n3xx,product=n320,serial=3181FFA,claimed=False
[INFO] [MPM.main] Launching USRP/MPM, version: 3.14.0.0-g6875d061
[INFO] [MPM.main] Spawning RPC process...
[INFO] [MPM.PeriphManager] Device serial number: 3181FFA
[INFO] [MPM.Rhodium-0] Successfully loaded all peripherals!
[INFO] [MPM.Rhodium-1] Successfully loaded all peripherals!
[INFO] [MPM.PeriphManager] Initialized 2 daughterboard(s).
[INFO] [MPM.PeriphManager] No QSFP board detected: Assuming it is disabled in the device tree overlay (e.g., HG, XG images).
[INFO] [MPM.PeriphManager] init() called with device args `time_source=internal,clock_source=internal'.
[INFO] [MPM.Rhodium-0] init() called with args `time_source=internal,clock_source=internal'
[INFO] [MPM.Rhodium-1] init() called with args `time_source=internal,clock_source=internal'
[INFO] [MPM.Rhodium-0.init.LMK04828] LMK initialized and locked!
[INFO] [MPM.Rhodium-1.init.LMK04828] LMK initialized and locked!
[INFO] [MPM.Rhodium-1.DAC37J82] DAC PLL Locked!
[INFO] [MPM.Rhodium-1.AD9695] ADC PLL Locked!
[INFO] [MPM.Rhodium-1.init] JESD204B Link Initialization & Training Complete
[INFO] [MPM.Rhodium-0.DAC37J82] DAC PLL Locked!
[INFO] [MPM.Rhodium-0.AD9695] ADC PLL Locked!
[INFO] [MPM.Rhodium-0.init] JESD204B Link Initialization & Training Complete
[INFO] [MPM.RPCServer] RPC server ready!
[INFO] [MPM.RPCServer] Spawning watchdog task...
[INFO] [MPM.PeriphManager] init() called with device args `mgmt_addr=127.0.0.1,clock_source=internal,time_source=internal,product=n320'.
[INFO] [MPM.Rhodium-0] init() called with args `mgmt_addr=127.0.0.1,clock_source=internal,time_source=internal,product=n320'
[INFO] [MPM.Rhodium-1] init() called with args `mgmt_addr=127.0.0.1,clock_source=internal,time_source=internal,product=n320'
[INFO] [0/Replay_0] Initializing block control (NOC ID: 0x4E91A00000000004)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000000320)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000320)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/FIFO_0] Initializing block control (NOC ID: 0xF1F0000000000000)
[INFO] [0/FIFO_1] Initializing block control (NOC ID: 0xF1F0000000000000)
 ____________________________________________________
/
|       Device: N300-Series Device
|     ______________________________________________________
|    /
|   |       Mboard: ni-n3xx-3181FFA
|   |   eeprom_version: 2
|   |   mpm_version: 3.14.0.0-g6875d061
|   |   pid: 16962
|   |   product: n320
|   |   rev: 6
|   |   rpc_connection: local
|   |   serial: 3181FFA
|   |   type: n3xx
|   |   MPM Version: 1.2
|   |   FPGA Version: 5.3
|   |   FPGA git hash: 3de8954.clean
|   |   RFNoC capable: Yes
|   |
|   |   Time sources:  internal, external, gpsdo, sfp0
|   |   Clock sources: external, internal, gpsdo
|   |   Sensors: gps_tpv, temp, gps_sky, fan, gps_time, gps_locked, ref_locked, gps_gpgga
|   |     ____________________________________________________
|   |    /
|   |   |       RX Dboard: A
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 3175A79
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: Rhodium
|   |   |   |   Antennas: TX/RX, RX2, CAL, TERM
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 60.0 step 1.0 dB
|   |   |   |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|   |   |   |   Connection Type:
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Codec: A
|   |   |   |   Name: ad9695-625
|   |   |   |   Gain Elements: None
|   |   |     ____________________________________________________
|   |    /
|   |   |       RX Dboard: B
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 3175A67
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: Rhodium
|   |   |   |   Antennas: TX/RX, RX2, CAL, TERM
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 60.0 step 1.0 dB
|   |   |   |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|   |   |   |   Connection Type:
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Codec: B
|   |   |   |   Name: ad9695-625
|   |   |   |   Gain Elements: None
|   |   |     ____________________________________________________
|   |    /
|   |   |       TX Dboard: A
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 3175A79
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: Rhodium
|   |   |   |   Antennas: TX/RX, CAL, TERM
```

```
|    |    |    |   Sensors: lo_locked
|    |    |    |   Freq range: 1.000 to 6000.000 MHz
|    |    |    |   Gain range all: 0.0 to 60.0 step 1.0 dB
|    |    |    |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|    |    |    |   Connection Type:
|    |    |    |   Uses LO offset: No
|    |    |    |     ____________________________________________________
|    |    |    |    /
|    |    |    |       TX Codec: A
|    |    |    |   Name: dac37j82
|    |    |    |   Gain Elements: None
|    |    |     ________________________________________________________
|    |    |    /
|    |    |       TX Dboard: B
|    |    |   ID: Unknown (0x0152)
|    |    |   Serial: 3175A67
|    |    |     ____________________________________________________
|    |    |    /
|    |    |    |       TX Frontend: 0
|    |    |    |   Name: Rhodium
|    |    |    |   Antennas: TX/RX, CAL, TERM
|    |    |    |   Sensors: lo_locked
|    |    |    |   Freq range: 1.000 to 6000.000 MHz
|    |    |    |   Gain range all: 0.0 to 60.0 step 1.0 dB
|    |    |    |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|    |    |    |   Connection Type:
|    |    |    |   Uses LO offset: No
|    |    |    |     ____________________________________________________
|    |    |    /
|    |    |    |       TX Codec: B
|    |    |    |   Name: dac37j82
|    |    |    |   Gain Elements: None
|    |    |     ________________________________________________________
|    |    |    /
|    |    |       RFNoC blocks on this device:
|    |    |
|    |    |   * Replay_0
|    |    |   * Radio_0
|    |    |   * Radio_1
|    |    |   * DDC_0
|    |    |   * DDC_1
|    |    |   * DUC_0
|    |    |   * DUC_1
|    |    |   * FIFO_0
|    |    |   * FIFO_1
```

5.11.3.3

```
$ 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
[INFO] [MPM.PeriphManager] init() called with device args `time_source=internal,clock_source=internal,product=n320,mgmt_addr=192.168.20.2'.
[INFO] [MPM.Rhodium-0] init() called with args `time_source=internal,clock_source=internal,product=n320,mgmt_addr=192.168.20.2'
[INFO] [MPM.Rhodium-1] init() called with args `time_source=internal,clock_source=internal,product=n320,mgmt_addr=192.168.20.2'
[INFO] [0/Replay_0] Initializing block control (NOC ID: 0x4E91A00000000004)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000000320)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000320)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/FIFO_0] Initializing block control (NOC ID: 0xF1F0000000000000)
[INFO] [0/FIFO_1] Initializing block control (NOC ID: 0xF1F0000000000000)
  ____________________________________________________
 /
|       Device: N300-Series Device
|     ______________________________________________________
|    /
|   |       Mboard: ni-n3xx-3166646
|   |   eeprom_version: 2
|   |   mpm_version: 3.14.0.0-g6875d061
|   |   pid: 16962
|   |   product: n320
|   |   rev: 6
|   |   rpc_connection: remote
|   |   serial: 3166646
|   |   type: n3xx
|   |   MPM Version: 1.2
|   |   FPGA Version: 5.3
|   |   FPGA git hash: 3de8954.clean
|   |   RFNoC capable: Yes
|   |
|   |   Time sources:  internal, external, gpsdo, sfp0
|   |   Clock sources: external, internal, gpsdo
|   |   Sensors: gps_sky, gps_time, gps_gpgga, gps_locked, fan, gps_tpv, ref_locked, temp
|   |     ________________________________________________________
|   |    /
|   |   |       RX Dboard: B
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 316D814
|   |   |     ____________________________________________________
|   |   |    /
|   |   |    |       RX Frontend: 0
|   |   |    |   Name: Rhodium
|   |   |    |   Antennas: TX/RX, RX2, CAL, TERM
|   |   |    |   Sensors: lo_locked
|   |   |    |   Freq range: 1.000 to 6000.000 MHz
|   |   |    |   Gain range all: 0.0 to 60.0 step 1.0 dB
|   |   |    |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|   |   |    |   Connection Type:
|   |   |    |   Uses LO offset: No
|   |   |    |     ____________________________________________________
```

```
|   |   |    /
|   |   |   |       RX Codec: B
|   |   |   |   Name: ad9695-625
|   |   |   |   Gain Elements: None
|   |     ____________________________________________________
|   |    /
|   |   |       RX Dboard: A
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 316D810
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Frontend: 0
|   |   |   |   Name: Rhodium
|   |   |   |   Antennas: TX/RX, RX2, CAL, TERM
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 60.0 step 1.0 dB
|   |   |   |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|   |   |   |   Connection Type:
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       RX Codec: A
|   |   |   |   Name: ad9695-625
|   |   |   |   Gain Elements: None
|   |     ____________________________________________________
|   |    /
|   |   |       TX Dboard: B
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 316D814
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: Rhodium
|   |   |   |   Antennas: TX/RX, CAL, TERM
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 60.0 step 1.0 dB
|   |   |   |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|   |   |   |   Connection Type:
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Codec: B
|   |   |   |   Name: dac37j82
|   |   |   |   Gain Elements: None
|   |     ____________________________________________________
|   |    /
|   |   |       TX Dboard: A
|   |   |   ID: Unknown (0x0152)
|   |   |   Serial: 316D810
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Frontend: 0
|   |   |   |   Name: Rhodium
|   |   |   |   Antennas: TX/RX, CAL, TERM
|   |   |   |   Sensors: lo_locked
|   |   |   |   Freq range: 1.000 to 6000.000 MHz
|   |   |   |   Gain range all: 0.0 to 60.0 step 1.0 dB
|   |   |   |   Bandwidth range: 250000000.0 to 250000000.0 step 0.0 Hz
|   |   |   |   Connection Type:
|   |   |   |   Uses LO offset: No
|   |   |     ____________________________________________________
|   |   |    /
|   |   |   |       TX Codec: A
|   |   |   |   Name: dac37j82
|   |   |   |   Gain Elements: None
|   |     ____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * Replay_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * DDC_0
|   |   |   * DDC_1
|   |   |   * DUC_0
|   |   |   * DUC_1
|   |   |   * FIFO_0
|   |   |   * FIFO_1
```

If you see warnings such as:

```
[WARNING] [UDP] The recv buffer could not be resized sufficiently.
```

You need to resize the socket buffers for your network interface card:

```
sudo sysctl -w net.core.rmem_max=288000
sudo sysctl -w net.core.wmem_max=288000
sudo sysctl -w net.core.rmem_max=33554432
```

5.11.4

The UHD driver includes several example programs, which may serve as test programs or the basis for your application program. The source code can be obtained from the UHD repository on github at: https://github.com/EttusResearch/uhd/tree/master/host/examples

You can quickly verify the operation of your USRP N3xx by running the `rx_ascii_art_dft` UHD example program.

The `rx_ascii_art_dft` utility is a simple console –based, real-time FFT display tool. It is not graphical in nature, so it can be easily run over an SSH connection within a terminal window, and does not need any graphical capability, such as X Windows, to be installed. It can also be run over a serial console connection, although this is not recommended, as the formatting may not render correctly.

You can run a simple test of the N3xx USRP by connecting an antenna and observing the spectrum of a commercial FM radio station in real-time, following the steps below:

1. Attach an antenna to the `Ch0/RX2`– antenna port of the N3xx.

2. From your host computer, run the command:

N300/N310

```
$ /usr/local/lib/uhd/examples/rx_ascii_art_dft --args "master_clock_rate=125e6,mgmt_addr=192.168.1.151,addr=192.168.10.2" --freq 98.5e6 --rat
```

N320/N321

```
$ /usr/local/lib/uhd/examples/rx_ascii_art_dft --args "master_clock_rate=250e6,mgmt_addr=192.168.1.151,addr=192.168.10.2" --freq 98.5e6 --rat
```

NOTE: Modify the command– line argument `freq` ?above to specify a tuning frequency for a strong local FM radio station. You will also need to update the IP Address to match your device IP.

3. You should see a real-time FFT display of 2.5 MHz of spectrum, centered at the specified tuning frequency.

4. Type "`Q`" or `Ctrl--C` to stop the program and to return to the Linux command line.

5. You can run with the `?----help` ?argument to see a description of all available command-line options.

Example Output:

```
$ /usr/local/lib/uhd/examples/rx_ascii_art_dft --args "master_clock_rate=125e6,mgmt_addr=192.168.1.151,addr=192.168.10.2" --freq 98.5e6 --rat

Creating the usrp device with: master_clock_rate=125e6,mgmt_addr=192.168.1.151,addr=192.168.10.2...
[INFO] [UHD] linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_3.11.1.HEAD-0-gad6b0935
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=192.168.1.151,type=n3xx,product=n310,serial=313ABDA,claimed=False,mas
[INFO] [MPM.main] Launching USRP/MPM, version: 3.11.1.0-gunknown
[INFO] [MPM.main] Spawning RPC process...
[INFO] [MPM.PeriphManager] Device serial number: 313ABDA
[INFO] [MPM.PeriphManager] Found 2 daughterboard(s).
[INFO] [MPM.RPCServer] RPC server ready!
[INFO] [MPM.RPCServer] Spawning watchdog task...
[INFO] [MPM.PeriphManager] init() called with device args `mgmt_addr=192.168.1.151,product=n310,master_clock_rate=125e6'.
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000004)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1336 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1338 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1346 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1350 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000000310)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000310)
[INFO] [0/Radio_2] Initializing block control (NOC ID: 0x12AD100000000310)
[INFO] [0/Radio_3] Initializing block control (NOC ID: 0x12AD100000000310)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DDC_2] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DDC_3] Initializing block control (NOC ID: 0xDDC0000000000001)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_2] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_3] Initializing block control (NOC ID: 0xD0C0000000000000)
Using Device: Single USRP:
  Device: N300-Series Device
  Mboard 0: ni-n3xx-313ABDA
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: Magnesium
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: Magnesium
  TX Channel: 1
    TX DSP: 0
    TX Dboard: B
    TX Subdev: Magnesium
  TX Channel: 2
    TX DSP: 0
    TX Dboard: C
    TX Subdev: Magnesium
  TX Channel: 3
    TX DSP: 0
    TX Dboard: D
    TX Subdev: Magnesium

Setting RX Rate: 2.500000 Msps...
Actual RX Rate: 2.500000 Msps...

Setting RX Freq: 98.500000 MHz...
Actual RX Freq: 98.500000 MHz...

Setting RX Gain: 50.000000 dB...
Actual RX Gain: 50.000000 dB...

Checking RX: all_los: locked ...

Done!
```

5.11.5

Included with the UHD driver example programs is a utility, `benchmark_rate` to benchmark the transport link of the system.

A system's maximum performance is dependent upon many factors. `benchmark_rate` will exercise the transport link and CPU of the system.

5.11.5.1

NOTE: This example requires the `HG` FPGA image to be loaded.

N300/N310

This example will test one full-duplex stream using "RF0/A:0", at a rate of 3.84 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.10.2,master_clock_rate=122.88e6" \
--duration 60 \
--channels "0" \
--rx_rate 3.84e6 \
--rx_subdev "A:0" \
--tx_rate 3.84e6 \
--tx_subdev "A:0"
```

N310

This example will test four full-duplex streams at 1.25 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.10.2,master_clock_rate=125e6" \
--duration 60 \
--channels "0,1,2,3" \
--rx_rate 1.25e6 \
--rx_subdev "A:0 A:1 B:0 B:1" \
--tx_rate 1.25e6 \
--tx_subdev "A:0 A:1 B:0 B:1"
```

N320/N321

This example will test one full-duplex stream using "RF0/A:0", at a rate of 3.84 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.10.2,master_clock_rate=245.76e6" \
--duration 60 \
--channels "0" \
--rx_rate 3.84e6 \
--rx_subdev "A:0" \
--tx_rate 3.84e6 \
--tx_subdev "A:0"
```

When streaming samples over a 1 Gb transport link, the maximum accumulative rate for all channels is 25 MS/s with a `sc16` OTW format. To achieve higher streaming rates, it is recommended to use the 10 Gb interfaces.

5.11.5.2

NOTE: This example will work with either the `HG` or `XG` FPGA image.

N300/N310

This example will test one full-duplex stream using "RF0/A:0", at a rate of 31.25 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.20.2,master_clock_rate=125e6" \
--duration 60 \
--channels "0" \
--rx_rate 31.25e6 \
--rx_subdev "A:0" \
--tx_rate 31.25e6 \
--tx_subdev "A:0"
```

N320/N321

This example will test one full-duplex stream using "RF0/A:0", at a rate of 31.25 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.20.2,master_clock_rate=250e6" \
--duration 60 \
--channels "0" \
--rx_rate 31.25e6 \
--rx_subdev "A:0" \
--tx_rate 31.25e6 \
--tx_subdev "A:0"
```

N310

This example will test four full-duplex streams at 30.72 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.20.2,master_clock_rate=122.88e6" \
--duration 60 \
--channels "0,1,2,3" \
--rx_rate 30.72e6 \
--rx_subdev "A:0 A:1 B:0 B:1" \
--tx_rate 30.72e6 \
--tx_subdev "A:0 A:1 B:0 B:1"
```

N320/N321

This example will test two full-duplex streams at 30.72 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.20.2,master_clock_rate=245.76e6" \
--duration 60 \
--channels "0,1,2,3" \
--rx_rate 30.72e6 \
```

```
--rx_subdev "A:0 B:0" \
--tx_rate 30.72e6 \
--tx_subdev "A:0 B:0"
```

5.11.5.3

NOTE: This example requires the `XG` FPGA image to be loaded.

N310

This example will test four full-duplex streams at 62.5 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.10.2,second_addr=192.168.20.2,master_clock_rate=125e6" \
--duration 60 \
--channels "0,1,2,3" \
--rx_rate 62.5e6 \
--rx_subdev "A:0 A:1 B:0 B:1" \
--tx_rate 62.5e6 \
--tx_subdev "A:0 A:1 B:0 B:1"
```

N320/N321

This example will test two full-duplex streams at 62.5 MS/s, for 60 seconds:

```
/usr/local/lib/uhd/examples/benchmark_rate  \
--args "type=n3xx,mgmt_addr=192.168.1.151,addr=192.168.10.2,second_addr=192.168.20.2,master_clock_rate=250e6" \
--duration 60 \
--channels "0,1,2,3" \
--rx_rate 62.5e6 \
--rx_subdev "A:0 B:0" \
--tx_rate 62.5e6 \
--tx_subdev "A:0 B:0"
```

5.12

5.12.1

- Using Ethernet-Based Synchronization on the USRP? N3xx Devices

5.12.2

- USRP N320/N321 LO Distribution
- 5G NR EVM Measurements with the USRP N320/N321

5.12.3

To avoid damaging the file system and causing any corruption, do not turn the device off with the power button without first shutting down the system. Use this command to cleanly and properly shut the system down:

```
shutdown --h now
```

5.12.4

Auto booting of the N3xx when power is applied can be configured by enabling the flag on the device's EEPROM with the following command:

```
eeprom-set-flags 0x1
```

5.12.5

The default user is `root` and the password is empty (no password).

It is recommended to update the `root` password, which can be done with the command `passwd`:

Example Output:

```
root@ni-n3xx-serial:~# passwd
Changing password for root
New password:
Re-enter new password:
passwd: password changed.
```

5.13

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

5.14

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

5.15

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

5.16

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

6 X310 Getting Started Guides

6.1

- USRP X300/X310
- 1 Gigabit Ethernet Cable
- SFP Adapter for 1 GigE
- Power Supply and US Cord
- USB 2.0 JTAG Debug Cable
- Four SMA-Bulkhead Cables
- Getting Started Guide

6.2

Make sure that your kit contains all the items listed above. If any items are missing, please contact your sales agent or Ettus Research Technical support immediately.

6.3

- A host computer with an available 1 GigE, 10 GigE, or PCI-Express port

6.4

All Ettus Research products are individually tested before shipment. The USRP? is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP? can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

6.5

The USRP X300/X310 and compatible RF daughterboards are shipped separately. To operate the device you will need to install the RF daughterboards and supplied bulkhead cables. This can be accomplished by removing the top plate of the USRP X300/X310, which is secured with two screws. After installation, the daughterboards and cables should be secured with the hardware provided. The device must be powered off when installing daughterboards to avoid potential damage. For a detailed step by step guide to assembling your X300/X310, see the USRP X Series Quick Start (Daughterboard Installation) guide.

6.6

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on your host computer. A step-by-step guide for doing this is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows Application Notes. Release 3.8.4 or later of the USRP Hardware Driver, UHD, is required. It is recommended to use the latest stable version of UHD that is available.

If you have a USB stick with the Live SDR Environment installed on it, then you may boot your host computer from that. The LiveUSB SDR Environment does not require anything to be installed on your host computer, and contains a Linux-based environment with the UHD software and the GNU Radio framework already installed. More information about the Live SDR Environment is available at the Live SDR Environment Getting Started Guides page.

6.7

This USRP X300/X310 supports multiple, high-speed, low-latency interface options. This kit includes all of the components necessary to communicate with the device through a 1 Gigabit Ethernet interface. To setup the device, follow these basic instructions:

- Configure the host Ethernet adapter to use an IP address of "192.168.10.1" and a subnet mask of "255.255.255.0".
- Slide the SFP Adapter into the SFP "Port 0".
- Using the supplied Ethernet cable, connect the adapter to a host computer.
- The Green Led above SFP "Port 0" should illuminate.
- To test communications, ping the USRP device at address "192.168.10.2"

For more detailed on network setup, please see the section Interfaces and Connectivity of the X300/X310 Hardware Resources page. For help troubleshooting connectivity issues with your X3x0 device, see Troubleshooting X300/X310 Device Discovery Issues.

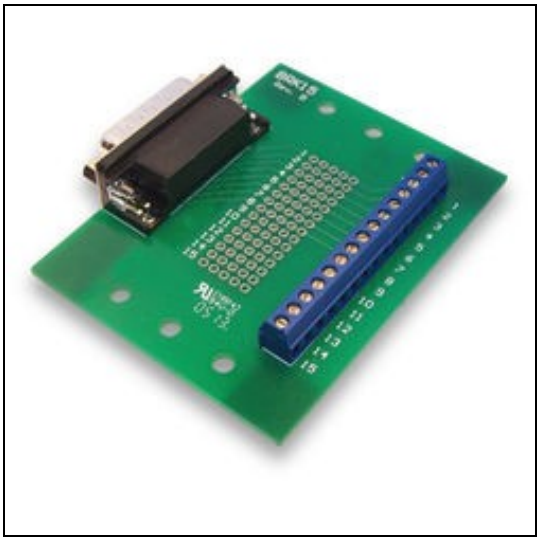
6.8

The USRP X300/X310 also supports 10 Gigabit Ethernet and PCI-Express. To use these interfaces, we recommend our high-speed interface kits. For more information about these interface kits, please see the section Interfaces and Connectivity of the X300/X310 Hardware Resources page.

6.9

Once the software tools are installed on the host computer, or using the Live SDR Environment, verify the correct operation of the USRP by running the utility programs on the host computer. More information is available at the Verifying the Operation of the USRP Using UHD and GNU Radio Application Note.

6.10



This General Purpose Input/output (GPIO) breakout kit provides access to general purpose digital I/O signals with simple terminal blocks, and a prototyping area where wires and components can be soldered. Each GPIO pin is connected to an FPGA digital line allowing it to be configured as an input, or an output, using the various software frameworks that support the USRP? GPIO.

These GPIO signals can serve the following functions:

- Control of external devices, such as power amplifiers and RF switches
- Provide output signals that can help with debugging
- Provide observables to be analyzed by oscilloscopes or other external equipment
- Accept input from external devices for local, software-based triggering
- Implement a protocol line such as SPI or I2C

6.10.1

- 1 GPIO Breakout Board
- 1 DB-15, 1-meter cable
- GPIO Quick Reference

6.10.2

The GPIO signals exposed with this breakout kit are routed directly to the USRP device's FPGA with limited protection circuitry. However, the user must take precautionary measures to ensure input/output signals meet the specifications shown in this document. Over voltage, excess current draw, and other conditions can damage the USRP device and void the warranty. Special care should be taken when the USRP is powered off.

6.10.3

The GPIO breakout board can be mounted directly to the DB15 connector of a USRP ? device, or mounted remotely with the cable provided in this kit. The screws on the DB15 connector of the breakout board must be removed to mount the board directly. For remote mounting, the breakout board is supplied with rubber standoffs to avoid scratching surfaces, and several through-holes for hard mounting with screws or other hardware (not provided).

6.10.4

When used with UHD, or other third party frameworks that leverage UHD, the GPIO expansion can be controlled with simple API calls. For more information, on the C++ API, and examples of how to use the GPIO in frameworks such as GNU Radio, please see the Application Notes section of the Ettus Research Knowledge Base.

6.10.5

Parameter	Typical
Configured as Input	
Default Voltage Standard	3.3V LVCMOS
Voltage High Threshold	2.0V
Voltage Low Threshold	0.8V
Voltage Input Limits (no damage)	-0.3V/3.45V

Configured as Output	
Voltage Standard	3.3V LVCMOS
Voltage High Output	2.8V
Voltage Low Output	0.4V
Current Source Capability	12 mA
Output Source Impedance	>33 ohms typical

6.11

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

6.12

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

6.13

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?. Please be sure to include your order number and the serial number of your USRP.

6.14

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

7 USRP-2974 Getting Started Guide

7.1

- NI USRP-2974
- 30 dB SMA Attenuator
- SMA-male to SMA-male Cable
- Power Supply
- Getting Started Guide

7.2

Make sure that your kit contains all the items listed above. If any items are missing, please contact your sales agent.

7.3

1. To prevent electrostatic discharge (ESD) from damaging the device, ground yourself using a grounding strap or by holding a grounded object, such as your computer chassis.

2. Remove the device from the package and inspect the device for loose components or any other sign of damage.

3. Never touch the exposed pins of connectors.

4. Unpack any other items and documentation from the kit.

NOTE: Do not install a device if it appears damaged in any way. Store the device in the antistatic package when the device is not in use.

7.4

All NI products are individually tested before shipment. The USRP? is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP? can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the USRP with proper anti-static methods.
- Never allow the USRP to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the USRP.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than **+10 dBm** of power into RF ports RF0 and RF1.

Never apply more than **+15 dBm** of power into the REF IN input.

Never apply more than **-15 dBm** of power into the GPS ANT input.

Always use at least 30dB attenuation if operating in loopback configuration

7.5

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on the SoM. A step-by-step guide for doing this is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on the Linux Application Note. **Release 3.14.1.0** or later of the USRP Hardware Driver (UHD), is needed.

7.6

This USRP-2974 host supports multiple, high-speed, low-latency interface options to the FPGA. To setup the device, follow these basic instructions:

- Configure the host ethernet adapter (enp1s0f0) to use an IP address of 192.168.40.1 and a subnet mask of 255.255.255.0
- Configure the host ethernet adapter (enp1s0f1) to use an IP address of 192.168.30.1 and a subnet mask of 255.255.255.0 (loopback with SFP+ cable needed)
- To test communications, ping the USRP FPGA at address "192.168.40.2" or ?192.168.30.2?

For more details on network setup, including PCIe connectivity, please see the section Interfaces and Connectivity of the NI USRP-2974 Hardware Resources page.

7.7

Once the software tools are installed on the onboard computer, verify the correct operation of the USRP by running the utility programs on the onboard computer. More information is available at the Verifying the Operation of the USRP Using UHD and GNU Radio Application Note.

7.8

7.8.1

- When rebooting the USRP-2974 open BIOS with DEL and go to Boot tab
- Enable PXE Network Boot with Legacy option, restart the system
- Go into the BIOS? Boot tab and set IBA CL Slot 00FE v0105 as first boot option, restart the system

7.8.2

- When rebooting the USRP-2974 open BIOS with DEL and go to Boot tab
- Enable PXE Network Boot with UEFI option, restart the system
- Go into the BIOS? Boot tab and set IPv4 as first boot option, restart the system

7.9

If you are connecting the USRP-2974 through the PCIe interface, then complete this section.

If you are connecting the USRP-2974 through the 1G or 10G Ethernet connection then do **NOT** complete this section.

1. Installer and commands taken from https://files.ettus.com/manual/page_ni_rio_kernel.html
2. Extract the installer and install as described (note the _ instead of ? in the folder name)
3. Enable or disable the PCIe link
    1. `$ sudo /usr/local/bin/niusrprio_pcie start`
    2. `$ sudo /usr/local/bin/niusrprio_pcie stop`
4. Check the status
    1. `$ sudo /usr/local/bin/niusrprio_pcie status`
5. see the connection over PCIe with
    1. `$ uhd_find_devices`

7.10

To find out if you have a **"Rev A"** hardware version of the USRP-2974 check the last character of the part number on the USRP-2974. E.g. "146873**A**"

To fully support Rev A with UHD software, install UHD as explained and run the following command:

```
$ /usr/local/lib/uhd/utils/usrp_burn_mb_eeprom --args="addr=192.168.40.2" --value="product=31131"
```

After a reboot run:

```
$ uhd_find_devices
```

Under the product category section you will see the following:

```
NI-2974
```

7.11

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

7.12

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

7.13

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?. Please be sure to include your order number and the serial number of your USRP.

7.14

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

7.15

- USRP-2974 Hardware Resource
- http://www.ni.com/pdf/manuals/377416c.pdf

8 X440 Getting Started Guide

8.1

8.1.1

- NI Ettus USRP X410 or X440
- DC Power Supply (12V, 20A)
- 1 Gigabit Ethernet Cat-5e Cable (3m)
- USB-A to USB-C Cable (1m)
- Getting Started Guide URL (QR Code)
- Safety, Environmental, and Regulatory Information

8.2

- https://kb.ettus.com/About_Sampling_Rates_and_Master_Clock_Rates_for_the_USRP_X440

8.3

- For Network Mode: A host computer with an available 1 or 10 Gigabit Ethernet interface for sample streaming. In addition to the Ethernet interface used for sampling streaming, your host computer will require a separate 1 Gigabit Ethernet interface for command and control streaming.
- For Stand-Alone Embedded Mode: A host computer with an available 1 Gigabit Ethernet port or a USB 2.0 port to remotely access the embedded Linux operating system running on ARM CPU.

8.4

All Ettus Research products are individually tested before shipment. The USRP is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP can cause the device to become non-functional. Take the following precautions to prevent damage to the unit.

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water or condensing moisture to come into contact with the device.
- Always use caution with FPGA, firmware, or software modifications.

X410: Never apply more than +14 dBm continuous <=3GHz, +17 dBm continuous >3GHz, or +20dBm more than 5 minutes >3GHz of power into any RF input.

X440: Never apply more than +13 dBm continuous <=2.5GHz, +17 dBm continuous between 2.5GHz and 3.6 GHz, or +20dBm continuous between 3.6 GHz and 4 GHz of power into any RF input.

X440: Always use at least 30dB attenuation if operating in loopback configuration

8.5

In order to use your Universal Software Radio Peripheral (USRP?), you must have the software tools correctly installed and configured on your host computer. The easiest way to install USRP Hardware Driver (UHD) is by getting a binary installer package for your operating system as described in the UHD manual about Binary Installation. If no binary packages are available for your operating system or you want to modify the sources by yourself, a step-by-step guide is available at the Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows Application Notes.

To find the latest release of UHD, see the UHD repository at https://github.com/EttusResearch/uhd.

The USRP X410 requires UHD version 4.1 or later. The USRP X440 requires UHD version 4.5 or later.

When you receive a brand-new device, it is strongly recommended that you download the latest filesystem image from the Ettus Research website update the unit. It is not recommended that you use the filesystem from the factory as-is. Instructions on downloading the latest filesystem image and updating it is listed below.

Note that if you are operating the device in Network Mode, the version of UHD running on the host computer and the USRP X4x0 must match.

8.6

Inside the kit you will find the X4x0 and an X4x0 power supply. Plug these in, connect the 1GbE RJ45 interface to your network, and power on the device by pressing the power button.

8.7

The STM32 microcontroller (also referred to as the "SCU") controls various low-level features of the X4x0 series motherboard: It controls the power sequencing, reads out fan speeds and some of the temperature sensors. It is connected to the RFSoC via an I2C bus. It is running software based on Chromium EC.

It is possible to log into the STM32 using the serial interface (see Connecting to the Microcontroller). This will allow certain low-level controls, such as remote power cycling should the CPU have become unresponsive for whatever reason.

8.7.1

The writable SCU image file is stored on the filesystem under /lib/firmware/ni/ec-titanium-revX.RW.bin (where X is a revision compatibility number). To update, simply replace the .bin file with the updated version and reboot.

8.8

The main non-volatile storage of the USRP is a 16 GB eMMC storage. This storage can be made accessible as a USB Mass Storage device through the USB-OTG connector on the back panel.

The entire root file system (Linux kernel, libraries) and any user data are stored on the eMMC. It is partitioned into four partitions:

Boot partition (contains the bootloader). This partition usually does not require modification. A data partition, mounted in /data. This is the only partition that is not erased during file system updates. Two identical system partitions (root file systems). These contain the operating system and the home directory (anything mounted under / that is not the data or boot partition). The reason there are two of these is to enable remote updates: An update running on one partition can update the other one without any effect to the currently running system. Note that the system partitions are erased during updates and are thus unsuitable for permanently storing information. Note: It is possible to access the currently inactive root file system by mounting it. After logging into the device using serial console or SSH (see the following two sections), run the following commands:

```
$ mkdir temp

$ mount /dev/mmcblk0p3 temp # This assumes mmcblk0p3 is currently not mounted

$ ls temp # You are now accessing the idle partition:

bin   data  etc   lib         media  proc  sbin  tmp    usr
boot  dev   home  lost+found  mnt    run   sys   uboot  var
```

The device node in the mount command might differ, depending on which partition is currently already mounted.

8.9

While Mender should be used for routine filesystem updates (see Updating Filesystems), it is also possible to access the X4x0's internal eMMC from an external host over USB. This allows accessing or modifying the filesystem, as well as the ability to flash the device with an entirely new filesystem.

In order to do so, you'll need an external computer with two USB ports, and two USB cables to connect the computer to your X4x0. The instructions below assume a Linux host.

First, connect to the APU serial console at a baud rate of 115200. Boot the device, and stop the boot sequence by typing noautoboot at the prompt. Then, run the following command in the U-boot command prompt:

```
ums 0 mmc 0
```

This will start the USB mass storage gadget to expose the eMMC as a USB mass storage device. You should see a spinning indicator on the console, which indicates the gadget is active.

Next, connect your external computer to the X4x0's USB to PS port using an OTG cable. Your computer should recognize the X4x0 as a mass storage device, and you should see an entry in your kernel logs (dmesg) that looks like this:

```
usb 3-1: New USB device found, idVendor=3923, idProduct=7a7d, bcdDevice= 2.23
usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 3-1: Product: USB download gadget
usb 3-1: Manufacturer: National Instruments
sd 6:0:0:0: [sdc] 30932992 512-byte logical blocks: (15.8 GB/14.8 GiB)
sdc: sdc1 sdc2 sdc3 sdc4
sd 6:0:0:0: [sdc] Attached SCSI removable disk
```

The exact output will depend on your machine, but from this log you can see that the X4x0 was recognized and /dev/sdc is the block device representing the eMMC, with 4 partitions detected (see eMMC Storage for details on the partition layout).

It is now possible to treat the X4x0's eMMC as you would any other USB drive: the individual partitions can be mounted and accessed, or the entire block device can be read/written.

Once you're finished accessing the device over USB, the u-boot gadget may be stopped by hitting Ctrl-C at the APU serial console.

8.10

Once the X4x0's eMMC is accessible over USB, it's possible to write the filesystem image and thus change the device's filesystem. You can obtain the latest filesystem image by running:

```
uhd_images_downloader -t sdimg -t x4xx
```

The output of this command will indicate where the downloaded images were put, or specify a custom location using using the `-i INSTALL_LOCATION` argument.

There are 2 ways to write the image to the X4x0's eMMC: using `dd` and `bmaptool`. Run one of the following commands, replacing `/dev/sdX` with the block device of the X4x0's eMMC (found in the device's kernel log or by running `lsblk`). Take care to use the correct block device or else you might overwrite the wrong drive!

```
sudo dd if=/path/to/usrp_x4xx_fs.sdimg of=/dev/sdX bs=1M
```

```
sudo bmaptool copy --bmap /path/to/usrp_x4xx_fs.sdimg.bmap /path/to/usrp_x4xx_fs.sdimg /dev/sdX
```

The former is generally preferred as it will always work, even if it slower than the latter.

8.11

Like any other USRP, all X4x0 USRPs are controlled by the UHD software. To integrate a USRP X4x0 into your C++ application, you would generate a UHD device in the same way you would for any other USRP:

```
auto usrp = uhd::usrp::multi_usrp::make("type=x4xx");
```

For a list of which arguments can be passed into make(), see Section Device Arguments.

8.12

Mender is a third-party software that enables remote updating of the root file system without physically accessing the device (see also the Mender website). Mender can be executed locally on the device, or a Mender server can be set up which can be used to remotely update an arbitrary number of USRP devices. Mender servers can be self-hosted, or hosted by Mender (see mender.io for pricing and availability).

When updating the file system using Mender, the tool will overwrite the root file system partition that is not currently mounted (note: the onboard flash storage contains two separate root file system partitions, only one is ever used at a single time). Any data stored on that partition will be permanently lost, including the currently loaded FPGA image. After updating that partition, it will reboot into the newly updated partition. Only if the update is confirmed by the user, the update will be made permanent. This means that if an update fails, the device will be always able to reboot into the partition from which the update was originally launched (which presumably is in a working state). Another update can be launched now to correct the previous, failed update, until it works.

To obtain the file system Mender image (these are files with a `.mender` suffix), run the following command on the host computer with Internet access:

```
$ sudo uhd_images_downloader -t mender -t x4xx --yes
```

NOTE: In the output of the command, the folder destination where the images are saved is printed out.

Next, you will need to copy this Mender file system image to the USRP X4xx. This can be done with the Linux utility `scp`.

```
$ scp /usr/local/share/uhd/images/usrp_x4xx_fs.mender root@192.168.1.51:~/.
```

Note: The path and IP may different for your configuration, the command above assumes you're using the default installation path of `/usr/local` and that the X4xx's IP is `192.168.1.51`.

After copying the Mender file system image to the X4xx, connect to the X4xx using either the Serial Console, or via SSH to gain shell access.

On the X4xx, run `mender install /path/to/latest.mender` to update the file system:

```
$ mender install /home/root/usrp_x4xx_fs.mender
```

The artifact can also be stored on a remote server:

```
$ mender install http://server.name/path/to/latest.mender
```

This procedure will take a few minutes to complete. After mender has logged a successful update, reboot the device:

```
$ reboot
```

If the reboot worked, and the device seems functional, commit the changes so that the boot loader knows to permanently boot into this partition:

```
$ mender -commit
```

To identify the currently installed Mender artifact from the command line, the following file can be queried on the X4x0:

```
$ cat /etc/mender/artifact_info
```

If you are using a Mender server, the updates can be initiated from a web dashboard. From there, you can start the updates without having to log into the device, and you can update groups of USRPs with a few clicks in a web GUI. The dashboard can also be used to inspect the state of USRPs. This is a simple way to update groups of rack-mounted USRPs with custom file systems.

If you are running a hosted server, the updates can be initiated from a web dashboard. From there, you can start the updates without having to log into the device, and can update groups of USRPs with a few clicks in a web GUI. The dashboard can also be used to inspect the state of USRPs. This is a simple way to update groups of rack-mounted USRPs with custom file systems.

8.13

The Ettus USRP X4x0 has various network interfaces:

eth0: RJ45 port.

The RJ45 port comes up with a default configuration of DHCP, that will request a network address from your DHCP server (if available on your network). This interface is agnostic of FPGA image flavor.

int0: internal interface for network communication between the embedded ARM processor and FPGA.

The internal network interface is configured with a static address: 169.254.0.1/24. This interface is agnostic of FPGA image flavor.

sfpX [, sfpX_1, sfpX_2, sfpX_3]: QSFP28 network interface(s), up-to four (one per lane) based on implemented protocol.

Each QSFP28 port has four high-speed transceiver lanes. Therefore, depending on the FPGA image flavor, up-to four different network interfaces may exist per QSFP28 port, using the sfpXfor the first lane, and sfpX_1-3 for the other three lanes. Each network interface has a default static IP address. Note that for multi-lane protocols, such as 100 GbE, a single interface is used (sfpX). The configuration files for these network interfaces are stored in: `/data/network/`

Interface Name	Description	Default Configuration	Configuration File	Example: X4_200/X4_400 FPGA image
eth0	RJ45	DHCP	eth0.network	DHCP
int0	Internal	169.254.0.1/24	int0.network	169.254.0.1/24
sfp0	QSFP28 0 (4-lanes interface or lane 0)	192.168.10.2/24	sfp0.network	192.168.10.2/24
sfp0_1	QSFP28 0 (lane 1)	192.168.11.2/24	sfp0_1.network	192.168.11.2/24
sfp0_2	QSFP28 0 (lane 2)	192.168.12.2/24	sfp0_2.network	192.168.12.2/24
sfp0_3	QSFP28 0 (lane 3)	192.168.13.2/24	sfp0_3.network	192.168.13.2/24
sfp1	QSFP28 1 (4-lanes interface or lane 0)	192.168.20.2/24	sfp1.network	N/C
sfp1_1	QSFP28 1 (lane 1)	192.168.21.2/24	sfp1_1.network	N/C
sfp1_2	QSFP28 1 (lane 2)	192.168.22.2/24	sfp1_2.network	N/C
sfp1_3	QSFP28 1 (lane 3)	192.168.23.2/24	sfp1_3.network	N/C

8.14

Once the X4x0 has booted, determine the IP address and verify network connectivity by running uhd_find_devices on the host computer:

X410:

```
$ uhd_find_devices

-- UHD Device 0

Device Address:
serial: 1234ABC
addr: 10.2.161.10
claimed: False
mgmt_addr: 10.2.161.10
product: x410
type: x4xx
```

X440:

```
$ uhd_find_devices

-- UHD Device 0

Device Address:
serial: 1234ABC
addr: 10.2.161.10
claimed: False
mgmt_addr: 10.2.161.10
product: x440
type: x4xx
```

By default, an X4x0 will use DHCP to attempt to find an address.

At this point, you should run:

`uhd_usrp_probe --args addr=<IP address>` to ensure functionality of the device.

Note: If you receive the following error:

`Error: RuntimeError: Graph edge list is empty for rx channel 0` then you will need to download a UHD-compatible FPGA as described in Updating the FPGA or using the following command (it assumes that FPGA images have been downloaded previously using uhd_images_downloader, or that the command is run on the device itself):

X410: `uhd_image_loader --args type=x4xx,addr=<ip address>,fpga=X4_200`

X440: `uhd_image_loader --args type=x4xx,addr=<ip address>,fpga=X4_400`

When running on the device, use `127.0.0.1` as the IP address.

You can now use existing UHD examples or applications (such as rx_sample_to_file, rx_ascii_art_dft, or tx_waveforms) or other UHD-compatible applications to start receiving and transmitting with the device.

See Network Interfaces for further details on the various network interfaces available on the X4x0.

8.14.1

The Ettus USRP X4x0 is equipped with status LEDs for its network-capable ports: RJ45 and QSFP28s, see RJ45 LED Behavior and QSFP28 LED Behavior accordingly.

8.14.1.1

The RJ45 port has two independent LEDs: green (right) and yellow (left). The table below summarizes the LEDs' behavior. Note that link speed indication is not currently supported.

Link / Activity	Green LED	Yellow LED
No Link	Off	Off
Link / No Activity	On	Off
Link / Activity	On	Blinking

8.14.1.2

Each QSFP28 connector has four LEDs, one for each high-speed transceiver lane. The table below summarizes the LEDs' behavior, note that for multi-lane protocols, such as 100 GbE, the corresponding LEDs are ganged together. Within the same image, multiple speeds on the same port (e.g., both 10 GbE and 100 GbE) are not supported, therefore link speed indication is not supported.

Link / Activity	QSFP28 LED (4 Total)
No Link	Off
Link / No Activity	Green (solid)
Link / Activity	Amber (blinking)

8.15

The X4x0 ships without a root password set. It is possible to ssh into the device by simply connecting as root, and thus gaining access to all subsystems. To set a password, run the command

`$ passwd` on the device.

8.16

It is possible to gain access to the device using a serial terminal emulator. To do so, the USB debug port needs to be connected to a separate computer to gain access. Most Linux, OSX, or other Unix flavors have a tool called 'screen' which can be used for this purpose, by running the following command:

`$ sudo screen /dev/ttyUSB2 115200` In this command, we prepend 'sudo' to elevate user privileges (by default, accessing serial ports is not available to regular users), we specify the device node (in this case, /dev/ttyUSB2), and the baud rate (115200).

The exact device node depends on your operating system's driver and other USB devices that might be already connected. Modern Linux systems offer alternatives to simply trying device nodes; instead, the OS might have a directory of symlinks under /dev/serial/by-id:

```
$ ls /dev/serial/by-id
usb-Digilent_Digilent_USB_Device_2516351DDCC0-if02-port0
usb-Digilent_Digilent_USB_Device_2516351DDCC0-if03-port0
```

Note: Exact names depend on the host operating system version and may differ.

The first (with the if02 suffix) connects to the STM32 microcontroller (SCU), whereas the second (with the if03 suffix) connects to Linux running on the RFSoC APU.

`$ sudo screen /dev/serial/by-id/usb-Digilent_Digilent_USB_Device_2516351DDCC0-if03-port0 115200` After entering the username root (no password is set by default), you should be presented with a shell prompt similar to the following:

`root@ni-x4xx-1234ABC:~#` On this prompt, you can enter any Linux command available. Using the default configuration, the serial console will also show all kernel log messages (unlike when using SSH, for example), and give access to the boot loader (U-boot prompt). This can be used to debug kernel or bootloader issues more efficiently than when logged in via SSH.

8.17

The microcontroller (which controls the power sequencing, among other things) also has a serial console available. To connect to the microcontroller, use the other UART device. In the example above:

```
$ sudo screen /dev/serial/by-id/usb-Digilent_Digilent_USB_Device_2516351DDCC0-if02-port0 115200
```

It provides a very simple prompt. The command 'help' will list all available commands. A direct connection to the microcontroller can be used to hard-reset the device without physically accessing it and other low-level diagnostics. For example, running the command reboot will emulate a reset button press, resetting the state of the device, while the command powerbtn will emulate a power button press, turning the device back on again.

8.18

The USRP X4x0 has two network connections: The dual QSFP28 ports, and an RJ45 connector. The latter is by default configured by DHCP; by plugging it into into 1 Gigabit switch on a DHCP-capable network, it will get assigned an IP address and thus be accessible via ssh.

In case your network setup does not include a DHCP server, refer to the section Serial Connection. A serial login can be used to assign an IP address manually.

After the device obtained an IP address you can log in from a Linux or OSX machine by typing:

`$ ssh root@ni-x4xx-1234ABC # Replace with your actual device name!` Depending on your network setup, using a .local domain may work:

`$ ssh root@ni-x4xx-1234ABC.local` Of course, you can also connect to the IP address directly if you know it (or set it manually using the serial console).

Note: The device's hostname is derived from its serial number by default (`ni-x4xx-$SERIAL`). You can change the hostname by creating the file `/data/network/hostname`, saving the desired hostname in it, then rebooting.

On Microsoft Windows, the connection can be established using a tool such as PuTTY, by selecting a username of root without password.

Like with the serial console, you should be presented with a prompt like the following:

```
root@ni-x4xx-1234ABC:~#
```

8.19

The USRP X4x0 can be configured to power on and boot automatically when power is applied. This setting can be controlled using the `eeprom-set-autoboot` script. This script is executed directly on the USRP X4x0. To enable autoboot, run `eeprom-set-autoboot on`; to disable autoboot, run `eeprom-set-autoboot off`.

8.20

The FPGA can be updated simply using uhd_image_loader:

`uhd_image_loader --args type=x4xx,addr=<IP address of device> --fpga-path <path to .bit>` or

`uhd_image_loader --args type=x4xx,addr=<IP address of device>,fpga=FPGA_TYPE` A UHD install will likely have pre-built images in /usr/share/uhd/images/. Up-to-date images can be downloaded using the uhd_images_downloader script:

`uhd_images_downloader` will download images into /usr/share/uhd/images/ (the path may differ, depending on how UHD was installed).

Also note that the USRP already ships with compatible FPGA images on the device - these images can be loaded by SSH'ing into the device and running:

X410: `uhd_image_loader --args type=x4xx,mgmt_addr=127.0.0.1,fpga=X4_200`

X440: `uhd_image_loader --args type=x4xx,mgmt_addr=127.0.0.1,fpga=X4_400`

8.21

Unlike the USRP X310 or other third-generation USRP devices, the FPGA image flavors do not only encode how the QSFP28 connectors are configured, but also which master clock rates are available. This is because the data converter configuration is part of the FPGA image (the ADCs/DACs on the X4x0 are on the same die as the FPGA). The image flavors consist of two short strings, separated by an underscore, e.g. X4_200 (X410) or X4_400 (X440) is an image flavor which contains 4x 10 GbE, and can handle an analog bandwidth of 200 MHz or 400 MHz respectively. The first two characters describe the configuration of the QSFP28 ports: 'X' stands for 10 GbE, 'C' stands for 100 GbE. For details see FPGA Image Flavor in the USRP Hardware Driver and USRP Manual.

The analog bandwidth determines the available master clock rates.

X410: As of UHD 4.1, only the X4_200 image is shipped with UHD, which allows a 245.76 MHz or 250 MHz master clock rate. With UHD 4.2, the CG_400 image was added allowing for 491.52 MHz and 500 MHz master clock rates. With UHD 4.5, the UC_200 image (245.76 MHz and 250 MHz master clock rate) was added.

X440: As of UHD 4.5, UHD ships with X4_400, X4_1600, CG_400 and CG_1600 images. The X4_400 and CG_400 images allow master clock rates between 125 MHz and 512 MHz and the usage of all 8 channels while the X4_1600 and CG_1600 images allow master clock rates between 125 MHz and 2048 MHz but only the usage of channels 0 and 4.

Any other images are considered experimental (unsupported).

8.22

Key	Description	Example Value
addr	IPv4 address of primary SFP+ port to connect to.	addr=192.168.30.2
second_addr	IPv4 address of secondary SFP+ port to connect to.	second_addr=192.168.40.2
mgmt_addr	IPv4 address or hostname to which to connect the RPC client. Defaults to `addr'.	mgmt_addr=ni-sulfur-311FE00
find_all	When using broadcast, find all devices, even if unreachable via CHDR.	find_all=1
master_clock_rate	Master Clock Rate in Hz.	master_clock_rate=250e6
converter_rate	Converter Rate in Hz. Only X440 and together with master_clock_rate.	master_clock_rate=250e6,converter_rate=1000e6
serialize_init	Force serial initialization of daughterboards.	serialize_init=1
skip_init	Skip the initialization process for the device.	skip_init=1
time_source	Specify the time (PPS) source.	time_source=internal
clock_source	Specify the reference clock source.	clock_source=internal
ref_clk_freq	Specify the external reference clock frequency, default is 10 MHz.	ref_clk_freq=20e6
discovery_port	Override default value for MPM discovery port.	discovery_port=49700
rpc_port	Override default value for MPM RPC port.	rpc_port=49701

This is only a subset of the existing device arguments. For a complete list please consult the UHD user manual of the X4x0 device series.

8.23

The USRP X4x0 includes a Jackson Labs LTE-Lite GPS module. Its antenna port is on the rear panel (see Front and Back Panels). When the X4x0 has access to GPS satellite signals, it can use this module to read out the current GPS time and location as well as to discipline an onboard OCXO.

To use the GPS as a clock and time reference, simply use gpsdo as a clock or time source. Alternatively, set gpsdo as a synchronization source:

```
// Set clock/time individually:
usrp->set_clock_source("gpsdo");
usrp->set_time_source("gpsdo");
// This is equivalent to the previous commands, but faster, as it sets
// both settings simultaneously and avoids duplicating settings that are shared
// between these calls.
```

```
usrp->set_sync_source("clock_source=gpsdo,time_source=gpsdo");
```

Note the GPS module is not always enabled. Its power-on status can be queried using the gps_enabled GPS sensor (see also The Sensor API). When disabled, none of the sensors will return useful (if any) values.

When selecting gpsdo as a clock source, the GPS will always be enabled. Note that acquiring a GPS lock can take some time after enabling the GPS, so if a UHD application is enabling the GPS dynamically, it might take some time before a GPS lock is reported.

8.24

The USRP X4x0 has two HDMI front-panel connectors, which are connected to the FPGA. For a description of the GPIO control API, see the USRP X4x0 GPIO UHD Manual Entry, the USRP X4x0 Series Manual, the ZBX ATR section (X410) and the FBX ATR section (X440).

8.25

The RF ports on the front panel of the X410 + ZBX correspond to the following subdev specifications:

Label	Subdev Spec
DB 0 / RF 0	A:0
DB 0 / RF 1	A:1
DB 1 / RF 0	B:0
DB 1 / RF 1	B:1

The RF ports on the front panel of the X440 + FBX correspond to the following subdev specifications (for xx_400 FPGA images):

Label	Subdev Spec
DB 0 / RF 0	A:0
DB 0 / RF 1	A:1
DB 0 / RF 2	A:2
DB 0 / RF 3	A:3
DB 1 / RF 0	B:0
DB 1 / RF 1	B:1
DB 1 / RF 2	B:2
DB 1 / RF 3	B:3

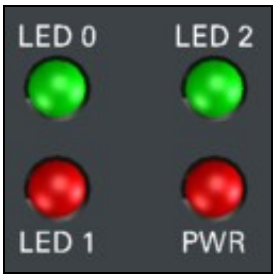
When using a xx_1600 FPGA image on X440, only A:0 and B:0 are available.

The subdev spec slot identifiers "A" and "B" are not reflected on the front panel. They were set to match valid subdev specifications of previous USRPs, maintaining backward compatibility.

These values can be used for uhd::usrp::multi_usrp::set_rx_subdev_spec() and uhd::usrp::multi_usrp::set_tx_subdev_spec() as with other USRPs.

8.26

The USRP X4x0 is equipped with four LEDs located on the device's rear panel. Each LED supports four different states: Off, Green, Red, and Amber. One LED (PWR) indicates the device's power state (see Power LED below). The other three LEDs (LED 0, LED 1, and LED 2) are user-configurable, different behaviors are supported for each of these LEDs (see User-configurable LEDs below).



8.26.1

Power LED The USRP X4x0's PWR LED is reserved to visually indicate the user the device's power state. Power LED Behavior describes what each LED state represents.

8.26.2

PWR LED State	Meaning
Off	No power is applied
Amber	Power is good but X4x0 is powered off
Green	Power is good and X4x0 is powered on
Red	Power error state

8.26.3

The USRP X4x0's user-configurable rear panel status LEDs (LED 0, LED 1, and LED 2) allow the user to have visual indication of various device conditions. Supported LED Behaviors provides a complete list of the supported behaviors for each user-configurable LED. By default, these LEDs are configured as described in LEDs Default Behavior.

The user may alter the default LEDs behavior either temporarily or persistently, see the Temporarily change the LED Behavior or Persistently in the UHD manual to change the LED Behavior accordingly.

https://files.ettus.com/manual/page_usrp_x4xx.html

8.27

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

8.28

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

- NOTE: This USRP product is a piece of test equipment.

8.29

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

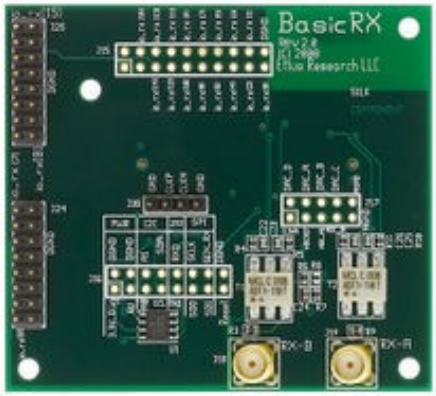
8.30

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

9 BasicRX Getting Started Guides

9.1

- BasicRX or BasicTX Daughterboard
- Screws



9.2

9.2.1

- N or X Series

9.2.2

- N or X Series

9.3

For complete step by step instructions to installing a daughterboard, please see the USRP N Series Quick Start (Daughterboard Installation) or USRP X Series Quick Start (Daughterboard Installation) guides.

9.4

All Ettus Research products are individually tested before shipment. The BasicTX/BasicRX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the BasicTX/BasicRX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than +10 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

9.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

9.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

9.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

9.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

10 CBX Getting Started Guides

10.1

- CBX Daughterboard
- Screws

10.2

10.2.1

- N or X Series

10.2.2

- X Series only

10.3

For complete step by step instructions to installing a daughterboard, please see the USRP N Series Quick Start (Daughterboard Installation) or USRP X Series Quick Start (Daughterboard Installation) guides.

10.4

All Ettus Research products are individually tested before shipment. The CBX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the CBX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

10.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

10.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

10.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

10.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

11 LFRX Getting Started Guides

11.1

- LFTX or LFRX Daughterboard
- Screws

11.2

11.2.1

- N or X Series

11.2.2

- N or X Series

11.3

For complete step by step instructions to installing a daughterboard, please see the USRP N Series Quick Start (Daughterboard Installation) or USRP X Series Quick Start (Daughterboard Installation) guides.

11.4

All Ettus Research products are individually tested before shipment. The LFTX/LFRX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the LFTX/LFRX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

⚠ Never apply more than +10 dBm of power into any RF input.

⚠ Always use at least 30dB attenuation if operating in loopback configuration

11.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

11.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

11.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

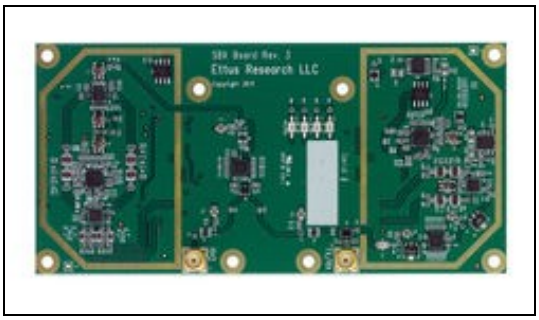
11.8

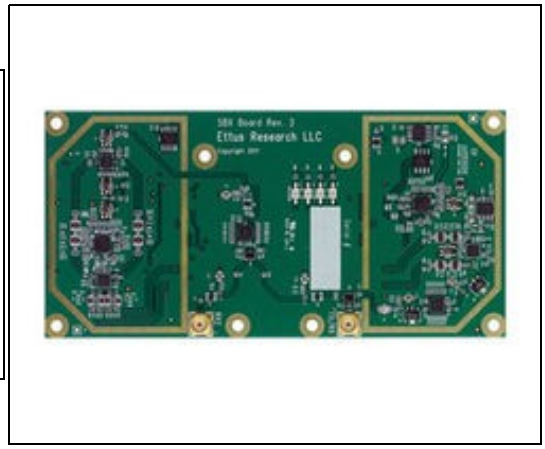
Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

12 SBX Getting Started Guides

12.1

- SBX Daughterboard
- Screws





12.2

12.2.1

- N or X Series

12.2.2

- X Series only

12.3

For complete step by step instructions to installing a daughterboard, please see the USRP N Series Quick Start (Daughterboard Installation) or USRP X Series Quick Start (Daughterboard Installation) guides.

12.4

All Ettus Research products are individually tested before shipment. The SBX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the SBX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

12.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

12.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

12.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

12.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

13 TwinRX Getting Started Guides

13.1

- TwinRX Daughterboard
- Two Front panel cables (1.4? SMA BH female to MMCX RA male)
- Two Front panel washer and bolts
- Two LO sharing cables (5? MMCX RA male to MMCX RA male)
- Screw kit
- Hardware installation guide
- Getting started guide

13.2

- USRP X300/X310

13.3

For complete step by step instructions to installing a daughterboard, please see the TwinRX Module Installation Guide, or the USRP X Series Quick Start (Daughterboard Installation) guide.

13.4

Note: LO sharing cables are not required for a single TwinRX setup. LO sharing cables are only required with two TwinRX daughterboards in a single USRP X300/X310.

Connector	Description	Min	Nominal	Damage
J1	LO2 Export	0 dBm	0 dBm	NA (Output)
J2	LO2 Input	0 dBm	+20dBm	
J3	LO1 Export	-12 dBm	+5 dBm	NA (Output)
J4	LO1 Input	-10 dBm	-5 dBm	+10dBm

13.4.1

TwinRX (A Slot)	TwinRX (B Slot)
J1 LO2 Export	J2 LO2 Input
J2 LO2 Input	J1 LO2 Export
J3 LO1 Export	J4 LO1 Input
J4 LO1 Input	J3 LO1 Export

13.5

13.5.1

- USRP Hardware Driver? (UHD): 3.10.0.0
- GNU Radio: 3.7.10

In order to use your TwinRX, you must have the software tools correctly installed and configured on your host computer. A step-by-step guide for doing this is available at the "Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux, OS X and Windows" Application Notes. UHD Release 3.10.0.0 or later is required; it is recommended to use the latest stable version of UHD that is available. The minimum version supported by GNU Radio is 3.7.10.

13.6

The UHD driver includes several example programs, which may serve as test programs or the basis for your application program. These example programs are already included with the UHD driver installation, and the source code can be obtained from the UHD repository on GitHub at: https://github.com/EttusResearch/uhd/tree/master/host/examples Additional information on usage to verify the USRP operating is available at the Verifying the Operation of the USRP Using UHD and GNU Radio Application Note.

Once you have the UHD driver installed, you can quickly verify the operation of your TwinRX by running the `rx_ascii_art_dft` UHD example program. The `rx_ascii_art_dft` program is included with the UHD driver and is located in the `$install_prefix/lib/uhd/examples` folder.

The `rx_ascii_art_dft` utility is a simple console-based, real-time FFT display tool. It is not graphical in nature, so it can be easily run over an SSH connection within a terminal window, and does not need any graphical capability, such as X Windows, to be installed.

You can run a simple test of the TwinRX device by connecting an antenna to the RX1 port and observing the spectrum of a commercial FM radio station in real-time. Please follow the steps listed below.

1. Attach an antenna to the RX1 antenna port of the TwinRX.

2. At a terminal prompt, run:

```
/usr/local/lib/uhd/examples/rx_ascii_art_dft --freq 88.1e6 --rate 1e6 --gain 70 --ref-lvl -30 --subdev "A:0" --ant RX1
```

- This step assumes that the TwinRX is installed in daughterboard slot A.
- To test the RX2 channel, change the antenna flag to `RX2`, and change the subdev spec to `A:1`

3. Modify the command-line argument `--freq` above to specify a tuning frequency for a strong local FM radio station.

4. You should see a real-time FFT display of 1 MHz of spectrum, centered at the specified tuning frequency.

5. Type "`Q`" or `Ctrl-C` to stop the program and to return to the Linux command line.

6. You can adjust the size of your terminal window and then re-run the command to enlarge or shrink the FFT display.

7. You can run with the `--help` option to see a description of all available command-line options.

Once you have installed GNU Radio, you can run the graphical tool `uhd_fft` to provide a GUI based real time FFT tool. To run `uhd_fft`, run the command below in a terminal:

```
uhd_fft -s 1e6 -g 70 -f 88.1e6 --spec "A:0" -A RX1
```

- Modify the command-line argument `-f` above to specify a tuning frequency for a strong local FM radio station.
- This step assumes that the TwinRX is installed in daughterboard slot A.
- To test the RX2 channel, change the antenna flag to `RX2`, and change the subdev spec to `A:1`

13.7

All Ettus Research products are individually tested before shipment. The TwinRX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the TwinRX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than +10 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

13.8

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

13.9

Every country has laws governing the reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

13.10

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

13.11

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

14 UBX Getting Started Guides

14.1

- UBX Daughterboard
- Screws

14.2

14.2.1

- N or X Series

14.2.2

- X Series only

14.3

For complete step by step instructions to installing a daughterboard, please see the USRP N Series Quick Start (Daughterboard Installation) or USRP X Series Quick Start (Daughterboard Installation) guides.

14.4

All Ettus Research products are individually tested before shipment. The UBX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the UBX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

14.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

14.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

14.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

14.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

15 WBX Getting Started Guides

15.1

- WBX Daughterboard
- Two MCX-Bulkhead cables

15.2

15.2.1

- N or X Series

15.2.2

- X Series only

15.3

For complete step by step instructions to installing a daughterboard, please see the USRP N Series Quick Start (Daughterboard Installation) or USRP X Series Quick Start (Daughterboard Installation) guides.

15.4

All Ettus Research products are individually tested before shipment. The WBX is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the WBX can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

15.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

15.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

15.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

15.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

16 Getting Started with RFNoC in UHD 4.0

16.1

AN-400 by Sugandha Gupta, Brent Stapleton, Wade Fife, and Michael Dickens

16.2

This guide describes how to get started with FPGA and Software development for RF Network-on-Chip (RFNoC?). It gives a brief introduction to RFNoC and explains the steps needed to generate, build, and use custom RFNoC images and introduces the process for creating and integrating new RFNoC IP blocks.

16.3

This guide is written for hardware and software engineers who want to use the RF Network-on-Chip (RFNoC?) architecture or want to develop intellectual property (IP) using the RFNoC architecture. For more details on the architecture please refer to the RFNoC Specification.

This guide assumes that you have some basic familiarity with USRPs, such as connecting them, configuring your network interfaces, etc., so that your USRP is ready for use. See the Getting Started Guide for your USRP if you are just getting started with USRPs.

16.4

The RFNoC code base is open source, including code that executes on the host, as well as code targeted to the USRP hardware (FPGA and microcontroller firmware). RFNoC is available under the open-source GNU Lesser General Public License (LGPL). For more information on our licensing policy, please contact info@ettus.com.

16.5

RFNoC is currently supported on the USRP X410 series, and all the Generation-3 USRPs in the X series (X3xx), E series (E3xx) and N series (N3xx). For details on the hardware, software, and process required to build custom USRP FPGA images that include RFNoC blocks, see the USRP Build Documentation in the UHD and USRP Manual.

It is recommended that you learn how to build an FPGA image for your USRP and download it to the device before starting this guide if you have never done so before. This will ensure you have all the necessary software installed.

16.6

16.6.1

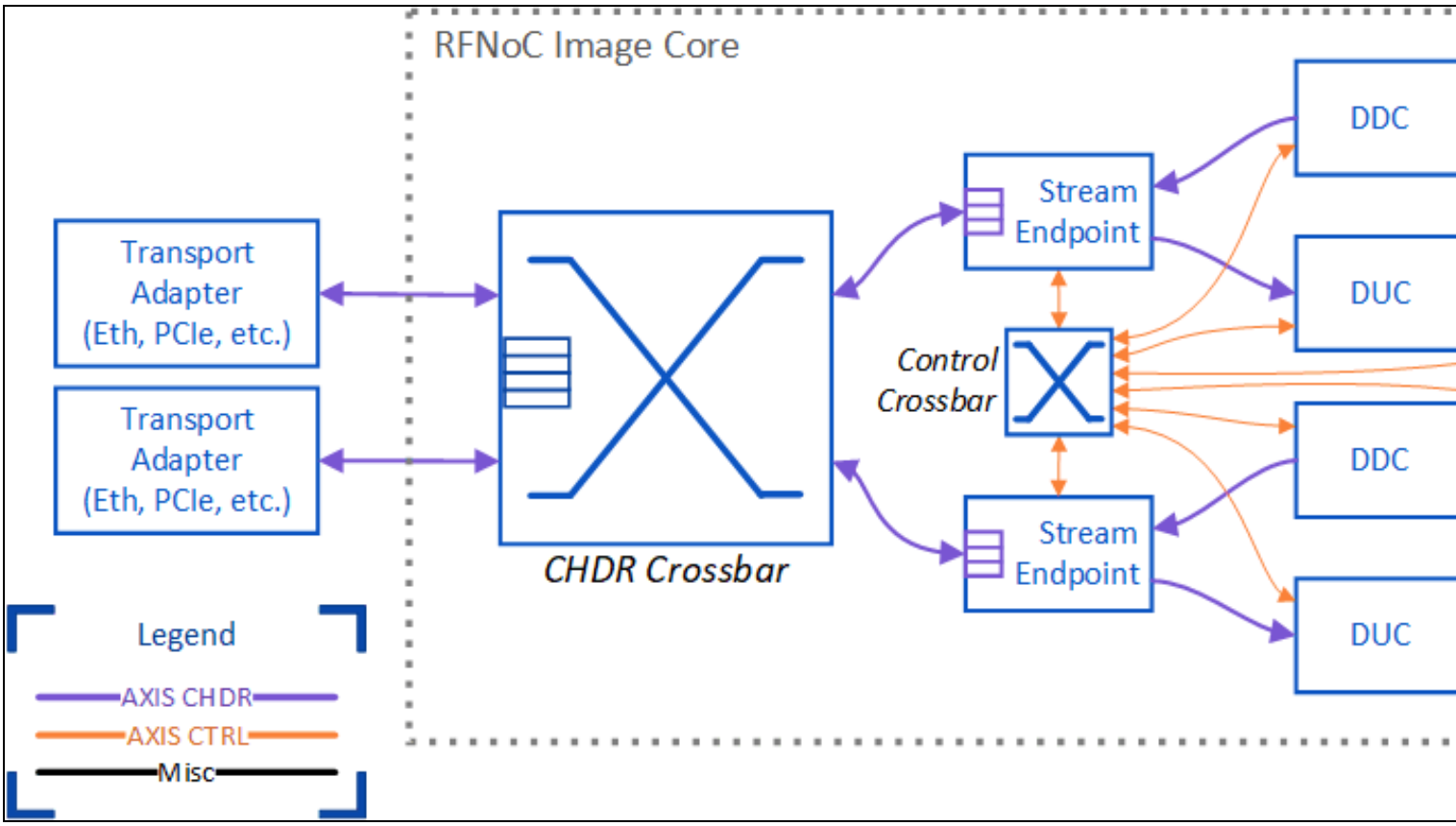
RFNoC? is a heterogeneous processing framework used to implement high-throughput DSP in the FPGA for Software Defined Radio (SDR) systems in an easy-to-use and flexible way. It provides all the infrastructure required to insert signal-processing IP into the FPGA logic and communicate with it through software. It also provides highly-optimized software and FPGA code to enable high-performance streaming to and from IP blocks on the USRP device.

The IP blocks in RFNoC are called *RFNoC blocks*. The RFNoC blocks wrap the IP and provide a custom interface to the RFNoC infrastructure through a tool-generated interface called the *NoC Shell*. Many standard blocks are included in UHD 4.0. These blocks enable typical operation of the USRP and allow RFNoC to connect to the different hardware components of the USRP. Several of the included blocks are described in the section Available RFNoC Blocks. You can mix the available blocks for your application, or develop your own custom RFNoC blocks with your own IP to open up new applications. The NoC Shell hides the complexity of RFNoC from your block, making it easy to plug your IP into the USRP.

RFNoC is used on all Generation-3 USRPs and is installed with UHD 4.0. USRPs ship with a default RFNoC image that can be used as is, or can be modified and/or customized to suite your application. So if you're using a modern USRP, it's running RFNoC under the hood even if you haven't customized anything.

16.6.2

The diagram below shows a simplified view of an RFNoC FPGA image that is similar to the default images available on USRPs.



Notice that an RFNoC FPGA is made up of several components. A description of each one is provided below.

- **RFNoC Image Core**
  This is the main block of the RFNoC framework and instantiates the components that make up RFNoC. The contents of the image core are described by an RFNoC image core YAML file, and the Verilog code that represents this block is automatically generated by the RFNoC Image Builder tool based on the YAML description. In other words, you provide a description of what you want to be included in RFNoC and the tools generate the code to implement that description.
- **Transport Adapter**
  The transport adapter is a component of the USRP that allows communication with an outside interface. These vary depending on the USRP model in use. For example, this could be a 10 Gbps Ethernet link for SFP+ port of the USRP, or a cabled PCIe interface.
- **CHDR Crossbar**
  The CHDR crossbar is a dynamic router for RFNoC traffic. This is a high-throughput crossbar designed for RF streaming applications. It is reconfigurable at run time via software and can be used to configure routes between stream endpoints and transport adapters. The number of ports available is configurable by changing the RFNoC image core YAML file. CHDR (Condensed Hierarchical Datagram for RFNoC) refers to the network protocol that is used for RFNoC.
- **Stream Endpoint (SEP)**
  The stream endpoint (SEP) provides the high-level flow control for traffic over the network. It also separates control traffic from data traffic to create a separate AXIS-Ctrl network for control traffic. Control traffic refers to things like register reads and writes for configuring and monitoring RFNoC blocks.
- **Control Crossbar**
  The control crossbar is similar to the CHDR Crossbar, but is only for control traffic. It has a much lower throughput and has been optimized for control-traffic, making it much less resource intensive than the CHDR crossbar.
- **Other Blocks**
  The DDC (digital down converter), DUC (digital up converter), and Radio in this example are RFNoC blocks that are included with UHD. These are standard components included the default USRP images and enable typical RF applications. Most blocks only communicate with the RFNoC network, but some blocks require access to external hardware interfaces, such as the radio or DRAM. In this example, the radio blocks connect to the radio hardware on the USRP.

16.6.3

The routes between blocks that go through the crossbar are *dynamic*. That is, the routes can be changed at run time and are controlled by software. The CHDR crossbar is very powerful in that it allows any of its ports to communicate with each other. It allows for streaming between blocks on the same FPGA, between RFNoC blocks and a host computer, or between blocks on different USRPs. New signal processing chains can be added to this crossbar, as needed, for the application.

Similarly, the control crossbar supports dynamic routing, allowing any RFNoC block to send control traffic to any other RFNoC block, even to blocks on a different USRP. Control traffic can also be sent from the host computer to the RFNoC blocks, for example to read/write registers. Additionally, RFNoC blocks can send control to the host computer. In this example, the Radio block sends control traffic to the computer to report specific events, such as overflow or underflow in the radio.

The block connections that don't go through a crossbar (e.g., the connection from the radio to the DDC, and the DDC to the stream endpoint, etc.) are *static* connections. That is, they cannot be changed at run time. Making a connection static has the advantage that it does not require additional ports on the crossbar and that the connection can be made much simpler because the high-level network flow-control protocol used by CHDR is not required. This reduces latency between blocks and leads to FPGA resource savings, allowing more logic be included in a single FPGA image. The static connections are described by the RFNoC image core YAML file. In order to change the static connections, the YAML description needs to be updated and the FPGA image needs to be rebuilt.

16.6.4

The RFNoC framework makes it easy to customize the RFNoC image and add your own IP blocks. Later on in this guide, we'll explore how you can customize the RFNoC image to add or remove blocks, as well as create your own RFNoC blocks and include them in your FPGA builds. This allows you to create highly-customized and high-performance FPGA images for your USRP.

16.7

To use RFNoC in UHD 4.0 you need to perform the following:

- Clone the UHD repository
- Install UHD
- Update your device's filesystem and FPGA image

Building custom RFNoC images requires the FPGA source code and build system. These are included in the UHD repository, located in `<repo>/fpga/usrp3/`, where `<repo>` refers to the location where you cloned the UHD repository.

Please see the Binary Installation for UHD installation documentation or Building and Installing instructions to build and install UHD from source. Additionally, AN-445 provides step-by step instructions for cloning the repository and installing UHD from source.

Starting with UHD 3.15, RFNoC is enabled by default, and starting with UHD 4.0, it can no longer be disabled, so no additional instructions are necessary to install support for RFNoC in UHD.

Many of the UHD utilities we will use in this guide are based on Python. Please ensure that your `PYTHONPATH` is set correctly. If not set correctly, you will see errors indicating that Python could not find the `uhd` or `image_builder` libraries. The correct setting may depend on where you installed UHD and which Linux distribution you are using. If you installed UHD from source to the default location on Ubuntu, you may need to add `/usr/local/lib/python3/dist-packages` to your `PYTHONPATH` variable. For example:

```
$ export PYTHONPATH=/usr/local/lib/python3/dist-packages
```

Instructions for updating your device's filesystem (applicable to E3xx and N3xx) and FPGA image can be found in that device's Getting Started Guide. If your filesystem is already up to date, or your device does not use a filesystem (e.g., X3xx), then instructions on flashing an FPGA image to a device can be found in the following locations:

- X3xx series
- E3xx series
- N3xx series

NOTE: FPGA images are specific to the USRP device, NOT the USRP series. For example, a USRP X300 FPGA image will NOT work on a USRP X310 and vice versa. Loading an image that does not correspond to your USRP device will likely lead to an error message but can brick the device under some circumstances.

16.7.1

Make sure you are using the `Bash` shell. Many of the build scripts used by RFNoC are written for `Bash`. See Reconfigure Default Shell in AN-315 for detailed instructions.

16.8

Before continuing, please verify that you have downloaded and flashed the latest FPGA image for your version of UHD. All FPGA images for Generation-3 and above devices are RFNoC images, so there is no requirement to download a special image for RFNoC.

16.8.1

Before we start customizing our FPGA, let's get familiar with the default FPGA image, which is pre-built with a set of RFNoC blocks.

Run the following command, with your USRP connected to your PC, to see what is available on the device.

```
$ uhd_usrp_probe --args type=x300
```

Note that your `args` may be different, depending on the USRP device you're using. The example here works for X300/X310. Refer to the manual page on Identifying USRP Devices for more details. If an RFNoC image was successfully loaded onto the USRP, the output will show something like the following:

```
  ___________________________________________________
 /
|       Device: X-Series Device
|     _____________________________________________________
|    /
|   |       Mboard: X310
|   |   revision: 11
|   |   revision_compat: 7
|   |   product: 30818
|   |   mac-addr0: 00:80:2f:17:40:6d
|   |   mac-addr1: 00:80:2f:17:40:6e
|   |   gateway: 192.168.10.1
|   |   ip-addr0: 192.168.10.2
|   |   subnet0: 255.255.255.0
|   |   ip-addr1: 192.168.20.2
|   |   subnet1: 255.255.255.0
|   |   ip-addr2: 192.168.30.2
|   |   subnet2: 255.255.255.0
|   |   ip-addr3: 192.168.40.2
|   |   subnet3: 255.255.255.0
|   |   serial: 311EF81
|   |   FW Version: 6.0
|   |   FPGA Version: 38.0
|   |   FPGA git hash: be53058
|   |
|   |   Time sources:  internal, external, gpsdo
|   |   Clock sources: internal, external, gpsdo
|   |   Sensors: ref_locked
|     _____________________________________________________
```

```
|    /
|   |       RFNoC blocks on this device:
|   |
|   |   * 0/DDC#0
|   |   * 0/DDC#1
|   |   * 0/DUC#0
|   |   * 0/DUC#1
|   |   * 0/Radio#0
|   |   * 0/Radio#1
|   |   * 0/Replay#0
|    ____________________________________________________
|    /
|   |       Static connections on this device:
|   |
|   |   * 0/SEP#0:0==>0/DUC#0:0
|   |   * 0/DUC#0:0==>0/Radio#0:0
|   |   * 0/Radio#0:0==>0/DDC#0:0
|   |   * 0/DDC#0:0==>0/SEP#0:0
|   |   * 0/Radio#0:1==>0/DDC#0:1
|   |   * 0/DDC#0:1==>0/SEP#1:0
|   |   * 0/SEP#2:0==>0/DUC#1:0
|   |   * 0/DUC#1:0==>0/Radio#1:0
|   |   * 0/Radio#1:0==>0/DDC#1:0
|   |   * 0/DDC#1:0==>0/SEP#2:0
|   |   * 0/Radio#1:1==>0/DDC#1:1
|   |   * 0/DDC#1:1==>0/SEP#3:0
|   |   * 0/SEP#4:0==>0/Replay#0:0
|   |   * 0/Replay#0:0==>0/SEP#4:0
|   |   * 0/SEP#5:0==>0/Replay#0:1
|   |   * 0/Replay#0:1==>0/SEP#5:0
...
```

More than this will likely be shown. The specifics of the output depend on the UHD version and which device you're running on. However, we're interested in the following sections:

- **Device description**. At the top, there is a section with basic information about your device, such as serial number, revision, etc. You can use this information to verify that you are indeed communicating with the correct device, and that the device has been updated. The FPGA git hash identifies the commit from which the FPGA image was built.
- **RFNoC blocks on this device**. This section lists all the RFNoC blocks in the loaded FPGA image. The blocks are listed by their block IDs. For example, `0/Radio#0` means that we're on device zero (`0/`), and we're talking about the first radio (with index `#0`). Unless you probe multiple devices at once by specifying multiple addresses in the `--args` argument, the device number will always be zero, but the block index will change depending on the number of blocks of that type. For example, on X310, `Radio#0` corresponds to RF A and `Radio#1` corresponds to RF B.
- **Static connections on this device**. This section lists how the blocks are pre-connected in the FPGA image. For example, the line `0/Radio#0:0==>0/DDC#0:0` means that radio zero, port zero (`:0`), is connected statically to DDC zero, port zero. Static connections can only be changed at compile time when building the FPGA image.

The connection endpoints labeled `SEP` are not RFNoC blocks. They are Stream Endpoints (SEPs). SEPs can send data packets to one another, or to streamers in software, using the CHDR crossbar, which is a dynamic router on the FPGA.

16.8.2

Many RFNoC blocks come with UHD. The HDL source code for these blocks resides in `<repo>/fpga/usrp3/lib/rfnoc/blocks/`. several of the blocks are described below.

Block Name	HDL Name	Description
AddSub	`rfnoc_block_addsub`	Add/Subtract Verilog/VHDL/HLS Example
DmaFIFO	`rfnoc_block_axi_ram_fifo`	FIFO that uses an AXI4 memory-mapped interface for storage. For use with external DRAM or on-chip SRAM.
DDC	`rfnoc_block_ddc`	Digital Down Converter
DUC	`rfnoc_block_duc`	Digital Up Converter
FFT	`rfnoc_block_fft`	Fast Fourier Transform
FIR	`rfnoc_block_fir_filter`	Finite Impulse Response Filter
Fosphor	`rfnoc_block_fosphor`	FFT and waterfall display tool
KeepOneInN	`rfnoc_block_keep_one_in_n`	Keep one sample/packet in N
LogPwr	`rfnoc_block_logpwr`	Computes an estimate of log2(i^2+q^2)
MovingAverage	`rfnoc_block_moving_avg`	Outputs the running average of the N most recent inputs of data stream
NullSrcSink	`rfnoc_block_null_src_sink`	Data source generator, sink, and loopback for testing
Radio	`rfnoc_block_radio`	Radio Interface
Replay	`rfnoc_block_replay`	Record/Playback using AXI4 memory-mapped interface. For use with external DRAM or on-chip SRAM.
SigGen	`rfnoc_block_siggen`	Signal Generator. Supports sinusoidal, constant, and random outputs, with configurable gain.
SplitStream	`rfnoc_block_split_stream`	Splits a single data stream into two
Switchboard	`rfnoc_block_switchboard`	A configurable RFNoC datapath switch for testing
VectorIIR	`rfnoc_block_vector_iir`	Implements an IIR filter with variable length delay line
Window	`rfnoc_block_window`	Windowing module for use with FFT block

16.8.3

In the UHD installation directory, you'll find example applications (e.g., in `/usr/lib/uhd/examples/` or `/usr/local/lib/uhd/examples/`). We'll look at `rfnoc_rx_to_file` to familiarize ourselves with the RFNoC API.

If we look at the source code for this example (located at `<repo>/host/examples/rfnoc_rx_to_file.cpp`), we can see the components necessary in an RFNoC application. In the `UHD_SAFE_MAIN` function, we create a few crucial objects, such as an RFNoC graph, a radio block controller, a DDC block controller, and an RX streamer.

```
auto graph = uhd::rfnoc::rfnoc_graph::make(args);
// ...
auto radio_ctrl = graph->get_block<uhd::rfnoc::radio_control>(radio_ctrl_id);
```

```
// ...
uhd::rfnoc::ddc_block_control::sptr ddc_ctrl;
// ...
auto rx_stream = graph->create_rx_streamer(1, stream_args);
```

We also make connections in our graph and configure our blocks.

```
// Connect blocks and commit the graph
for (auto& edge : chain) {
    if (uhd::rfnoc::block_id_t(edge.dst_blockid).match(uhd::rfnoc::NODE_ID_SEP)) {
        graph->connect(edge.src_blockid, edge.src_port, rx_stream, 0);
    } else {
        graph->connect(
            edge.src_blockid, edge.src_port, edge.dst_blockid, edge.dst_port);
    }
}
...
rate = radio_ctrl->set_rate(rate);
...
radio_ctrl->set_rx_frequency(freq, radio_chan);
```

Once our graph and blocks are configured, we use the `recv_to_file` function to receive data from our device and write it to a file on our host computer. Running this example with the `--help` argument shows the available options. For example, to receive 3 seconds of data at a specific frequency and sample rate, we can run:

```
rfnoc_rx_to_file --args type=x300 --freq 2.4e9 --rate 10e6 --duration 3
```

Now you are ready to move to the next step and build your own blocks and FPGA images.

16.9

UHD provides tooling to help develop custom RFNoC images. In this guide we will demonstrate how to use the RFNoC Image Builder, which will allow you to change which blocks are included and the connections between blocks. Let's go over common decisions you'll have to make with this tool.

- **Blocks Included**
  The particular blocks to be included in the image is the most impactful decision you'll make in the image building process, both in terms of image functionality and FPGA resource utilization. Adding more blocks means more processing in the image, but it also means longer FPGA build times and less space in the FPGA.
- **Static Connections**
  Recall that blocks can be statically connected, as opposed to connecting every block to the CHDR crossbar (dynamically connected). The number of dynamic connections has a large impact on the size of the CHDR crossbar, so static connections are used to lower resource utilization. However, once a block is statically connected in a chain of blocks, data cannot be sent between arbitrary blocks; it must traverse and be processed by each block in the order defined by the static connections.
- **Hardware Connections**
  All blocks share some connections defined by the RFNoC Specification. These connections are made automatically and cannot be changed. Many blocks require additional connections, such as clocks for the internal DSP or external DRAM interfaces. These connections must also be specified.

Now that we've gone over some of the considerations, let's look at how to actually build a custom RFNoC image. This begins with the RFNoC image core YAML description.

16.9.1

The image YAML file defines RFNoC blocks in the FPGA image. It is designed to be both human-readable and machine-parseable, which allows us to make edits quickly and easily. Each device type has a default YAML image file, which is named `<DEVICE>_rfnoc_image_core.yml` (e.g., `x310_rfnoc_image_core.yml`). Take a look at the RFNoC image core YAML file for the X310 by opening `<repo>/fpga/usrp3/top/x300/x310_rfnoc_image_core.yml`. Notice that it has the following sections.

- **General Parameters**
  Here we define the device, the bit width of our CHDR connections, as well as some versioning and licensing.
- **Stream Endpoints**
  In the `stream_endpoints` section we list each stream endpoint (SEP) that we wish to instantiate. These will be directly connected to the CHDR crossbar and will allow us to make dynamic connections within the CHDR crossbar. Typically you will have one SEP that forms the start and end of a single DSP chain of RFNoC blocks.
  You can also add some per-stream-endpoint configuration here, such as the ingress buffer size, which affects streaming performance from your computer to that SEP. For example, if we know that one SEP will be receiving data transferred from your computer to the USRP then a data buffer is needed to accept those incoming packets, in which case we specify the buffer size by setting the `buff_size` option on that SEP. Alternatively, if we know that a particular SEP only sends data from the USRP to the computer, then we won't need the ingress data buffer and we can set `buff-size` to 0, thus saving FPGA resources.
  Each SEP can have an AXIS-Ctrl and an AXIS-CHDR port, as indicated by the `ctrl` and `data` options. At least one AXIS-Ctrl port is required to communicate with the RFNoC blocks, so `ctrl` typically enabled on just the first SEP. Every SEP will usually have AXIS-CHDR connections to one or more RFNoC blocks, so `data` is usually enabled on all SEPs.
- **NoC Blocks**
  In the `noc_blocks` section we specify all the RFNoC blocks to include in the FPGA image. We'll need to give each block a unique name and reference a block definition YAML file (which we'll discuss in a later section). The blocks chosen have the greatest impact on the functionality of the image, as well as providing very coarse control over the FPGA resource utilization. We'll look at how to change the blocks included in the image as part of our example below.
- **Static Connections**
  The `connections` section defines static connections between blocks, stream endpoints, and various hardware interfaces on the USRP. Statically connected chains of blocks will usually start and end at a stream endpoint (SEP). SEPs are automatically connected to the CHDR crossbar by the RFNoC infrastructure. Remember that data must be passed through the entire statically connected chain; dynamic connections cannot be made to the statically connected blocks in the middle of the chain. Other hardware connections, such as to the external DRAM, would also be specified here. We'll take a closer look at making connections as part of our example below.
- **Clock Domains**
  The `clk_domains` section defines which clocks to connect to each block's clock inputs. Some blocks do not need any clock connections beyond the base clocks required by RFNoC. These required connections are not listed here, since they are always the same for each block. Other blocks may require additional clocks. For example, the radio blocks should be connected to the `radio` clock. Many other blocks require a `ce` (Compute Engine) clock, which is used for the block's internal DSP.

16.9.2

Before we look at editing the image core YAML file, let's go over how to use the RFNoC image builder to generate a bitstream from that YAML file. Additional details on how to run the image builder can be found with the `--help` option:

```
$ rfnoc_image_builder --help
```

Here are some of the options provided:

- **`-y`**: Path to the RFNoC image core YAML file
- **`-t`**: The image target you would like to build. These are the same targets that are used in the FPGA `make` process. More details on these can be found in the Generation 3 USRP Build Documentation of the UHD and USRP Manual. If not specified, the default specified in the image core YAML file will be used.
- **`-l debug`**: Use the `debug` log level. This prints more information about the FPGA connections and available port names, which can be useful for debugging connection errors in the YAML file.
- **`--generate-only`**: Generate the HDL files required, but do not build the FPGA. Users can then build the FPGA later using `make <target>`.
- **`-I`**: Path to the directory containing out-of-tree block YAML descriptions (the YAML files installed with UHD are included by default). This option is only needed if using an out-of-tree RFNoC block.
- **`-F`**: Path to the FPGA source code (e.g., `<repo>/fpga`). This path is only required if the current working directory is not within the UHD repository.

For example, to build the default RFNoC image for X310, you might use the following command:

```
$ cd <repo>/fpga/usrp3/top/x300/
$ rfnoc_image_builder -y ./x310_rfnoc_image_core.yml -t X310_XG
```

In this example `<repo>` refers to the location where you cloned the UHD repository and should be replaced by the location you used. `x310_rfnoc_image_core.yml` is the default RFNoC image core YAML file for the X310, which is in the x300 project directory. `X310_XG` is the make target to use for the build. `XG` in this case refers to dual 10 Gbps Ethernet.

For an out-of-tree RFNoC block, you will also need to specify the location of the block information. For example:

```
$ rfnoc_image_builder -F <repo>/fpga -I <repo>/host/examples/rfnoc-example -y <repo>/host/examples/rfnoc-example/icores/x310_rfnoc_image_c
```

This example shows how to build an FPGA with the Gain example out-of-tree RFNoC block, which is located in `<repo>/host/examples/rfnoc-example/`. The `-F` option is added to specify the location of the FPGA source, and `-I` specifies the location of the out-of-tree block YAML.

The image builder performs several steps in order to build the FPGA image from the image coure YAML:

1. It generates the `<device>_rfnoc_image_core.v` file. This file includes the Verilog code described by the YAML image core file. A `<device>_static_router.hex` file is also generated. This file describes the static connections that should be made by the Verilog code. In the case of the X310 examples above, the output files would be named `x310_rfnoc_image_core.v` and `x310_static_router.hex`, and would be placed in the project directory for the X310 (`<repo>/fpga/usrp3/top/x300/`).
2. The image builder will configure the environment for building the FPGA. This is equivalent to sourcing the `setupenv.sh` script for the device type being built. For example, in our example above, it would run `source <repo>/fpga/usrp3/top/x300/setupenv.sh`.
3. The image builder runs `make <target>` to build the IP and the FPGA bitstream for the indicated target. The generated `rfnoc_image_core.v` and `static_router.hex` are automatically pulled into this build. The completed bitstream will be located in `<repo>/fpga/usrp3/top/{project}/build` directory, which is the same directory created through the normal make process. For example, for X310 it would be located in `<repo>/fpga/usrp3/top/X300/build`.

16.9.3

As an example, let's run through how to modify the YAML file to modify the RFNoC image. Suppose we have an application that we'd like to run on a USRP X310 and we would like to offload the FFT processing to the FPGA. UHD provides an FFT RFNoC block, and the default X310 image has some extra space in it, so we should be able to add this block. First, we copy the default X310 image core file, named `x310_rfnoc_image_core.yml`.

```
$ cd <repo>/fpga/usrp3/top/x300/
$ cp x310_rfnoc_image_core.yml x310_with_fft.yml
```

Now open `x310_with_fft.yml` in your favorite text editor. We'll start by making some room for our new block. The default images use very large ingress FIFO buffers for the main SEPs to maximize streaming performance. But we want to make sure we have enough memory buffer space for our new blocks. So Start by reducing the `buf_size` parameters for `ep0` and `ep2` from 65536 to 32768. If using a device other than X310, the numbers may be different, but you can similarly reduce the `buff_size` parameter by half. On smaller devices, it may be necessary to remove the Replay block to make room for the FFT blocks. After making the changes on X310, the result should look like the following:

```
stream_endpoints:
  ep0:                      # Stream endpoint name
    ctrl: True                     # Endpoint passes control traffic
    data: True                     # Endpoint passes data traffic
    buff_size: 32768               # Ingress buffer size for data
  ...
  ep2:                      # Stream endpoint name
    ctrl: False                    # Endpoint passes control traffic
    data: True                     # Endpoint passes data traffic
    buff_size: 32768               # Ingress buffer size for data
```

Now we can add our FFT block. We'll put it on its own stream endpoint, so we first need to add a new stream endpoint. Add the following to the `stream_endpoints` section:

```
stream_endpoints:
  ...
  ep_fft:                # The name can be incremented from previous SEP
    ctrl: False          # Only the first SEP needs control traffic
    data: True           # We do want to pass data through this SEP
    buff_size: 32768     # Ingress buffer size for data
```

In this example, we've named the SEP `ep_fft`. Other names could be given, as long as the name is unique (it does not have to be numbered). Now that we've allocated an SEP for our block, we need to instantiate the actual block. In the next section, add the following:

```
noc_blocks:
  ...
  fft0:                          # FFT block name
    block_desc: 'fft_1x64.yml'   # Block YAML descriptor file
    parameters:                  # Specify any Verilog module parameters (optional)
```

```
      EN_FFT_SHIFT: 1
```

Again, the name `fft0` is arbitrary, but the name must be unique. In some cases, there will also be block parameters you'll want to pass, such as the data format of the FFT output data. In our example, we're setting `EN_FFT_SHIFT` parameter to 1, which causes the FFT block to center the zero frequency bin.

Now that we've added our FFT block, we need to connect it to the stream endpoint. In the next section of the YAML file, we add the static connections:

```
connections:
  ...
  - { srcblk: ep_fft, srcport: out0,  dstblk: fft0,   dstport: in_0 }
  - { srcblk: fft0,   srcport: out_0, dstblk: ep_fft, dstport: in0  }
```

This connects the output of SEP `ep_fft` to the input of block `fft0`, and the output of `fft0` to the input of `ep_fft`. Since we're placing the FFT block on its own SEP, the only connections we need to make are between the SEP and the FFT. All SEPs are automatically connected to the CHDR crossbar, so this effectively connects the FFT block to the crossbar, allowing it to communicate with anything on the RFNoC network.

The names of block ports are defined in the YAML descriptions for the blocks. Blocks can use any names for their ports, and they don't have to be numbered (unless the number of ports is parameterized). Generally, the block ports are named `in_N` for inputs to the block and `out_N` for outputs. SEP ports are named `in0` for the input `out0` for the output.

If you are having trouble connecting a block due to an unresolved connection, running `rfnoc_image_builder` with the `-l debug` option will list all available block ports.

Finally, the FFT block has an additional clock input port named `ce` that is used for the FFT signal processing. We need to connect it to a clock domain. Any clock that's at least as fast as the incoming data rate should be sufficient. For example, X3xx devices have a `ce` clock (214.286 MHz) that is usually a good choice, but `rfnoc_chdr` clock (200 MHz) should also work for this example. For N31x and E3xx devices, `rfnoc_chdr` clock is a good choice (200 MHz on N31x and 100 MHz on E3xx). For N32x, `radio` clock (250 MHz) is a good choice. For example, this is how you would connect the X310's `ce` clock to the `ce` port of the FFT block:

```
clk_domains:
  ...
  - { srcblk: _device_, srcport: ce, dstblk: fft0, dstport: ce }
```

And this is how you would connect `rfnoc_chdr` clock to the `ce` port:

```
clk_domains:
  ...
  - { srcblk: _device_, srcport: rfnoc_chdr, dstblk: fft0, dstport: ce }
```

And finally, this is how you would connect `radio` clock:

```
clk_domains:
  ...
  - { srcblk: _device_, srcport: radio, dstblk: fft0, dstport: ce }
```

The source block name `_device_` is special and refers to the USRP device itself. Choose a clock that's apprpriate for your device and connect it as shown above.

And that's it! The next step will be to run the image builder on our modified YAML file. Once that's done, and the bitstream has been created, you can load it onto your USRP X310 device, and verify the blocks, as we did in the Inspect the Default Image section.

For example, from the X300 directory, you would run the following command to run the image builder:

```
$ rfnoc_image_builder -y x310_with_fft.yml -t X310_XG
```

To download the FPGA bitstream to the X310, run the following:

```
$ uhd_image_loader --args "type=x300,addr=192.168.30.2" --fpga-path ./build/usrp_x310_fpga_XG.bin
```

You may need to change the IP address to match your device, depending on your configuration. After completing the flash update and power-cycling the USRP, run `uhd_usrp_probe` to confirm that the FFT block was recognized.

```
$ uhd_usrp_probe --args "type=x300,addr=192.168.30.2"
```

Take a look at the RFNoC blocks and the static connections on the device. You should see the following new blocks and connections:

```
|     ____________________________________________________
|    /
|   |       RFNoC blocks on this device:
...
|   |   * 0/FFT#0
...
|     ____________________________________________________
|    /
|   |       Static connections on this device:
...
|   |   * 0/SEP#6:0==>0/FFT#0:0
|   |   * 0/FFT#0:0==>0/SEP#6:0
...
```

16.9.4

Now that we've added the FFT block and verified that it is operating as expected, let's see if we can modify it a little. Let's say that we're sure that we always want to receive the FFT bins from our device, and we don't want to see the raw samples. In order to save some FPGA resources, we can move our FFT block to be between the DDC and the SEP, on the RX data path. Instead of the previous modifications to our YAML file, we want the following:

```
stream_endpoints:
  ...
  # Unchanged from the default image core (no need for ep_fft).

noc_blocks:
  ...
  fft0:                          # FFT block name
    block_desc: 'fft_1x64.yml'   # Block YAML descriptor file
    parameters:                  # Specify any Verilog module parameters (optional)
      EN_FFT_SHIFT: 1
```

```
connections:
  ...
  # Change this line:
  # - { srcblk: ddc0, srcport: out_0, dstblk: ep0, dstport: in0  }
  # Change it to the following to add fft0 between ddc0 and ep0:
  - { srcblk: ddc0, srcport: out_0, dstblk: fft0, dstport: in_0 }
  - { srcblk: fft0, srcport: out_0, dstblk: ep0,  dstport: in0 }

clk_domains:
  ...
  # As before, we still connect our FFT block to the clock domain
  - { srcblk: _device_, srcport: rfnoc_chdr, dstblk: fft0, dstport: ce }
```

This last line is valid for E310/E320; for X300/X310/N300/N310/N320/N321 use the following:

```
  - { srcblk: _device_, srcport: ce, dstblk: fft0, dstport: ce }
```

And as simply as that, we have added the FFT block to our RX block chain. Remember that this is a static connection, so when the resulting FPGA image is loaded onto your device, you will no longer be able to receive samples from the DDC directly; all samples on that chain will be processed by the FFT block, and you will only receive FFT bins from this radio chain.

16.10

16.10.1

Now that we've gone over what is provided by UHD, let's start to look at custom RFNoC development. It's recommended that custom RFNoC development be kept separate from the in-tree UHD RFNoC infrastructure and examples, in order to simplify version control and licensing. Modules located outside of the repository are called *out-of-tree* (OOT).

In order to understand how OOT modules for RFNoC are created, let's take a look at the example that's provided with UHD. You can find the example located in `<repo>/host/examples/rfnoc-example/`. This example includes a simple Gain RFNoC block that we can use as a reference for creating our own OOT blocks.

16.10.2

Take a look at the `rfnoc-example` directory structure. An OOT module is a collection of block implementations, the software controllers for those blocks, and sometimes applications to demonstrate their functionality. These different components within the OOT module are organized into the directories described below.

- HDL and Image Resources
  - `fpga/`: Contains the HDL (e.g., Verilog) required for the module. Each RFNoC block should have its own subdirectory here, such as `rfnoc_block_gain`, for the Gain block.
  - `blocks/`: Contains the YAML RFNoC block definition files. These describe the block's interfaces and are used by the image builder. We'll go into more detail with regards to what goes into these YAML files in the Creating Your Own RFNoC Block section.
  - `icores/`: Contains example FPGA image core YAML files, to demonstrate how to create an FPGA image using your custom blocks. These image core files may demonstrate important parameters or show recommended configurations for chains of blocks.
- Block Controller and Example Software
  - `include/`: Contains the headers for the block controller software. Once installed, these headers will be used by UHD and other applications to interface with and control your custom RFNoC blocks.
  - `lib/`: Contains the block controller implementations. The files here should implement the features outlined in the headers located in the `include/` directory. All unit test code should also be located here.
  - `apps/`: Contains example applications for your custom blocks. These applications may demonstrate how to use the block controllers, common configurations, or how to process data from your blocks.
- Infrastructure
  - `cmake/`: Contains any custom CMake commands the module may need. In general, simple modules will not need to modify this directory at all. More complicated modules with additional external dependencies may need to modify this.

Take some time to explore the files in the `rfnoc-example`. Note the following files and directories related to the Gain example:

- `rfnoc-example/fpga/rfnoc_block_gain/`
  This is the HDL for the Gain RFNoC block.
  - `rfnoc_block_gain.v`
    The top-level synthesizable file for the Gain block
  - `noc_shell_gain.v`
    The NoC Shell for the Gain block
  - `rfnoc_block_gain_tb.sv`
    The simulation testbench
  - `Makefile`
    The simulation Makefile
- `rfnoc-example/blocks/gain.yml`
  The YAML block definition file for the Gain block
- `rfnoc-example/icores/x310_rfnoc_image_core.yml`
  An example RFNoC image core YAML description showing how to connect the Gain block
- `rfnoc-example/include/rfnoc/example/gain_block_control.hpp`
  The software block controller header
- `rfnoc-example/lib/gain_block_control.cpp`
  The software block controller implementation
- `rfnoc-example/apps/init_gain_block.cpp`
  An example showing how to find and initialize the Gain block

16.10.3

To create your own OOT module, make a copy of the `rfnoc-example` directory. This will contain all the subdirectories and infrastructure files that you need to create your custom module. The directory name for your copy should be renamed to your desired name; we recommend something like `rfnoc-foo`, for example, where `foo` is the name for your OOT module. You'll also need to change `rfnoc-example` within the module's `CMakeLists.txt` files to your module's name.

Let's start by creating our own copy of the `rfnoc-example` to work with. We'll call our module `demo`. You can put the copy wherever you like, but for this example we'll put it in our home directory.

```
$ cp -r <repo>/host/examples/rfnoc-example ~/
$ mv ~/rfnoc-example ~/rfnoc-demo
```

Now we need to edit our `CMakeLists.txt` files to rename our module. Edit the following files and change all instances of `rfnoc-example` to `rfnoc-demo`:

```
~/rfnoc-demo/CMakeLists.txt
~/rfnoc-demo/lib/CMakeLists.txt
```

16.10.4

The OOT module will need to be built and installed in order to use its RFNoC blocks. To build and install the OOT module we just created, do the following:

```
$ mkdir ~/rfnoc-demo/build
$ cd ~/rfnoc-demo/build
$ cmake -DUHD_FPGA_DIR=<repo>/fpga/ ../
```

This configures the project to be installed in the default location. You may want to provide a different install directory to CMake using the `-DCMAKE_INSTALL_PREFIX`. Please refer to the UHD Installation Guide or the installation guide for other CMake-based projects for instructions on configuring CMake.

Note that we specify the location of the FPGA source code using the `-DUHD_FPGA_DIR` option. This allows you to run the FPGA testbenches and to build the example FPGA image provided with the OOT example.

At this point, you can run `make help` to see what options are available. A few of the available make targets are described below:

- `make rfnoc-demo`: Build the block controllers for the RFNoC blocks
- `make testbenches`: Run the testbenches for the RFNoC blocks
- `make x310_rfnoc_image_core`: Build the FPGA example image
- `make install`: Install the module

To build and install the block, perform the following steps.

```
$ cd ~/rfnoc-demo/build
$ make
$ make install
```

Note that `sudo` may be required depending on the install location.

16.10.5

At this point you can build the example FPGA that's included in the `icores` directory. Take a look at `~/rfnoc-demo/icores/x310_rfnoc_image_core.yml`. Note that the `gain` block is added in exactly the same way that the FFT block was in Example: Adding an FFT Block. You can build this example FPGA for the X310 using the following steps:

```
$ cd ~/rfnoc-demo/build
$ make x310_rfnoc_image_core
```

Alternatively, you can call `rfnoc_image_builder` directly. In order to get the image builder to find your custom blocks, you may need to supply some additional arguments to find the OOT module, as shown in the following example:

```
$ rfnoc_image_builder -F <repo>/fpga ?I ~/rfnoc-demo/include/rfnoc -y ~/rfnoc-demo/icores/x310_rfnoc_image_core.yml
```

The `-I` option is only required if the OOT module has not been installed on the system.

If you have an X310, you can build and use the provided example to test the Gain block. If you have a different USRP, you can use its default `rfnoc_image_core.yml` as a starting point and follow the same steps that we followed for the FFT block in Example: Adding an FFT Block, then build it using the process described above.

16.10.6

After building an FPGA image for your USRP and downloading it to your device, the Gain block should be available. Test this by running `uhd_usrp_probe`:

```
$ uhd_usrp_probe --args "type=x300,addr=192.168.30.2"
```

Again, note that your `args` may be different or may not be required if you only have a single USRP connected.

Take a look at the RFNoC blocks and the static connections on the device. You should see the following new blocks and connections:

```
|     ____________________________________________________
|    /
|   |       RFNoC blocks on this device:
...
|   |   * 0/Block#0
...
|     ____________________________________________________
|    /
|   |       Static connections on this device:
...
|   |   * 0/SEP#4:0==>0/Block#0:0
|   |   * 0/Block#0:0==>0/SEP#4:0
...
```

16.11

The Gain block is a useful example and could even be used as the starting point for a new block, but rather than trying to copy and edit the Gain block, it is best to use the RFNoC ModTool to create a new block template. This is also required if you want to change the interfaces provided to your IP that are exposed by the NoC Shell.

16.11.1

The first step in creating a new block is to write a YAML description for the block that you want to create. The options available are described RFNoC Specification. You can also look at the blocks available in `<repo>/host/include/uhd/rfnoc/blocks/` for examples. We'll provide a brief overview here.

Open up and take a look at the Gain block definition located in `~/rfnoc-demo/blocks/gain.yml`. Notice the different sections, which are described below:

- **General Parameters**
  In this section we give our block a name (`module_name`) and a unique block ID (`noc_id`). The NoC ID is an arbitrary 32-bit number that will be used by the software to identify the block during system discovery. You can also specify the CHDR bus width, which should be 64 for Generation-3 USRPs.
- **Clocks**
  In the `clocks` section, we define the clocks that will be used by the block. The `rfnoc_chdr` and `rfnoc_ctrl` clocks are required. Many blocks also have a `ce` clock for internal DSP.
- **Control Interface**
  The `control` section defines what type of control interface to make available to your block. The control interface is used for register reads and writes. Most blocks do not need to modify this section and will use the `ctrlport` interface type. Control Port is the standard register interface used by RFNoC. The `clk_domain` parameter allows you to specify which clock domain to use for the register interface exposed to your block. Typically this is the same clock that is used for the `data` interface, to avoid clock crossings.
- **Data Interface**
  The `data` section defines what type of data interface to make available to your block and describes the ports. Most blocks use either the `axis_pyld_ctxt` or the `axis_data` interface types.
  The `clk_domain` parameter allows you to specify which clock domain to use for the data interfaces exposed to your block. Typically this is the same clock that is used for the `control` interface, to avoid clock crossings.
  Under the `inputs` and `outputs` sections you can describe the input and output ports.
- **IO Ports**
  The `io_ports` section is used to describe other device-specific ports to which your block needs to connect (such as the DRAM or radio interfaces). For most blocks, this section is not used.

A few other sections might be present in the YAML file (e.g., `registers` or `properties`). These sections are reserved for future use and can be left empty.

16.11.2

Let's start by making a copy of the Gain block YAML for our use case.

```
$ cp ~/rfnoc-demo/blocks/gain.yml ~/rfnoc-demo/blocks/demo.yml
```

Now open the `demo.yml` file you just created and make the following changes:

1. Change the name of the block from `gain` to `demo`
2. Change the `noc_id` to `0x0000DE30`
3. Change the `format` for the `in` and `out` ports from `int32` to `sc16`.

This YAML file now describes a block with the following features:

- The block named "demo"
- NoC ID of 0xDE30
- CtrlPort register interface on the `rfnoc_chdr` clock domain
- A single 32-bit input port named "In" on the `rfnoc_chdr` clock domain that will provide sc16 samples to your IP
- A single 32-bit output port named "Out" on the `rfnoc_chdr` clock domain that will receive sc16 samples from your IP

16.11.3

Now that we have a block definition YAML file, we can generate the source code templates and NoC Shell for our block. To do so, run the following command:

```
$ python3 <repo>/host/utils/rfnoc_blocktool/rfnoc_create_verilog.py -c ~/rfnoc-demo/blocks/demo.yml -d ~/rfnoc-demo/fpga/rfnoc_block_demo
```

This will create a folder named `~/rfnoc-demo/fpga/rfnoc_block_demo` with an RFNoC block template for you to use. Explore the folder that was created. You should see the following files:

- **`Makefile`**: This is the Makefile for the simulation testbench. See Running a Testbench in the UHD and USRP Manual for instructions on how to run a testbench.
- **`Makefile.srcs`**: This is another makefile that identifies the HDL source code that makes up your RFNoC block.
- **`noc_shell_demo.v`**: This is the NoC Shell that was generated for your block. It provides the interfaces described in the block definition YAML.
- **`rfnoc_block_demo.v`**: This is a template for your RFNoC block. It includes all the top-level ports required by RFNoC and instantiates the NoC Shell.
- **`rfnoc_block_demo_tb.sv`**: This is a testbench template for your RFNoC block. It instantiates your RFNoC block and the bus functional models (BFMs) needed to communicate with your block in simulation.

Take a moment to look at the `rfnoc_block_demo.v` and `rfnoc_block_demo_tb.sv` files that were generated for your block. Notice the "User Logic" section towards the end of the `rfnoc_block_demo` module. This is the location where you can insert your own IP.

The input data samples come in on the `m_in_payload` AXI4-Stream bus. The output data samples go out on the `s_out_payload` AXI4-Stream data bus.

If your input packets are the same size as your output packets (i.e., for every sample in you generate one sample out) then the `m_in_context` data bus can be used to drive `s_out_context`. The context bus is described in detail in the RFNoC Specification.

Similarly, the testbench has a section where you can add your own test sequences to the test bench.

If you decide you need to change the RFNoC interfaces for your block (e.g., add ports, or change the interface type), then it is recommended to update the block definition YAML then regenerate the block. Please take care to not overwrite any code you have written when regenerating the files for your block! Rerunning the tool will cause new files to be generated, with a new NoC Shell, and will demonstrate how to update your code for the new interfaces.

Refer to the Gain example and other RFNoC blocks for examples of how to complete your RFNoC block logic and testbenches.

16.12

In this guide we have given a brief introduction on how to develop for RFNoC in UHD 4.0. There's a lot more you can do than is described here, but this will hopefully get you started. Happy coding!

17 RFNoC 4 Migration Guide

18

The UHD 4.0 release includes a major upgrade to the RFNoC framework called RFNoC 4. This article is a guide to aid users in migrating their existing RFNoC blocks from RFNoC 3 to RFNoC 4. The RFNoC Block Development Environment section provides guidance on how to setup an environment for developing out-of-tree RFNoC blocks in RFNoC 4. The UHD, FPGA, GNU Radio Migration sections provide general information on topics that most users will encounter when migrating their blocks. Finally, an equivalent RFNoC 3 and RFNoC 4 implementation of a digital gain RFNoC Block has been provided as a reference.

19

19.1

```
sudo apt-get install autoconf automake build-essential ccache cmake cpufrequtils doxygen ethtool \
g++ git inetutils-tools libboost-all-dev libncurses5 libncurses5-dev libusb-1.0-0 libusb-1.0-0-dev \
libusb-dev python3-dev python3-mako python3-numpy python3-requests python3-scipy python3-setuptools \
python3-ruamel.yaml libtinfo5 libncurses5
```

19.1.1

Please reference to Xilinx (xilinx.com) for installation instructions.

Note: The dependencies step above included installing libtinfo5 libncurses5, which is a workaround for getting Vivado 2019.1 to run on Ubuntu 20.04

19.1.2

```
git clone --branch UHD-4.1 https://github.com/ettusresearch/uhd.git uhd
mkdir uhd/host/build; cd uhd/host/build
cmake ..
make
sudo make install
```

19.1.3

Note: If your design does not use GNU Radio, then installing GNU Radio and gr-ettus is not required

```
git clone --branch maint-3.8 --recursive https://github.com/gnuradio/gnuradio.git gnuradio
mkdir gnuradio/build; cd gnuradio/build;
cmake ..
make
sudo make install
```

Please refer to the GNU Radio Build Instructions for dependencies and a more detailed description.

19.1.4

```
git clone --branch maint-3.8-uhd4.0 https://github.com/ettusresearch/gr-ettus.git gr-ettus
mkdir gr-ettus/build; cd gr-ettus/build;
cmake -DENABLE_QT=True ..
make
sudo make install
```

20

Two options exist for developing RFNoC blocks depending on whether the your RFNoC block integrates with GNU Radio in an out-of-tree module or if it only uses UHD?s C++ API in a standalone application. The sections below outline how to setup the development environment for each scenario.

20.1

The tool rfnocmodtool automates the process of creating GNU Radio out-of-tree (OOT) modules that also have support for RFNoC blocks. This tool is part of gr-ettus and it has been ported to RFNoC 4.

Due to changes in almost every source file, it is recommended to use rfnocmodtool to generate a new RFNoC block from scratch and then update the generated ?skeleton? files.

20.1.1

The following steps show how to create an OOT module called *example* and RFNoC block called *gain* using rfnocmodtool. The naming is only for example purposes.

```
rfnocmodtool newmod
Name of the new module: example

cd rfnoc-tutorial
rfnocmodtool add
Enter name of block/code (without module name prefix): gain
Enter valid argument list, including default arguments: (leave blank)
Add Python QA code? [y/N] N
Add C++ QA code? [y/N] N
Block NoC ID (Hexadecimal): (Enter Noc ID of your block)
Skip Block Controllers Generation? [UHD block ctrl files] [y/N] N
Skip Block interface files Generation? [GRC block ctrl files] [y/N] N
```

Note: Noc IDs have been reduced from 64-bits in RFNoC 3 to 32-bits in RFNoC 4

The following are the relevant files that need to be updated when migrating your RFNoC Block.

```
rfnoc-example/
    grc/
        example_gain.block.yml           ? RFNoC Block GNU Radio Companion YAML file
    examples/
        gain.grc                         ? Example flowgraph using gain RFNoC Block
    include/tutorial/
        gain.h                           ? GNU Radio block C++ header
        gain_block_ctrl.hpp              ? RFNoC Block Controller C++ header
    lib/
        gain_impl.cc                     ? GNU Radio block C++ source
        gain_impl.h                      ? GNU Radio block C++ header
        gain_block_ctrl_impl.cpp         ? RFNoC Block Controller C++ source
    rfnoc/blocks/
        gain.yml                         ? RFNoC Block Description YAML file
    rfnoc/fpga/rfnoc_block_gain
        noc_shell_gain.v                 ? RFNoC Block Noc Shell Verilog Source
        rfnoc_block_gain.v               ? RFNoC Block Verilog Source
        rfnoc_block_gain_tb.v            ? RFNoC Block Testbench
    rfnoc/icores
        gain_x310_rfnoc_image_core.yml   ? Image Core YAML file with gain block
```

20.1.1.1

```
cd rfnoc-tutorial
mkdir build; cd build
cmake -DUHD_FPGA_DIR=(path to uhd/fpga directory) ..
make
sudo make install
```

20.1.1.2

CMake automatically creates makefile targets to run the generated testbench code for each added RFNoC block. For example, here is how to run the gain block testbench:

```
cd rfnoc-tutorial/build
make rfnoc_block_gain_tb
```

20.1.1.3

CMake automatically creates makefile targets to build FPGA images using the generated image core yaml files found in rfnoc/icore. Every RFNoC block created by rfnocmodtool automatically has an image core yaml file generated in that directory. For example, here is how to build an FPGA image using the image core yaml file generated for the gain block:

```
cd rfnoc-tutorial/build
make gain_x310_rfnoc_image_core
```

20.1.2

For applications that only use the UHD API, an example out-of-tree (UHD source tree) RFNoC block exists called rfnoc-example. It is located in the UHD source at uhd/host/examples/rfnoc-example. This directory can be copied outside of the UHD source tree and used a starting point to migrate your RFNoC block.

The following are the relevant files that need to be updated when migrating your RFNoC Block.

```
rfnoc-example/
    apps/
        init_gain_block.cpp         ? Example C++ application testing gain block
    blocks/
        gain.yml                    ? RFNoC Block Description YAML file
    fpga/rfnoc_block_gain
        noc_shell_gain.v            ? RFNoC Block Noc Shell Verilog Source
        rfnoc_block_gain.v          ? RFNoC Block Verilog Source
```

```
        rfnoc_block_gain_tb.v       ? RFNoC Block Testbench
    icores/
        x310_rfnoc_image_core.yml   ? Example Image Core YAML file
    include/rfnoc/example
        gain_block_control.hpp      ? RFNoC Block Controller C++ header
    lib/
        gain_block_control.cpp      ? RFNoC Block Controller C++ source
```

20.1.2.1

```
cd rfnoc-example
mkdir build; cd build
cmake ..
make
sudo make install
```

20.1.2.2

CMake automatically creates makefile targets to run RFNoC Block testbench simulations. For every RFNoC block subdirectory listed in the CMakeLists.txt file in the rfnoc-example/fpga directory, a target with the RFNoC block name appended with ?_tb? is added as a makefile target. For example, here is how to run the gain RFNoC block testbench:

```
cd rfnoc-example/build
make rfnoc_block_gain_tb
```

20.1.2.3

CMake automatically creates makefile targets to build a FPGA image for each image core yaml file listed in the CMakeLists.txt file in the rfnoc-example/icore directory. Each image core yaml file must be listed in the CMakeLists.txt. For example, here is how to build an FPGA image using the image core yaml file generated for the gain block:

```
cd rfnoc-tutorial/build
make gain_x310_rfnoc_image_core
```

21

This ZIP archive, File:migration example.zip, contains equivalent RFNoC 3 and RFNoC 4 versions of a digital gain RFNoC Block. The following sections will refer to files in this archive to show how the file structure changes when migrating from RFNoC 3 to RFNoC 4.

22

Migration reference files for this section from the Gain RFNoC Block example:

Description	RFNoC 3 Files	RFNoC 4 Files
Block Description	rfnoc/blocks/gain.xml	lib/gain_block_ctrl_impl.cpp include/example/gain_block_ctrl.hpp
Block Controller	rfnoc/blocks/gain.yml	lib/gain_block_ctrl_impl.cpp include/example/gain_block_ctrl.hpp

Note: Files are relative to the rfnoc-example directory in the respective rfnoc3 and rfnoc4 directories

22.1

RFNoC 3 used Noc Script XML, a domain specific language, to describe the configuration of a RFNoC block: the Noc ID, register names and addresses, args for writing to the registers, and the input/output ports.

RFNoC 4 replaces the Noc Script XML file with an easier to read and edit Block Description YAML file format. From a high level, the Block Description YAML file serves a similar function as the Noc Script XML file, with some similarities and key differences outlined in table below:

Item	Noc Script XML	Block Descript YAML	RFNoC 4 Notes
Block Name	`<name>gain</name>`	`module_name: gain`	
Noc ID	`<id>B160000000000000</id>`	`noc_id: 0xB16`	Noc ID are limited to 32-bits

Registers	`<registers>`
	`  <setreg>`
	`  <name>GAIN</name>`
	`  <address>128</address>`
	`  </setreg>`
	`</registers>`
		N/A	Registers must be defined in the Block Controller

Arguments	`<args>`
	`  <arg>`
	`  <name>gain</name>`
	`  <type>int</type>`
	`  ...`
	`  </arg>`
	`</args>`
		N/A	Args are implemented with properties in the Block Controller

Data Ports	`<ports>`
	`  <sink>`
	`  <name>in</name>`
	`  </sink>`
	`  <name>out</name>`
	`</ports>`
		`data:`
		`  fpga_iface: axis_pyld_ctxt`
		`  clk_domain: rfnoc_chdr`
		`  inputs:`
		`  in:`
		`  ...`
		`  outputs:`
		`  out:`
		`  ...`	

Control Ports	N/A	`control:`
		`  sw_iface: nocscript`
		`  fpga_iface: ctrlport`
		`  interface_direction: slave`
		`  ...`	

Clocking	N/A	`clocks:`
		`  - name: rfnoc_chdr`
		`  freq: "[]"`
		`  - name: rfnoc_ctrl`
		`  freq: "[]"`	

Note: For a more detailed description of the RFNoC 4 Block Description YAML syntax and the various options, see the RFNoC Specification.

22.1.1

Much of the user facing RFNoC software API has not changed or remains very similar between RFNoC 3 and RFNoC 4. The table below outlines some of the notable differences:

RFNoC 3	RFNoC 4	RFNoC 4 Notes
`usrp = uhd::device3::make(...)`	`graph = uhd::rfnoc::rfnoc_graph::make()`	No longer need to create a device3 object
`usrp->get_block_ctrl(...)`	`graph->get_block(...)`	Rename
N/A	`graph->enumerate_static_connections()`	Used to check static connections, for example the DDC and DUC blocks are usually statically connected to the radio block
`usrp->get_tx_streamer(...)`	`graph->create_tx_streamer(...)`	Rename
`usrp->get_rx_streamer(...)`	`graph->create_rx_streamer(...)`	Rename
N/A	`graph->commit()`	Commit graph and run initial checks
`sr_write(...)`	`regs().poke32(...)`	Address increments by 4
`sr_read32(...)`	`regs().peek32(...)`	Address increments by 4
`sr_read64(...)`	`regs().poke64(...)`	Address increments by 8
`set_arg(...)`	`set_property(...)`	Block args replaced with block properties concept
`get_arg(...)`	`get_property(...)`	Block args replaced with block properties concept

22.1.2

In RFNoC 3, RFNoC blocks can have arguments (also known as args) that are used to write user registers. This is implemented in the Noc Script XML in the <args> section.

RFNoC 4 expands and generalizes this concept with block properties: a high-level representation of the state of the block. Zero or more properties can be defined by the user in their RFNoC Block?s Block Controller C++ class. When read or written to, they can trigger a callback to a user defined resolver function. The RFNoC Specification provides more details on properties in the ?Block Properties? section.

The following shows an example of how to migrate a RFNoC 3 Noc Script XML ?arg? based register write to a RFNoC 4 property based implementation in the Block Controller:

22.1.2.1

```
<registers>
  <setreg>
    <name>GAIN</name>
    <address>128</address>
  </setreg>
</registers>

<args>
  <arg>
    <name>gain</name>
    <type>int</type>
    <value>1</value>
    <check>GE($gain, 0) AND LE($gain, 32767)</check>
    <check_message>Gain must be in the range [0, 32767]</check_message>
    <action>SR_WRITE("GAIN", $gain)</action>
  </arg>
</args>
```

22.1.2.2

```
// <registers>
//    <setreg>
//      <name>GAIN</name>
//      <address>128</address>
//    </setreg>
// </registers>
// Note: In RFNoC 4, register addresses can start at address 0 instead of address 128 as in RFNoC 3.
const uint32_t gain_block_ctrl::REG_GAIN_ADDR    = 0;
const uint32_t gain_block_ctrl::REG_GAIN_DEFAULT = 1;

class gain_block_ctrl_impl : public gain_block_ctrl
{
public:
    RFNOC_BLOCK_CONSTRUCTOR(gain_block_ctrl)
    {
        _register_props();
    }
private:
    void _register_props()
    {
        register_property(&_user_reg, [this]() {
            int user_reg = this->_user_reg.get();
            // <check>GE($gain, 0) AND LE($gain, 32767)</check>
            // <check_message>Gain must be in the range [0, 32767]</check_message>
            if (user_reg < 0 || user_reg > 32767) {
                throw uhd::value_error("Size value must be in [0,32767]");
            }
            // <action>SR_WRITE("GAIN", $gain)</action>
            this->regs().poke32(REG_USER_ADDR, user_reg);
        });
    }

// <name>gain</name>
// <type>int</type>
// <value>1</value>
property_t<int> _user_reg{"gain", REG_USER_DEFAULT, {res_source_info::USER}};
}
```

As the above shows, writing to a register can be replicated with a property and a resolver function. Of course, the resolver function can also be made much more sophisticated. For additional examples, see the in-tree block controllers in uhd/host/lib/rfnoc.

23

Migration reference files for this section from the Gain RFNoC Block example:

Description	RFNoC 3 Files	RFNoC 4 Files
Block Verilog Code	rfnoc/fpga-src/noc_block_gain.v	rfnoc/fpga/rfnoc_block_gain/rfnoc_block_gain.v
Block Noc Shell	N/A	rfnoc/fpga/rfnoc_block_gain/noc_shell_gain.v
Block Testbnech	rfnoc/testbench/noc_block_gain/noc_block_gain_tb.sv	rfnoc/fpga/rfnoc_block_gain/rfnoc_block_gain_tb.sv
Image Core	N/A	rfnoc/icores/gain_x310_rfnoc_image_core.yml

Note: Files are relative to the rfnoc-example directory in the respective rfnoc3 and rfnoc4 directories

23.1

RFNoC 4 replaces the highly parameterized RFNoC 3 Noc Shell with a per-block customized Noc Shell generated from the block?s Block Description YAML file. The Noc Shell generated via rfnocmodtool or the existing one in rfnoc-example is acceptable for most blocks that require one input and output data port.

23.1.1

Some blocks may need multiple data ports or other modifications. This requires editing the Block Description YAML file and then using the Python script rfnoc_create_verilog.py (found in uhd/host/utils/rfnoc_blocktool) to generate a new Noc Shell instance.

The argument ?-c? is used to provide the YAML file location. ?-d? provides the output destination directory.

Note: It is suggested to not set the destination directory to your existing RFNoC block code, as the script will automatically overwrite the existing code!

Example usage:

```
rfnoc_create_verilog.py -c ./rfnoc-example/rfnoc/blocks/gain.yml -d ./output/
```

23.1.1.1

In the generated Noc Shell Verilog code, a block?s Noc ID can be changed by updating the NOC_ID parameter on the *backend_iface* module. Make sure this matches the Noc ID in both the Block Description YAML file and Block Controller C++ code.

23.1.1.2

The RFNoC 3 version of Noc Shell outputs / accepts CHDR data packets consisting of a header, optional timestamp, and payload on a 64-bit AXI stream bus. Most designs then used a module called AXI Wrapper to handle the conversion between CHDR data packets and sample streams on a 32-bit AXI stream bus. AXI Wrapper also supported SIMPLE_MODE which for some use cases could transparently handle the header portion of the CHDR data packet. Otherwise, the user would need to set the header via m_axis_data_tuser.

In RFNoC 4, Noc Shell has absorbed AXI Wrapper?s functionality. Noc Shell outputs two AXI stream buses per input / output port: a payload and context bus. The payload bus is in most cases identical to AXI Wrapper?s output: a 32-bit stream of samples on an AXI Stream bus with packets delimited by tlast. The context AXI stream bus carries the header, optional timestamp, and optional metadata. If your block used AXI Wrapper?s SIMPLE_MODE, then you can loop the context bus back into Noc Shell. If not, you will need to modify the context bus data. Refer to the RFNoC Specification for the format and timing diagram of the context bus.

*Important Note: If your block used the AXI Rate Change module, Noc Shell has another data port mode to support this use case called **axis_data** that can be set in the Block Descriptor YAML file (see the fpga_iface entry). This mode causes the Noc Shell data ports to look more like AXI Wrapper?s and therefore makes them compatible with AXI Rate Change. See the DDC, DUC, or Keep One in N RFNoC Blocks for an example.*

23.1.2

CtrlPort replaces the Settings Bus in RFNoC 4. The CtrlPort bus is similar to the Settings Bus with a few key differences. The table below compares the signaling between the two bus formats and provides notes on any differences. Timing diagrams and additional information on the CtrlPort bus are also available in the RFNoC Specification.

Settings Bus (RFNoC 3)	CtrlPort (RFNoC 4)	RFNoC 4 Notes
set_stb	ctrlport_reg_wr	Write strobe
set_addr	ctrlport_req_addr	20-bits instead of 8-bits, increments by 4 instead of by 1, no reserved addresses (versus addresses 0-127 for Settings Bus)
set_data	ctrlport_req_data	Write data
N/A	ctrlport_req_rd	Read strobe equivalent of ctrlport_req_wr
rb_addr	N/A	CtrlPort uses ctrlport_req_addr for both **read and write** addresses
rb_data	ctrlport_resp_data	Read data, 32-bits instead of 64-bits
rb_stb	N/A	CtrlPort requires ack strobe for **reads and writes**

One additional difference when using CtrlPort is that there is not an equivalent Settings Register module. The bus is simple enough to setup a clocked process to handle reading from and writing to registers. See the Verilog example below:

```
// Note: Register addresses increment by 4
localparam REG_USER_ADDR    = 0; // Address for example user register
localparam REG_USER_DEFAULT = 0; // Default value for user register

reg [31:0] reg_user = REG_USER_DEFAULT;

always @(posedge ctrlport_clk) begin
  if (ctrlport_rst) begin
    reg_user = REG_USER_DEFAULT;
  end else begin
    // Default assignment
    m_ctrlport_resp_ack <= 0;

    // Read user register
```

```
    if (m_ctrlport_req_rd) begin // Read request
      case (m_ctrlport_req_addr)
        REG_USER_ADDR: begin
          m_ctrlport_resp_ack  <= 1;
          m_ctrlport_resp_data <= reg_user;
        end
      endcase
    end

    // Write user register
    if (m_ctrlport_req_wr) begin // Write requst
      case (m_ctrlport_req_addr)
        REG_USER_ADDR: begin
          m_ctrlport_resp_ack <= 1;
          reg_user            <= m_ctrlport_req_data[31:0];
        end
      endcase
    end
  end
end
```

*Important Note: For blocks that make heavy use of the Settings Bus and/or Settings Registers, there is a CtrlPort to Settings Bus bridge available called **ctrlport_to_settings_bus**. See the Keep One In N RFNoC Block for example code on how to interface with it.*

23.1.3

While RFNoC 4 does overhaul the RFNoC 3 testbench infrastructure API, most of the high level concepts remain the same. The table below outlines some of the commonly used RFNoC 3 functions / code and the RFNoC 4 equivalent.

Operation	RFNoC 3	RFNoC 4

Setup RFNoC	`` `RFNOC_SIM_INIT(...) ``
	 `` `RFNOC_ADD_BLOCK(...) ``
	 `` `RFNOC_CONNECT(...) ``
		`RfnocBlockCtrlBfm #(...) blk_ctrl = new(...);`
		 `blk_ctrl.connect_master_data_port(...)`
		 `blk_ctrl.connect_slave_data_port(...)`
		 Note: Instantiate one Block Controller BFM per RFNoC Block

Setup Test Cases	`` `TEST_CASE_START(...) `` `` `TEST_CASE_DONE(...) ``	`test.start_test(...)` `test.end_test()`
Register Read	`tb_streamer.read_reg(...)`	`blk_ctrl.reg_read(...)`
Register Write	`tb_streamer.write_reg(...)`	`blk_ctrl.reg_write(...)`
Send Data / Samples	`tb_streamer.send(...)`	`blk_ctrl.send_items(...)`
Receive Data / Samples	`tb_streamer.recv(...)`	`blk_ctrl.recv_items(...)`

23.1.4

RFNoC 4 replaces uhd_image_builder, the RFNoC 3 FPGA image building tool, with a new tool called rfnoc_image_builder. This tool produces a FPGA bitstream based on an Image Core YAML file that describes the device configuration (e.g. X310 with dual 10GigE) and included RFNoC blocks along with their connections (both static and dynamic), clocking, and I/O.

Both rfnocmodtool and the UHD in-tree example called rfnoc-example automatically setup make targets to handle running rfnoc_image_builder. If you want to use rfnoc_image_builder directly, more details can be found in the Getting Started with RFNoC in UHD 4.0.

24

RFNoC 4 supports GNU Radio 3.8 only. Most of your RFNoC Block?s GNU Radio related changes will be due to API differences between GNU Radio 3.7 to 3.8. These changes are outside of the scope of this article. Instead, refer to GNU Radio 3.8 Migration Guide and GNU Radio Companion YAML sites for more information.

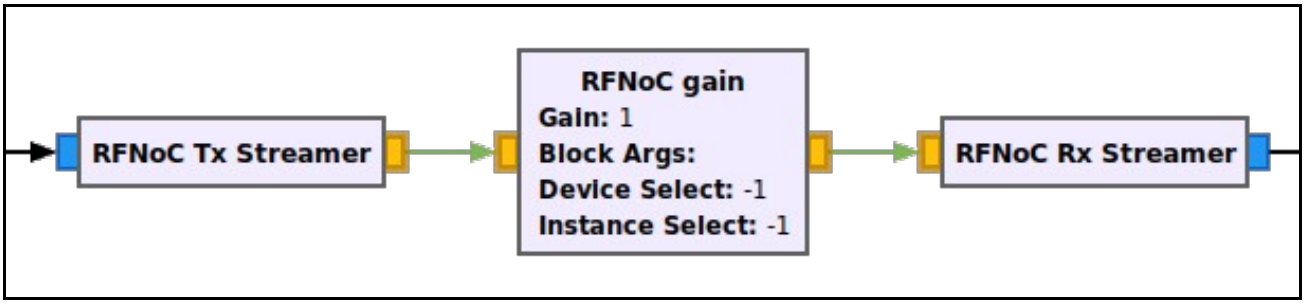
Migration reference files for this section from the Gain RFNoC Block example:

Description	RFNoC 3 Files	RFNoC 4 Files
GNU Radio Block	lib/gain_impl.cc lib/gain_impl.h include/example/gain.h	lib/gain_impl.cc lib/gain_impl.h include/example/gain.h
GRC Block Description	grc/gain.xml	grc/gain.yml
Example GRC Flowgraph	examples/gain.grc	examples/gain.grc

Note: Files are relative to the rfnoc-example directory in the respective rfnoc3 and rfnoc4 directories

24.1

When transition between a RFNoC block and a GNU Radio block or vice versa, you must insert either a RX stream or TX streamer block respectively. This differs from RFNoC 3, where a RFNoC block could be directly connected to a GNU Radio block.



24.1.1

The base class for RFNoC Block?s in GNU Radio have a set of functions that provide a shortcut to getting and setting properties without writing custom class methods. The table below lists the functions.

Property Type	Set Property	Get Property
Integer	set_int_property(...)	get_int_property(...)
Double	set_double_property(...)	get_double_property(...)
Bool	set_bool_property(...)	get_bool_property(...)
String	set_string_property(...)	get_string_property(...)

Example code for GNU Radio Companion YAML Block Description file

```
templates:
  imports: |-
    import example
  make: |-
    example.gain(
      self.rfnoc_graph,
      uhd.device_addr(${block_args}),
      ${device_select},
      ${instance_select})
    self.${id}.set_int_property('gain', ${gain})
  callbacks:
  - set_int_property('gain', ${gain})
```

25 Getting Started with RFNoC Development

25.1

AN-823

25.2

Note: This application note is DEPRECATED. It applies only to using RFNoC in UHD 3.x. The latest Getting Started in RFNoC application note is available here: Getting Started with RFNoC in UHD 4.0.

This application note guides a user through basic information on the RFNoC architecture, installing necessary software to develop custom RFNoC blocks, also called Computation Engines (CE), and walks through the steps of creating a custom RFNoC block using an example. RFNoC is currently supported on any 3rd generation USRP hardware, currently: E310/E312, E320, N300/N310/N320/N321, and X300/X310. **However**, this document primarily covers using RFNoC for the USRP X300/X310 and E310/E312. Using RFNoC with the other USRPs is similar to that documented herein.

25.3

First sections deal with installing tools and validating correct tool installation in order to do RFNoC development. Later sections deal with creating a custom RFNoC block, using the built-in testbench architecture, building an FPGA image with the custom block and finally testing out the new block within GNU Radio.

25.4

The RFNoC code base is open source, including code that executes on the host, as well as code targeted to the USRP hardware (FPGA and microcontroller firmware). RFNoC is available under the open-source GNU Lesser General Public License (LGPL). For more information on our licensing policy, please contact info@ettus.com.

25.5

RFNoC is only supported on 3rd generation USRP hardware as noted in the Abstract.

In order to build custom USRP FPGA images and RFNoC blocks the following hardware and software are needed.

- **Ubuntu 14.04.5 or 16.04.1 (preferred):** Currently PyBOMBS (which can be used to install the *Software build tools*), works most reliably in Ubuntu, and thus, we recommend using this distribution. Also, a majority of the scripts used during the build process are Linux (Ubuntu) specific. A PC with multiple cores and 8GB+ of RAM is recommended.
- **Xilinx Vivado tools (version 2017.4):** The specific version depends on the branch and state of the FPGA code. The default install location is `/opt/Xilinx/Vivado`. Once all of the Software build tools are installed the specific version for the downloaded code can be found in the `setupenv.sh` file located in the `{USER_PREFIX}/src/uhd-fpga/usrp3/top/{DEVICE}` directory. Further information can be found here.
- **Software build tools:** If UHD can be or has been compiled from source on the development PC then all the necessary software build components are present (PyBOMBS can be used to set all this up and instructions on how to do so are given in a following step).
- Any 3rd generation USRP hardware as noted in the Abstract.

NOTE:

- The edition of Xilinx Vivado that is required will depend on which USRP device is being used.
  - X3xx series devices: Design Edition or System Edition.
  - E3xx series devices: Design Edition, System Edition, or the free WebPack Edition.
- Other operating systems can be used, but the exact steps on how to proceed are not given in this Application Note.
- In some Linux distributions (e.g. Ubuntu) `dash` is set as default shell, which may cause some issues. It is recommended to set the shell to `bash` by running the following commands in the terminal. Choose `<No>` when prompted by the first command and the second command will validate the that bash will be used.

```
$ sudo dpkg-reconfigure dash
$ ll /bin/sh
```

25.6

While this Application Note goes through the process of integrating GNU Radio into the RFNoC development flow, it is by no means required to use or develop within the RFNoC framework, but it makes it a great deal easier to use a framework on top of RFNoC for aspects such as visualization and other features. GNU Radio is freely available and more information about it can be found here.

The following software packages are required in order to setup a development environment/sandbox:

- UHD
- GNU Radio
- gr-ettus

25.6.1

The cleanest way to set this up is to install everything into a dedicated directory. PyBOMBS is the simplest way to do this. If not already installed, PyBOMBS can be setup with the following commands:

```
$ sudo apt-get install git
$ sudo apt-get install python-setuptools python-dev python-pip build-essential

$ sudo pip install git+https://github.com/gnuradio/pybombs.git
$ pybombs recipes add gr-recipes git+https://github.com/gnuradio/gr-recipes.git
$ pybombs recipes add ettus git+https://github.com/EttusResearch/ettus-pybombs.git
```

These commands will do the following:

- Install `Git`

- Install `pip` and other Python dependencies
- Install the latest `PyBOMBS` from its Git repository
- Add the `gr-recipes` recipes which are used to install GNU Radio specific software
- Add the `ettus` recipes which are used to install Ettus Research specific software

From here, PyBOMBS can be used to setup and install the development environment/sandbox by running the following command:

```
$ pybombs prefix init ~/rfnoc -R rfnoc -a rfnoc
```

This will do the following:

- Create a directory in the user?s home directory called `rfnoc` (any valid directory name will work)
- Give the prefix an alias of `rfnoc` (`[-a alias]`, e.g. `?a rfnoc`), which would be the name given to this path. This name will be used in further steps that use PyBOMBS. When creating the first prefix and omitting the alias, the prefix will be setup as the default.
- Use the `rfnoc` prefix recipe (as opposed to a package recipe like `gqrx`) to clone UHD, FPGA, GNU Radio, and gr-ettus sources into the `~/rfnoc` directory as well as compile and install all the software

NOTE: A user can specify how many cores are used by builds when using PyBOMBS. The default is set to 4. For example, this will set the number of cores used to 3:

```
$ pybombs config makewidth 3
```

The value will be written into a configuration file and then applied to subsequent PyBOMBS commands. This value can temporarily be overridden for a specific build by specifying the `--config makewidth=X` argument, where ?`x`? is an integer number. If the user only has 4 cores it is recommend to use this argument in the pybombs command to limit the number of cores to <4 (e.g. 3) so that the computer stays responsive. Following are 2 examples, one using less cores and the other using more cores:

```
$ pybombs --config makewidth=3 prefix init ~/rfnoc -R rfnoc -a rfnoc
$ pybombs --config makewidth=7 prefix init ~/rfnoc -R rfnoc -a rfnoc
```

Then, it is necessary to setup the PyBOMBS environment, so that the system/terminal session will have the environmental variables pointing to this newly created prefix, which is done with the following commands:

```
$ cd ~/rfnoc
$ source ./setup_env.sh
```

Once the previous command is run, this terminal session will have access to the environmental variables that allow the complete use of the set of software that was just installed with PyBOMBS. If access to the software is needed in other terminals the same command must be run within them.

NOTE: Throughout the rest of this document the term `{USER_PREFIX}` will used at the beginning of different directories. For example, `{USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts` is a directory that contains useful scripts for compiling. The term `{USER_PREFIX}` is used to denote the folders that precede the `/src` directory. Examples of what `{USER_PREFIX}` could be: `/home/user/rfnoc` or `/home/user/myDevfolder/`. On many Linux environments using `~/` at the beginning of the target directory path is equivalent to the user?s home directory.(i.e `~/` is equal to `/home/user/`). So `{USER_PREFIX}` could also look like `~/rfnoc` or `~/myDevfolder/`.

25.6.2

As an alternative to using PyBOMBS, manually installing and configuring the software is done by following the individual install notes for GNU Radio, UHD and gr-ettus and by making sure they are reachable by linkers and compilers.

NOTE: The Application Note found here goes through the process of manually installing UHD and GNU Radio on Linux platforms.

To manually download the software, use these `git clone` commands, which will select the correct branches:

```
$ git clone --recursive -b rfnoc-devel https://github.com/EttusResearch/uhd.git
$ git clone --recursive -b maint https://github.com/gnuradio/gnuradio.git # master branch is also fine instead of maint
$ git clone -b master https://github.com/EttusResearch/gr-ettus.git
$ git clone -b rfnoc-devel https://github.com/EttusResearch/fpga.git
```

If UHD, GNU Radio and/or gr-ettus are already installed, it would be sufficient to checkout the branches mentioned and update them them (`git pull`). Thereafter, rebuild each of the repositories (rebuild order: UHD, GNU Radio, gr-ettus).

25.6.3

Running the command ?`uhd_config_info`? with the ?`--version`? flag will verify that the installation has been completed successfully.

NOTE: The version string output from this command may differ, however it should be similar to the output below.

```
$ uhd_config_info --version
linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_4.0.0.rfnoc-devel-161- g83150fdd

4.0.0.rfnoc-devel-161-g83150fdd
```

25.6.4

It is recommended to spend a moment looking at the Ettus Research default image, which is pre-built with a set of RFNoC blocks, as well as building a custom image with a unique set of pre-built RFNoC blocks. To get the default image(s), run the following command:

```
$ uhd_images_downloader
```

Ettus Research will be updating the default image(s) occasionally, and `uhd_images_downloader` can be run anytime after running `git pull` and re-installing to pull the most current images. Images are stored in the `{USER_PREFIX}/share/uhd/images` directory.

The following images have the corresponding RFNoC blocks (Computation Engines):

Image Name	Included Blocks
usrp_x300_fpga_HG.bit	2x DDC, 2x DUC
usrp_x300_fpga_XG.bit	

```
usrp_x310_fpga_HG.bit

usrp_x310_fpga_XG.bit
usrp_x300_fpga_RFNOC_HG.bit
                                        DUC, DDC (one channel), fosphor, window, fft, 2x AXI FIFOs
usrp_x300_fpga_RFNOC_XG.bit
usrp_x310_fpga_RFNOC_HG.bit
                                        DUC, DDC (one channel), fosphor, window, fft, 2x AXI FIFOs, Keep One in N, FIR, Siggen
usrp_x310_fpga_RFNOC_XG.bit
usrp_e310_fpga.bit
                                        1x DDC, 1x DUC
usrp_e310_fpga_sg3.bit
usrp_e310_fpga_RFNOC.bit (sg1 version)
                                        fosphor, window, fft, 2x AXI FIFOs, FIR
usrp_e310_fpga_RFNOC_sg3.bit
```

Instructions on flashing the image to a device can be found here for X3xx series and here for E3xx series.

NOTE: FPGA images are specific to the USRP device **NOT** the USRP series. For example, a USRP X300 FPGA image will **NOT** work on a USRP X310 and vice versa. Loading an image that does not correspond to a USRP device will likely brick the device.

By following the steps above the following should now be available:

- UHD/RFNoC code downloaded and installed
- FPGA code available
- A valid RFNoC image on your X3xx or E3xx series device

25.6.4.1

Run the following command, with a USRP connected to your PC, to verify current image on the USRP.

```
$ uhd_usrp_probe
```

If an RFNoC image was successfully loaded onto the USRP, there will be a lot of output text (RFNoC code is currently very verbose). The final lines of the output should be similar to the following for an USRP X310 (e.g. `usrp_x310_fpga_HG`):

```
|   |     ____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * DDC_0
|   |   |   * DDC_1
|   |   |   * DUC_0
|   |   |   * DUC_1
```

Final output for `usrp_x310_fpga_RFNOC_HG.bit` image:

```
|   |     ____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * DDC_0
|   |   |   * DUC_0
|   |   |   * FFT_0
|   |   |   * Window_0
|   |   |   * FIR_0
|   |   |   * SigGen_0
|   |   |   * KeepOneInN_0
|   |   |   * fosphor_0
|   |   |   * FIFO_0
|   |   |   * FIFO_1
```

NOTE: The actual names and number of blocks can differ. The list of blocks should start with the `DmaFIFO_x` and `Radio_x`, and then a couple more lines of block IDs should follow.

25.6.4.2

Because of the growing number of RFNoC blocks, the user has the option to build an FPGA image with a set of pre-built RFNoC blocks of their choosing. The following steps describe the process for doing this and by so doing will also validate proper tool installation. Because compilation can take a couple of hours, it is recommended the user begin this process while continuing the rest of this guide.

NOTE: FPGA compilations can run in the background, however they are very resource intensive. If the user intents to use the same computer that is compiling to walk through the rest of this Application Note, it is recommended that the computer has plenty of resources.

The script to initiate a compile is called `uhd_image_builder.py`, and is located in the `{USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts` directory. Run the help menu by typing:

```
$ cd {USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts
$ ./uhd_image_builder.py --help
```

A more detailed discussion of this script is given in an upcoming section. For now, compiling an FPGA image that has 2 RFNoC blocks (`window` and `fft`) and some `FIFOs`, is done by running the script with the following arguments.

Example for an X310 USRP:

```
$ ./uhd_image_builder.py window fft -d x310 -t X310_RFNOC_HG -m 5 --fill-with-fifos
```

Example for an E310 USRP with Speed Grade 3 (sg3) FPGA:

```
$ ./uhd_image_builder.py window fft -d e310 -t E310_RFNOC_sg3 -m 5 --fill-with-fifos
```

At the end of a successful compilation process, write the new image to a USRP. If the image was compiled for a USRP X310, the following command will load the new image. Update the `{IP_Address}` of the USRP and `{USER_PREFIX}` to the appropriate values for your configuration before running the command.

```
$ uhd_image_loader --args "type=x300,addr={IP_ADDRESS}" --fpga-path {USER_PREFIX}/src/uhd-fpga/usrp3/top/x300/build/usrp_x310_fpga_RFNOC_H
```

NOTE: FPGA images are specific to the USRP device **NOT** the USRP series. For example, a USRP X300 FPGA image will **NOT** work on a USRP X310 and vice versa. Loading an image that does not correspond to a USRP device will likely brick the device. Additional instructions on flashing a custom image to a device can be found here for X3xx series and here for E3xx series.

After the image has been successfully written to the USRP, power-cycle it and run the ?`uhd_usrp_probe`? utility to view the newly compiled blocks.

```
$ uhd_usrp_probe
```

The final lines of output for the image built for the X310 is as follows:

```
|   |     ____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * Window_0
|   |   |   * FFT_0
|   |   |   * FIFO_0
|   |   |   * FIFO_1
|   |   |   * FIFO_2
```

25.6.5

The following new examples included within the `rfnoc-devel` branch of UHD, are a good reference on how to use RFNoC from UHD.

The following example is based off of `rx_samples_to_file.cpp`. The example can be configured to place an RFNoC block in between the radio and host.

```
rfnoc_rx_to_file.cpp
```

This next example chains a null source to another block and streams the data to the host.

```
rfnoc_nullsource_ce_rx.cpp
```

These examples demonstrate the core features and flexibility of RFNoC.

For more information on UHD and UHD development please refer to the UHD Software Resource page, Getting Started with UHD and C++ Application Note or directly to the UHD user manual.

25.6.6

A good way of getting started with RFNoC in a more visual way is to use GNU Radio. The `gr-ettus` out-of-tree module (OOT) allows a user to use RFNoC blocks in their local GNU Radio / GNU Radio Companion (GRC) installation. This GNU Radio OOT contains blocks that allow you to configure your FPGA through GRC.

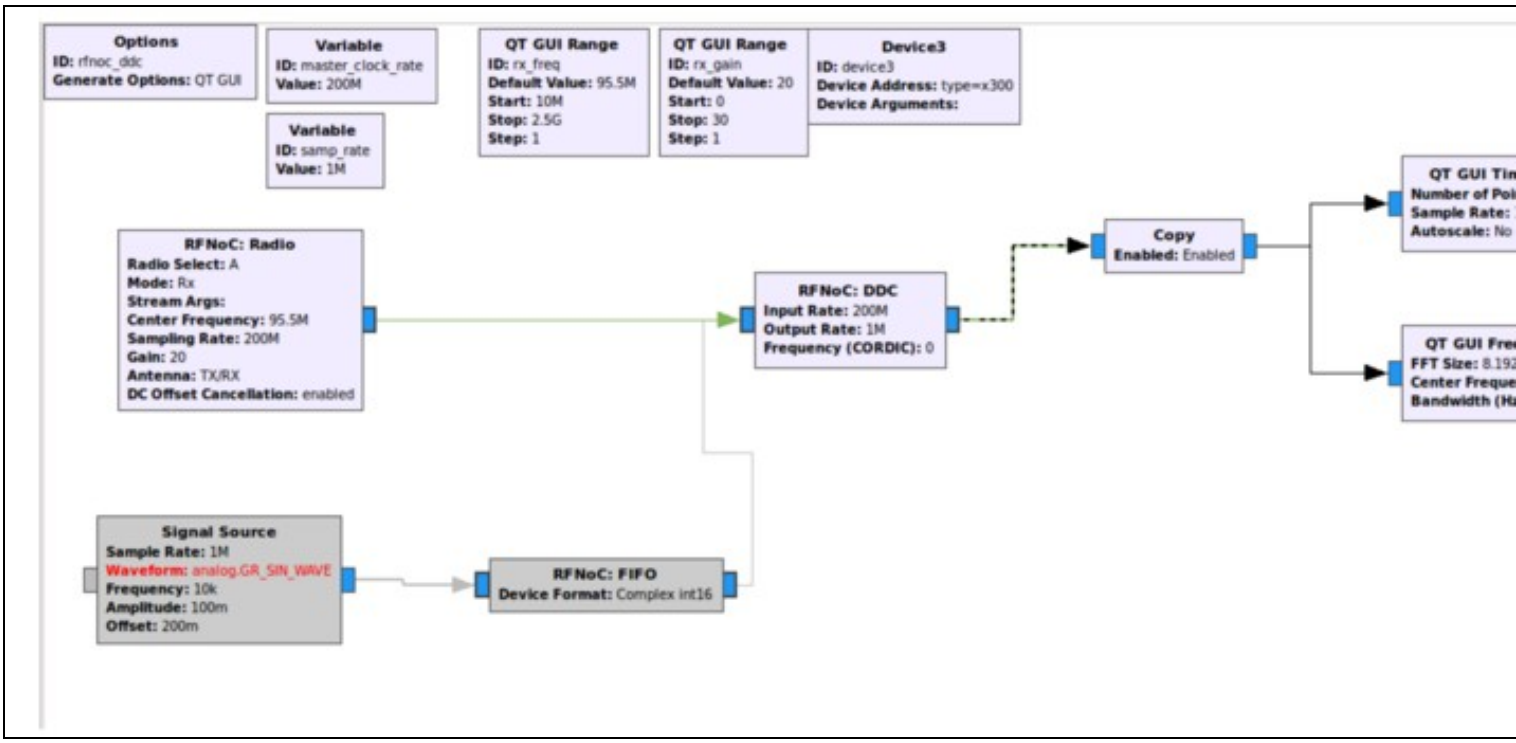
NOTE: As blocks in the `gr-ettus` OOT mature, they will be upstreamed to `gr-uhd`. Also, `gr-ettus` is a container used by Ettus Research to disseminate experimental or under-development features for `gr-uhd`. It is not a replacement for `gr-uhd` (in fact, the latter is a requirement for `gr-ettus`).

Examples can be run from `gr-ettus/examples/rfnoc`, provided that the appropriate RFNoC blocks are compiled into the FPGA image currently running on the USRP.

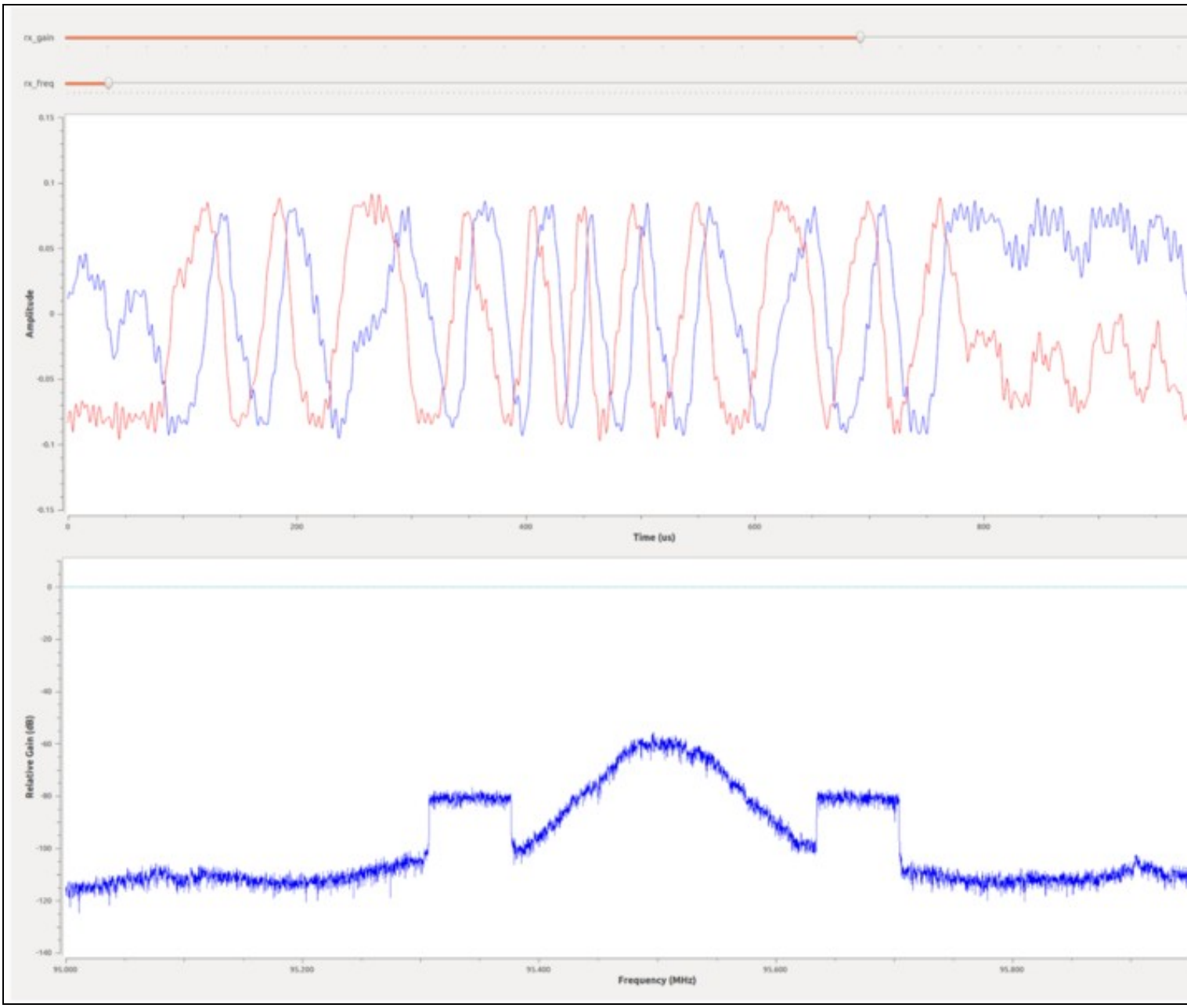
A couple of rules for building GNU Radio flowgraphs with RFNoC blocks:

- You always need a `Device3` object in your flow graph (it does not get connected, see screenshot below).
- You should have at least two RFNoC blocks connected together. Going `GNU Radio Block` -> `RFNoC Block` -> `GNU Radio Block` is not recommended (it will work, but with suboptimal performance).

The GNU Radio flowgraph `rfnoc_ddc.grc` is an example that can be run using the default RFNoC image. Below are screenshots of the flowgraph and what it produces.



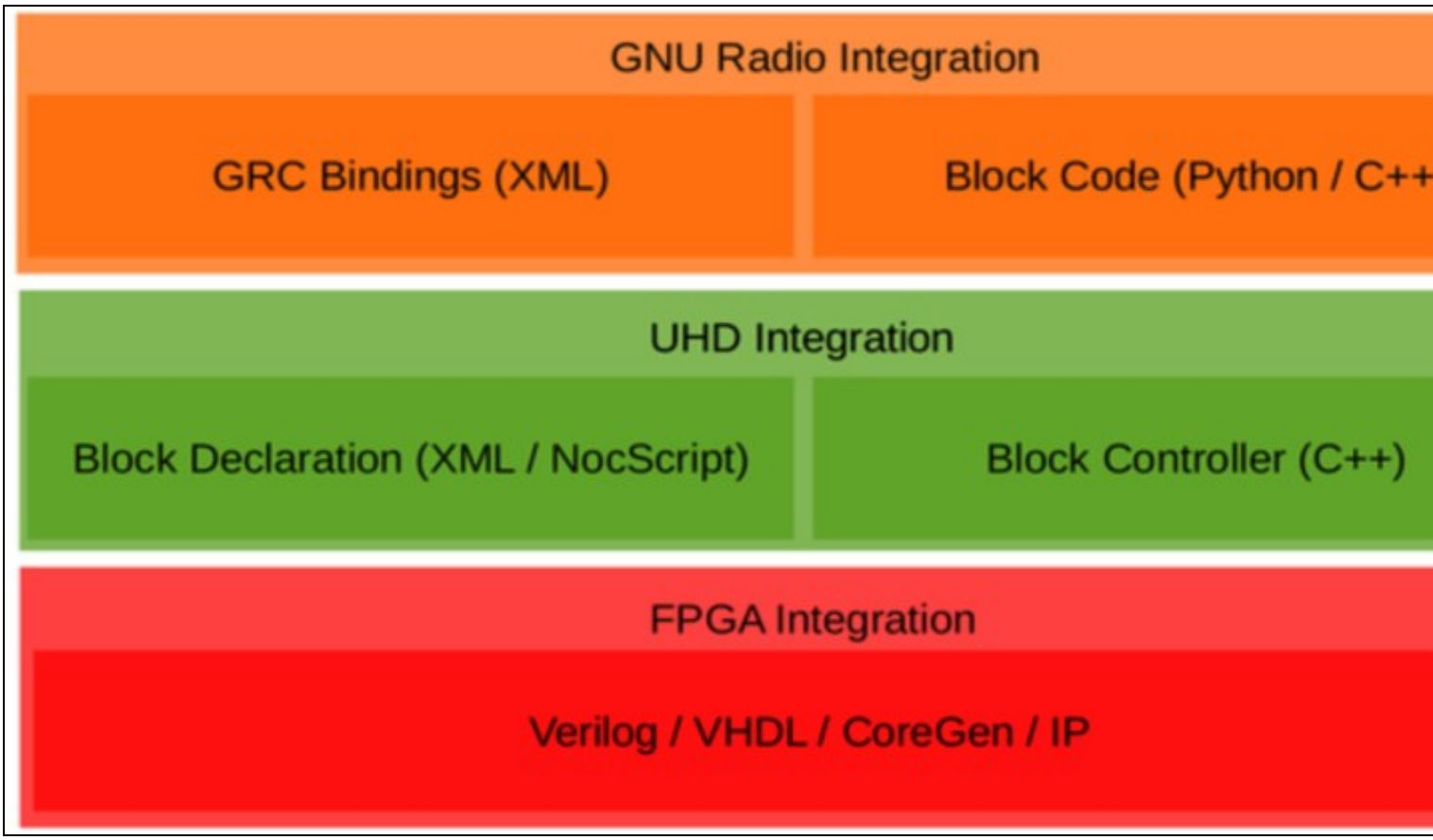
NOTE: Copy Block: In the RFNoC domain, streams of data can not be split as easily as they are in the GNU Radio domain. The ?copy? block depicted in the screenshot above serves the function as a stream splitter. Its main purpose, when ?enabled?, is to copy the samples it is getting at its input and put them into the output, but here it is also serving as a boundary between a RFNoC-domain and a GNURadio-domain. In the flowgraph above, after this boundary is passed, the data stream can easily be split into the two sinks to have them run simultaneously (standard GNU Radio functionality). It is possible to connect the GNU Radio blocks directly to RFNoC blocks without a ?copy? block, but only one would work at a time (the other ones would have to be disabled). Another way to split data streams from the RFNoC-domain is to use the ?RFNoC: split stream? block, which would split the streams in the RFNoC domain, but this is not very useful here as we are, in any case, moving into the GNURadio-domain.



For more information on GNURadio development please refer to the GNURadio user's manual and API.

25.7

The figure below shows the basic structure of the RFNoC Stack. Corresponding code is needed in each of the three sections in order to build a custom RFNoC block with GNU Radio integration. A tool called RFNoC Modtool was created in order to minimize the effort needed to implement a new RFNoC block. RFNoC Modtool creates a custom GNU Radio OOT module with the basic structure and the necessary files for each of these sections. RFNoC Modtool is currently a part of the GNU Radio OOT module `gr-ettus`.



25.7.1

NOTE: Console outputs may vary depending on the version of UHD the user is running. However, functionality should be the same or similar.

Because the RFNoC Modtool has similar functionality to the `gr_modtool` [gr_modtool] provided by GNU Radio, those that have worked with `gr_modtool` in the past will find the RFNoC Modtool familiar.

To check the usage of the tool, run the following:

```
$ rfnocmodtool help
linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_4.0.0.rfnoc-devel-162-g335a1317

Usage:
rfnocmodtool <command> [options] -- Run <command> with the given options.
rfnocmodtool help -- Show a list of commands.
rfnocmodtool help <command> -- Shows the help for a given command.

List of possible commands:

Name      Aliases          Description
====================================================================
disable   dis              Disable block (comments out CMake entries for files)
info      getinfo,inf      Return information about a given module
remove    rm,del           Remove block (delete files and remove Makefile entries)
makexml   mx               Make XML file for GRC block bindings
add       insert           Add block to the out-of-tree module.
newmod    nm,create        Create a new out-of-tree module
rename    mv               Rename a block in the out-of-tree module.
```

25.7.2

To start generating an RFNoC OOT module navigate to the source location (i.e. `cd ~/{USER_PREFIX}/src`) and type:

```
$ rfnocmodtool newmod [NAME OF THE MODULE]
```

Where `[NAME OF THE MODULE]` is a name the user gives the new module. In the following, a module is created with the name ?`tutorial`?. If the user does not write the name of the module following the `newmod` command the tool will ask for it interactively. Running this command will create a folder containing the basic folders that you may need for a functional module.

```
$ rfnocmodtool newmod tutorial
linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_4.0.0.rfnoc-devel-162-g335a1317

Creating out-of-tree module in ./rfnoc-tutorial... Done.
Use 'rfnocmodtool add' to add a new block to this currently empty module.
```

To see what files and directories were created run:

```
$ ls rfnoc-tutorial/
apps  cmake  CMakeLists.txt  docs  examples  grc  include  lib  MANIFEST.md  python  README.md  rfnoc  swig
```

In contrast with `gr_modtool`, this includes a folder called `rfnoc`, which is where the UHD/FPGA files are located.

25.7.3

In order to add blocks to a module, navigate to the folder just created and use the `add` command of `rfnocmodtool`. Continuing with the example above, run the following:

```
$ cd rfnoc-tutorial
$ rfnocmodtool add [NAME OF THE BLOCK]
```

For demonstrative purposes, a block named `gain` will be created. The `gain` block will multiply samples that pass through it by a constant. As before, if the name is not given, the tool will ask the user for the name. There are several arguments that can be passed to the tool, but running the tool without any of these arguments will give the following interactive parsing output:

```
$ rfnocmodtool add gain
linux; GNU C++ version 5.4.0 20160609; Boost_105800; UHD_4.0.0.rfnoc-devel-162-g335a1317

RFNoC module name identified: tutorial
Block/code identifier: gain
Enter valid argument list, including default arguments:
Block NoC ID (Hexadecimal): 1111222233334444
Skip Block Controllers Generation? [UHD block ctrl files] [y/N] N
Skip Block interface files Generation? [GRC block ctrl files] [y/N] N
```

Hitting `enter` on each one of the options will take the default values.

The following is a description of the valid argument list items:

- **NoC ID:** This ID is a Hexadecimal number which serves as identification between the hardware part and the software part of the design. It can be as long as 16 0-9 A-F digits. If a NoC ID is not provided, it will be set to a random number.
- **Block Controllers Generation:** The block controllers are the C++ control that the user can apply to the UHD-part of the design. In these files, the user can add more control over this layer of the design. Depending on the complexity of the block it may be possible to add all necessary control using NoCScript (more details on NoCScript can be found in the section labeled UHD Integration). In this case the cpp/hpp block control files generation are not needed. Default is to generate as their existence will be ignored if not edited.
- **Block Interface:** Add more design specific functionality to the design at the GNU Radio interface by generating these block-interface files and adding necessary logic. Depending on the complexity of the block it may be possible to add all necessary control using NoC-Script. In this case the block-interface files are not needed. Default is to generate as their existence will be ignored if not edited.

NOTE: If the user does not intend to use the block controllers or is not sure if they are needed, the presence of them in the design will do no harm. It is recommended to add them. This leaves the possibility to add more functions inside them in a future stage of development.

After finishing the parsing, the following files will be generated/edited:

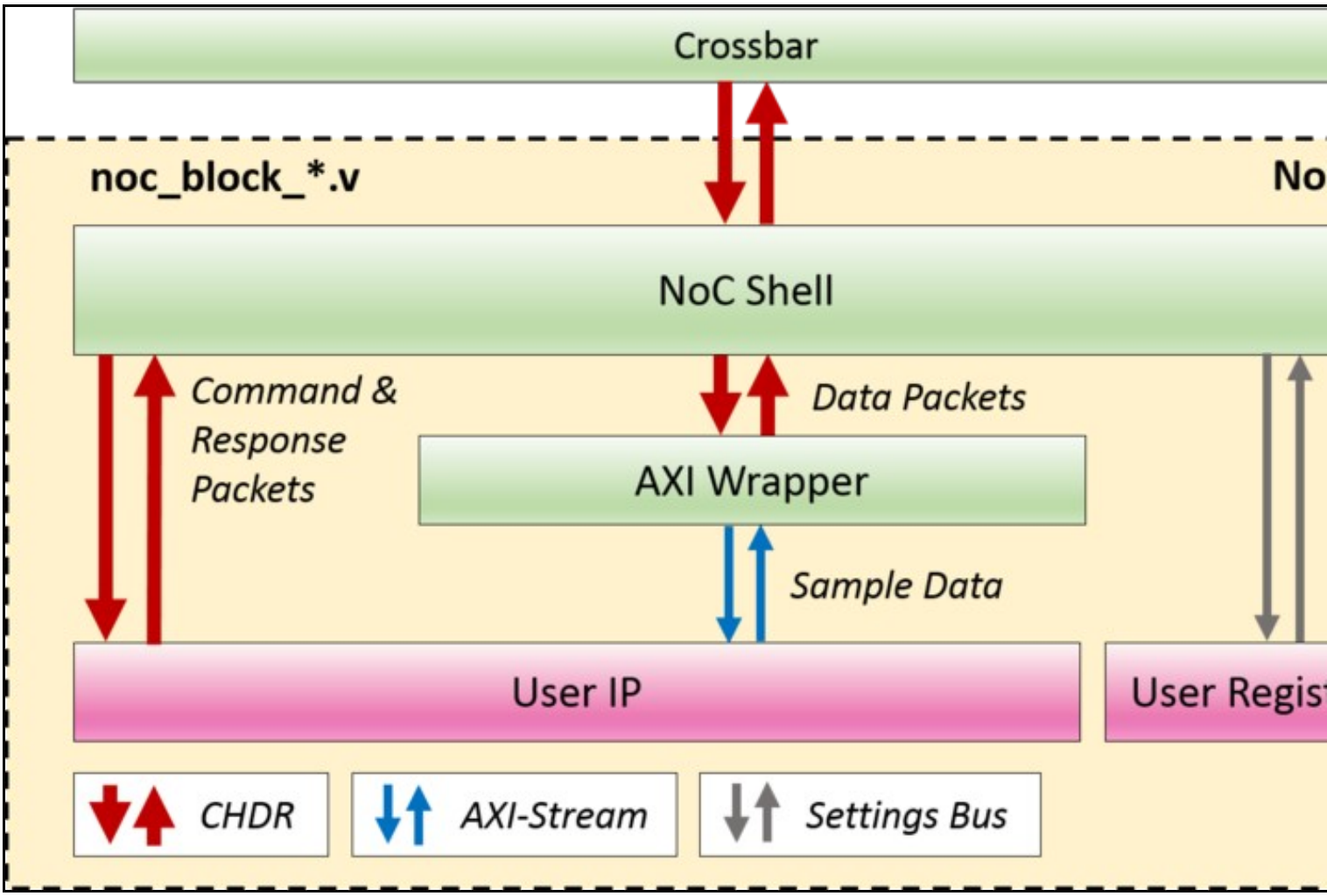
```
Adding file 'lib/gain_impl.h'...
Adding file 'lib/gain_impl.cc'...
Adding file 'include/tutorial/gain.h'...
Adding file 'include/tutorial/gain_block_ctrl.hpp'...
Adding file 'lib/gain_block_ctrl_impl.cpp'...
Editing swig/tutorial_swig.i...
Adding file 'python/qa_gain.py'...
Editing python/CMakeLists.txt...
Adding file 'grc/tutorial_gain.xml'...
Adding file 'rfnoc/blocks/gain.xml'...
Adding file 'rfnoc/fpga-src/noc_block_gain.v'...
rfnoc/testbenches/noc_block_gain_tb folder created
Adding file 'rfnoc/testbenches/noc_block_gain_tb/noc_block_gain_tb.sv'...
Adding file 'rfnoc/testbenches/noc_block_gain_tb/Makefile'...
Adding file 'rfnoc/testbenches/noc_block_gain_tb/CMakeLists.txt'...
```

25.8

25.8.1

RFNoC blocks or Computation Engines (CEs) in the FPGA use a NoC Shell instance to interface with the rest of RFNoC. NoC Shell implements RFNoC's core functionality: packet muxing and demuxing, flow control, and the settings register bus (i.e. write/read control/status registers). The NoC Shell has an interface to the RFNoC AXI stream crossbar and a user interface. NoC Shell AXI stream interfaces expect CHDR packets with a proper header. See the manual for information on CHDR and SID.

NOTE: AXI Stream is an ARM AMBA standard interface. Xilinx has an AXI Reference Guide with more details on this standard.



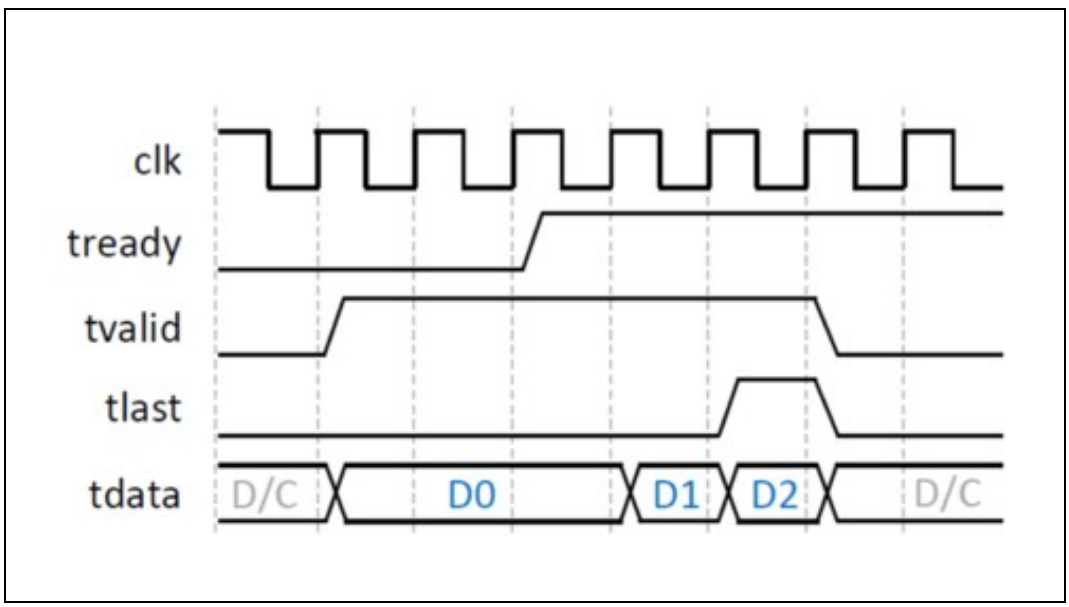
Many designs will want to use an AXI Stream interface with only sample data. However, as stated earlier, the NoC Shell block expects CHDR packets. To ease interfacing user code, the AXI Wrapper block provides the necessary logic to strip and insert the CHDR header, effectively converting packetized sample data into streaming sample data and vice versa. The example RFNoC blocks `noc_block_fft.v` and `noc_block_fir.v` show how AXI Wrapper is used to implement existing Xilinx AXI Stream based IP within a computation engine.

NOTE: AXI Wrapper also supports AXI Stream buses for configuration. These buses are driven via the setting register bus and do not have back pressure. They also consume two user register addresses per bus.

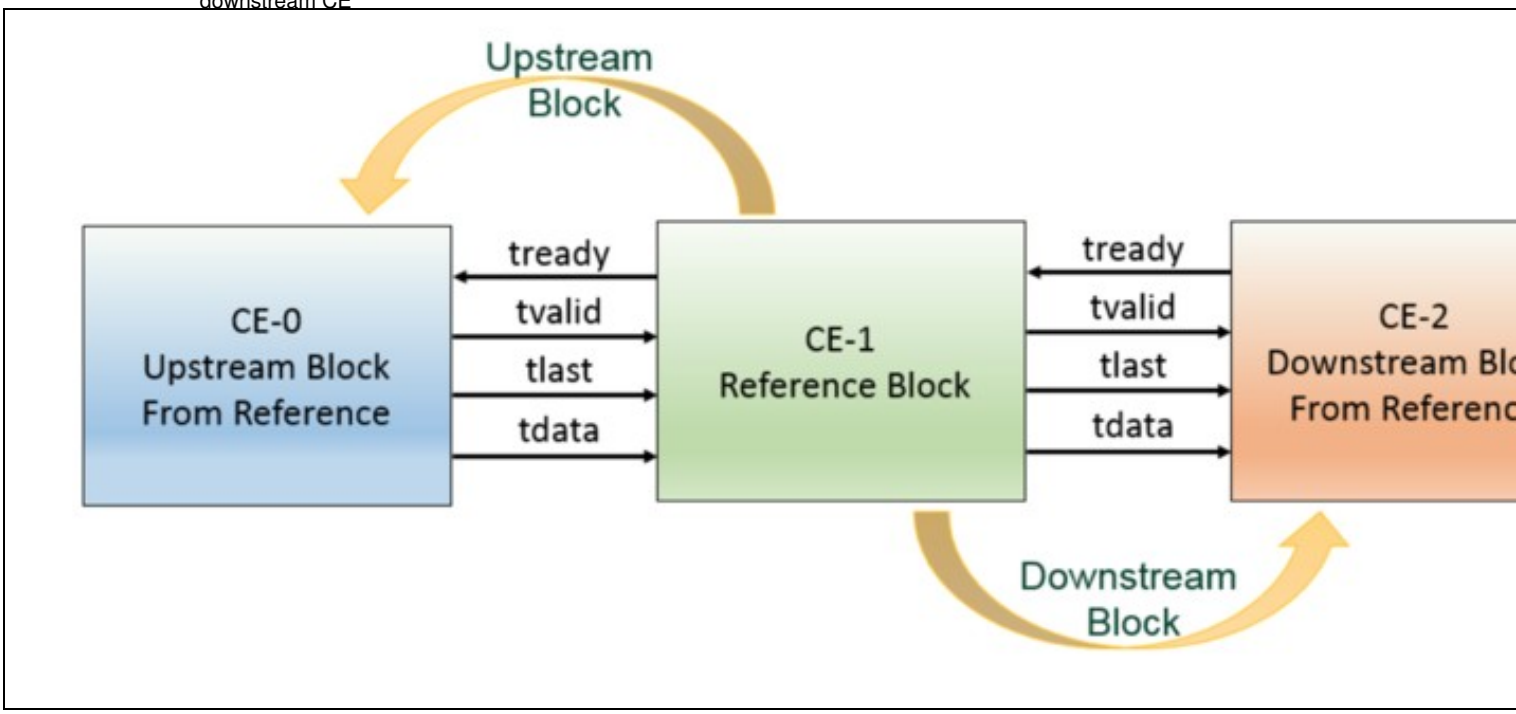
The primary user interface consists of four AXI stream interfaces (`tready, tvalid, tlast, tdata`) and a settings register bus (8-bit, valid user register addresses: `128-255`).

AXI Stream signals:

- **m_axis_data_tdata:** Input sample data packets
  - Data coming from host or another CE
- **s_axis_data_tdata:** Output sample data packets
  - Data going to another CE or host
- **m_axis_data_tready:** Input signal to CE
  - Used to notify CE that downstream CE is ready for data
- **s_axis_data_tready:** Output signal to CE
  - Used to notify upstream CE that CE is ready for data
- **m_axis_data_tvalid:** Input signal to CE
  - Used to indicate upstream CE has valid data

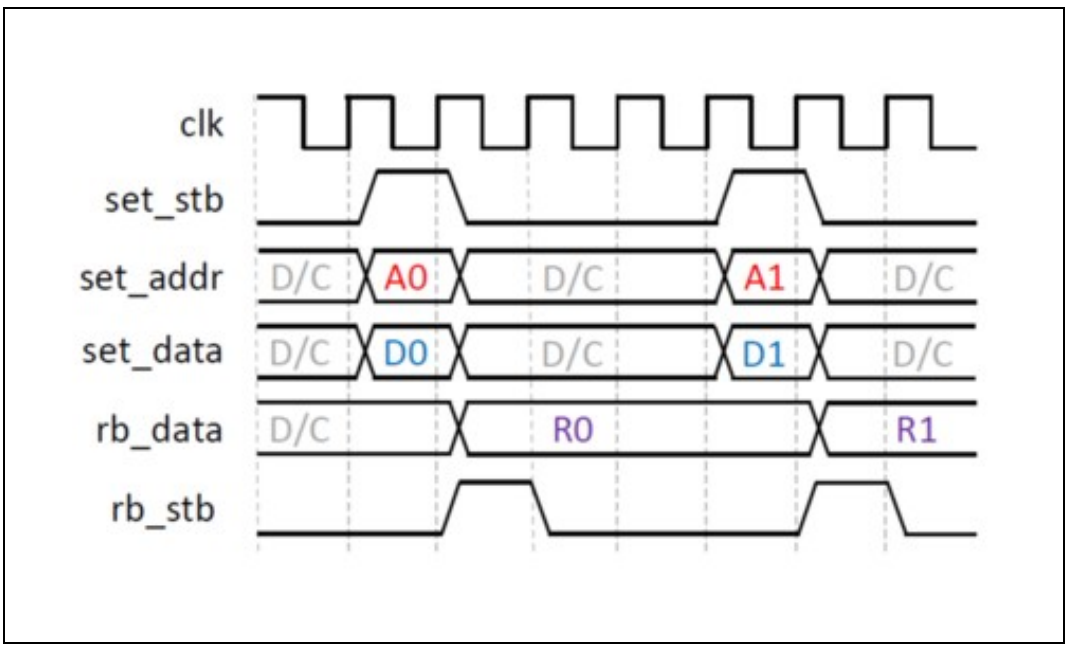


- **s_axis_data_tvalid:** Output signal to CE
    - Used to indicate to downstream CE that CE has valid data
- **m_axis_data_tlast:** Input signal to CE
    - Used to delimit packets from upstream CE
- **s_axis_data_tlast:** Output signal to CE
    - Used to delimit packets to downstream CE

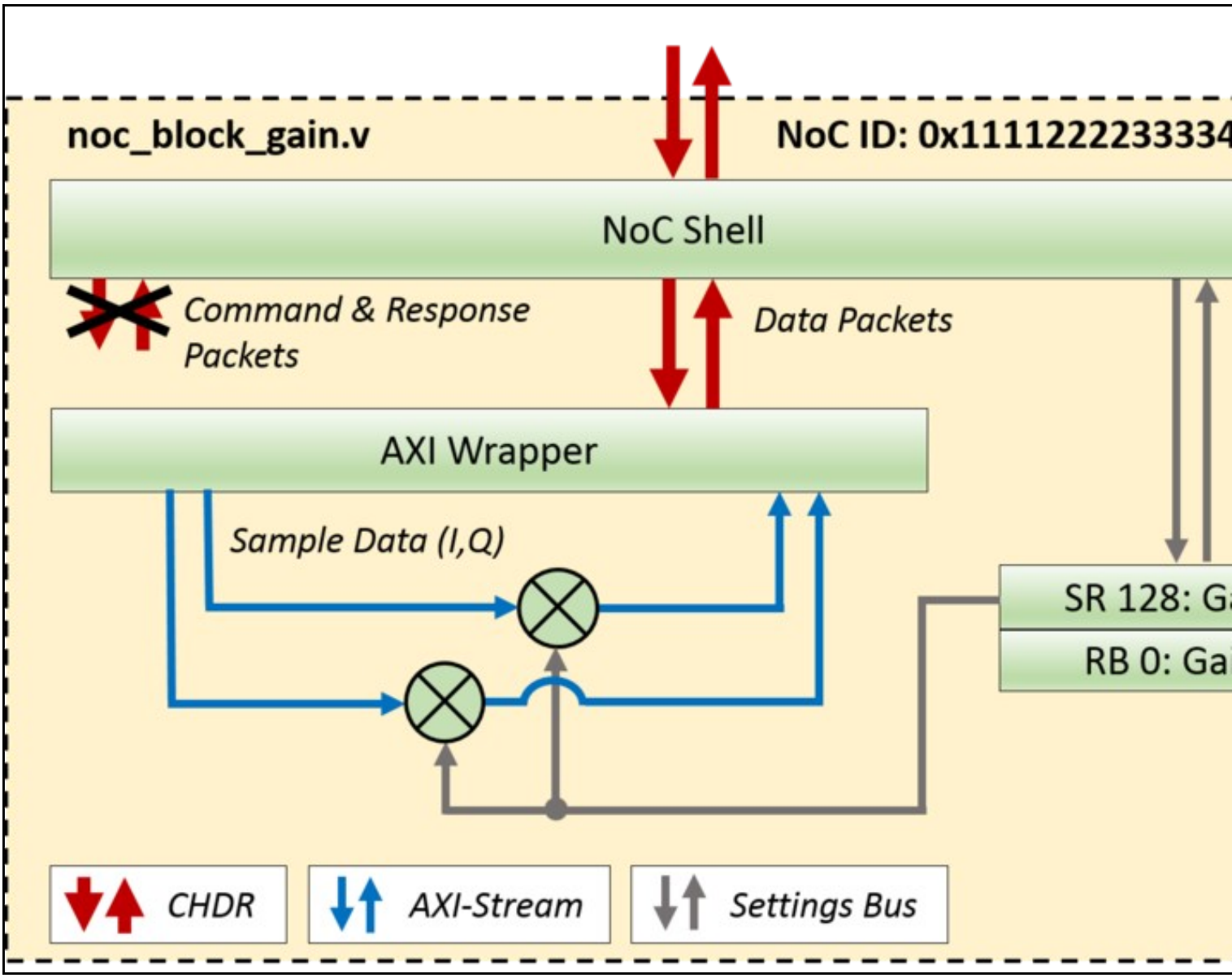


Settings Bus signals:

- **set_stb:** Assert to write **set_data** to register at **set_addr**ess
- **set_addr:** Register address to set
- **set_data:** Data to set
- **rb_data:** Data to read back
- **rb_strobe:** Assert to read **rb_data** from register at **set_addr**ess



For the `gain` example block the following architecture is desired:

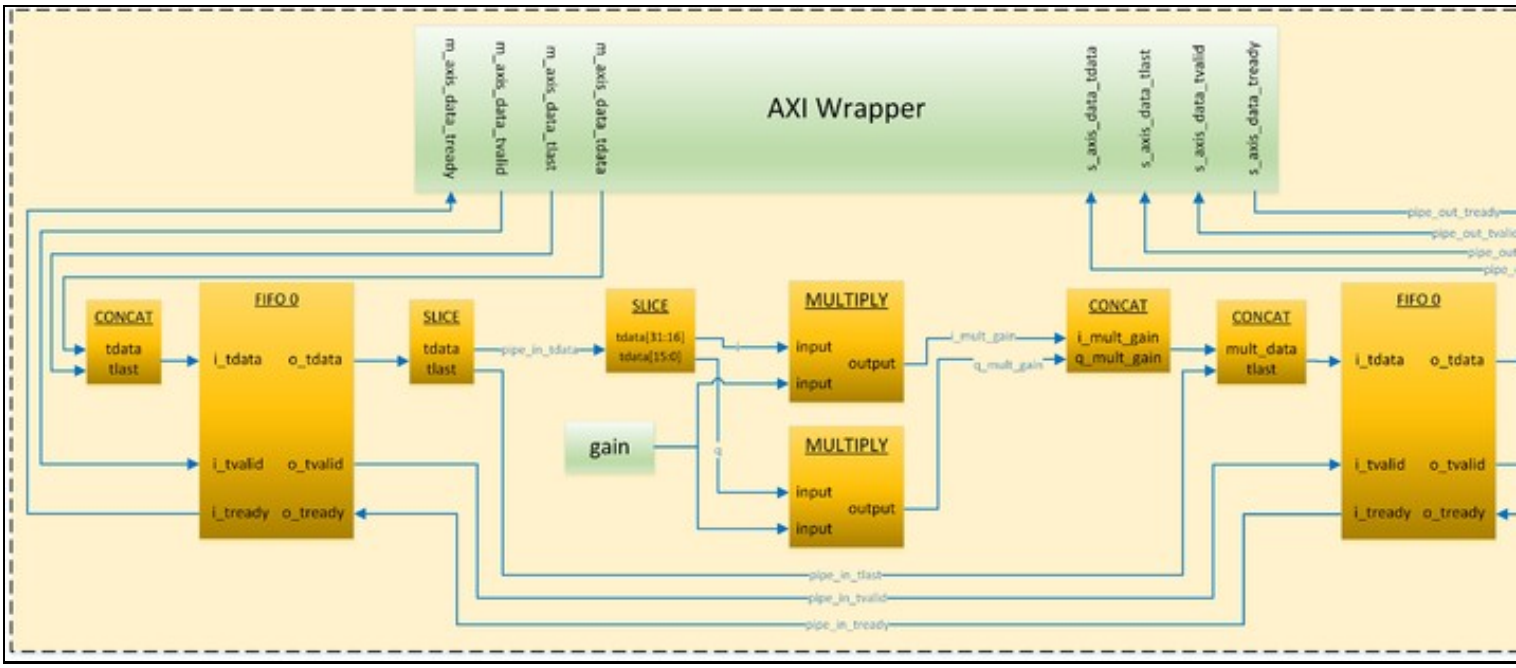


Open the file `rfnoc-tutorial/rfnoc/fpga-src/noc_block_gain.v` that contains the RFNoC block skeleton code that was created when the `$ rfnocmodtool add gain` command was run and modify the following (**BOLD** indicates changes to the skeleton code).

```
localparam [7:0] SR_GAIN = SR_USER_REG_BASE;
   localparam [7:0] SR_TEST_REG_1 = SR_USER_REG_BASE + 8'd1;

   wire [15:0] gain;
   setting_reg #(
     .my_addr(SR_GAIN), .awidth(8), .width(16))
   sr_gain (
     .clk(ce_clk), .rst(ce_rst),
     .strobe(set_stb), .addr(set_addr), .in(set_data), .out(gain), .changed());

    always @(posedge ce_clk) begin
       case(rb_addr)
         8'd0 : rb_data <= {48'd0, gain};
         8'd1 : rb_data <= {32'd0, test_reg_1};
         default : rb_data <= 64'h0BADC0DE0BADC0DE;
       endcase
    end

    wire [31:0] pipe_in_tdata;
    wire pipe_in_tvalid, pipe_in_tlast;
    wire pipe_in_tready;

    wire [31:0] pipe_out_tdata;
    wire pipe_out_tvalid, pipe_out_tlast;
    wire pipe_out_tready;

    // Adding FIFO to ensure Pipeline
    axi_fifo_flop #(.WIDTH(32+1))
    pipeline0_axi_fifo_flop (
      .clk(ce_clk),
      .reset(ce_rst),
      .clear(clear_tx_seqnum),
      .i_tdata({m_axis_data_tlast,m_axis_data_tdata}),
      .i_tvalid(m_axis_data_tvalid),
      .i_tready(m_axis_data_tready),
      .o_tdata({pipe_in_tlast,pipe_in_tdata}),
      .o_tvalid(pipe_in_tvalid),
```

```
    .o_tready(pipe_in_tready));

  wire [15:0] i = pipe_in_tdata[31:16];
  wire [15:0] q = pipe_in_tdata[15:0];

  wire [31:0] i_mult_gain = i*gain;
  wire [31:0] q_mult_gain = q*gain;

  wire [31:0] mult_gain = {i_mult_gain[15:0], q_mult_gain[15:0]};
  axi_fifo_flop #(.WIDTH(32+1))
  pipeline1_axi_fifo_flop (
    .clk(ce_clk),
    .reset(ce_rst),
    .clear(clear_tx_seqnum),
    .i_tdata({pipe_in_tlast,mult_gain}),
    .i_tvalid(pipe_in_tvalid),
    .i_tready(pipe_in_tready),
    .o_tdata({pipe_out_tlast,pipe_out_tdata}),
    .o_tvalid(pipe_out_tvalid),
    .o_tready(pipe_out_tready));

  /* Output Signals */
  assign pipe_out_tready = s_axis_data_tready;
  assign s_axis_data_tvalid = pipe_out_tvalid;
  assign s_axis_data_tlast  = pipe_out_tlast;
  assign s_axis_data_tdata  = pipe_out_tdata;
```

The following is a block diagram of the code created by the above Verilog:



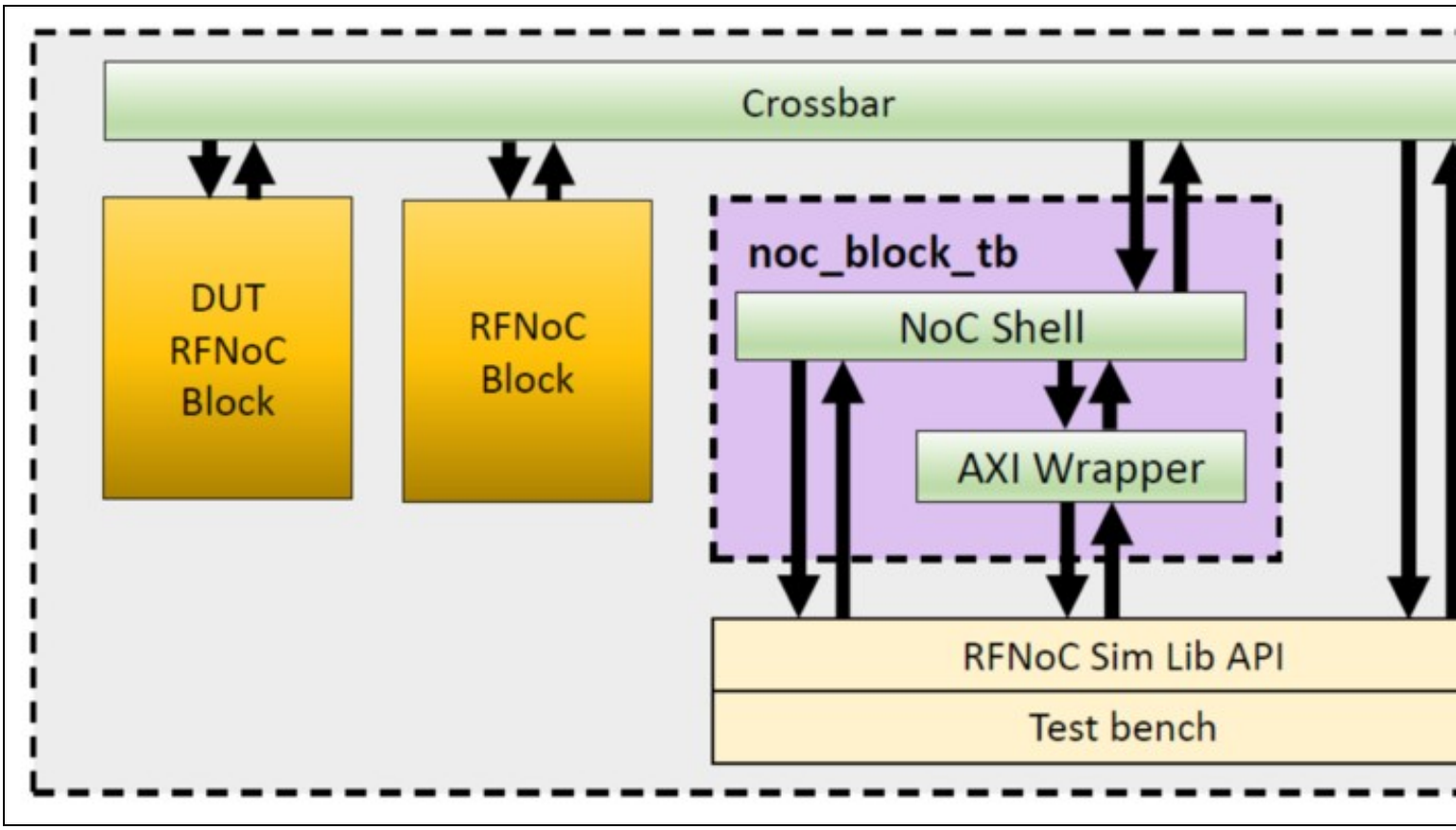
NOTE: In order to meet timing, FIFO blocks were added to either side of the Multiplication process.

25.8.2

In order to make the coding iteration process more efficient, it is recommended to create testbenches for all RFNoC blocks before compiling them into the FPGA image. This allows for flaw and/or bug detection early in the design. RFNoC Modtool provides the structure and files (e.g. noc_block_{USER_BLOCK_NAME}_tb) for the testbenches of each of the OOT blocks that are added with the `$ rfnocmodtool add` command.

Below is a figure that shows the general testbench architecture that is created by the RFNoC Modtool. This architecture allows a user to test their custom block in the exact same environment it will be placed in when it is built into the RFNoC architecture. Other benefits of the testbench architecture include:

- Testing through multiple blocks (e.g. FILTER -> FFT -> AVE)
- Testing with multiple streams (e.g. RFNoC block ADD/SUB takes 2 streams, one that will have a constant added to it and one that will have a constant subtracted from it)
- Data transfer abstraction (e.g. RFNoC Sim Lib API calls to `tb_streamer.send` and `tb_streamer.recv` which take care of all the AXI stream signaling)



NOTE: The `noc_block_tb` block is an instantiation of the `noc_block_export_io` that is used in testbenches to communicate to the RFNoC architecture. This makes it possible to talk ?RFNoC? to the user?s custom block and as such the custom block has a complete RFNoC experience (signaling, flowcontrol, addressing, etc)

From the Adding custom blocks to OOT Module section where the `gain` block was initially created, the last files generated were:

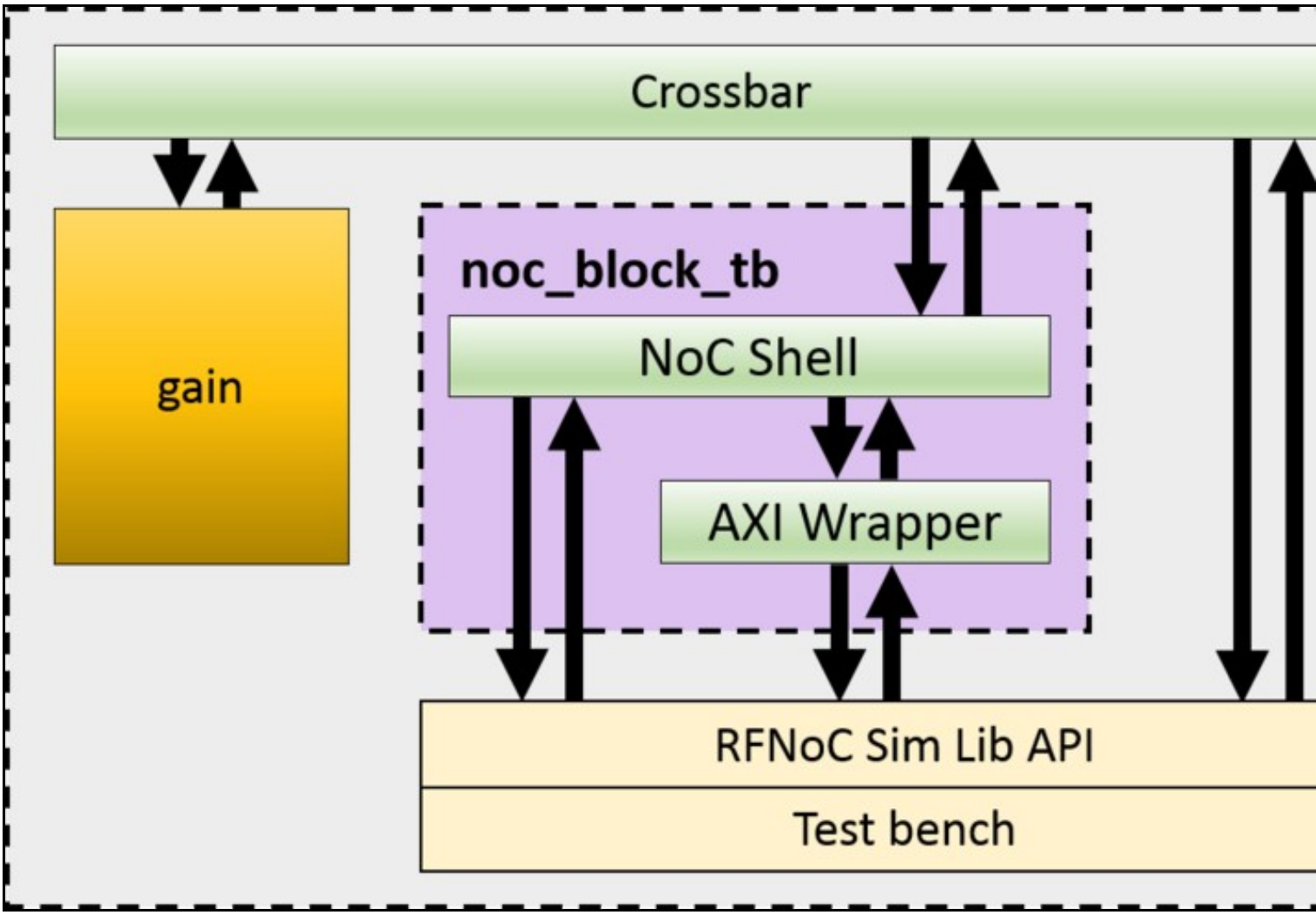
```
rfnoc/testbenches/noc_block_gain_tb folder created
Adding file 'rfnoc/testbenches/noc_block_gain_tb/noc_block_gain_tb.sv'...
Adding file 'rfnoc/testbenches/noc_block_gain_tb/Makefile'...
Adding file 'rfnoc/testbenches/noc_block_gain_tb/CMakeLists.txt'...
```

The `noc_block_gain_tb` is a folder generated to contain all the files related to the test bench of the `gain` block. Each time a new OOT block is created, a new folder will be generated as well.

Inside of this folder are the following three files:

- `CMakeLists.txt`: this is an empty file used, so far, only to increase the scope of the compilers.
- `noc_block_gain_tb.sv`: this is a *System Verilog* file, in which user custom tests are to be located. This is the **only** file that needs to be modified.
- `Makefile`: This file determines the directives that run the simulation.

The `noc_block_gain_tb.sv` testbench skeleton code creates the following architecture:



Open the file `rfnoc-tutorial/rfnoc/testbenches/noc_block_gain_tb/noc_block_gain_tb.sv` and modify the following lines:

Right under the ?Verification? section:

```
initial begin : tb_main
  string s;
  logic [31:0] random_word;
  logic [63:0] readback;
  logic [15:0] gain;
```

In the ?Test 4 -- Write / readback user registers? section:

```
`TEST_CASE_START("Write / readback user registers");
random_word = $random();
tb_streamer.write_user_reg(sid_noc_block_gain, noc_block_gain.SR_GAIN, random_word[15:0]);
tb_streamer.read_user_reg(sid_noc_block_gain, 0, readback);
$sformat(s, "User register 0 incorrect readback! Expected: %0d, Actual %0d", readback[15:0], random_word[15:0]);
`ASSERT_ERROR(readback[15:0] == random_word[15:0], s);
```

In the ?Test 5 -- Test sequence? section:

```
`TEST_CASE_START("Test sequence");
gain = 100;
tb_streamer.write_user_reg(sid_noc_block_gain, noc_block_gain.SR_GAIN, gain);
fork
  begin
    cvita_payload_t send_payload;
    for (int i = 0; i < SPP/2; i++) begin
      send_payload.push_back(64'(i));
    end
    tb_streamer.send(send_payload);
  end
  begin
    cvita_payload_t recv_payload;
    cvita_metadata_t md;
    logic [63:0] expected_value;
    tb_streamer.recv(recv_payload,md);
    for (int i = 0; i < SPP/2; i++) begin
      expected_value = i*gain;
```

Test #4 verifies that we can write and readback the `gain` value. Test #5 writes to the `gain` register, sends a sample set in the form of a ramp (1, 2, 3, 4, etc) to the RFNoC gain block and finally reads the values from the `gain` block and compares them to expected values. The followings steps will allow the user to run this testbench.

From within the `rfnoc-tutorial` directory, create a `build` directory and enter it by running:

```
$ mkdir build && cd build/
```

The next step is to run `cmake`. If PyBOMBS was used to create the development sandbox, `cmake` will automatically detect the location of the `fpga` repository. If PyBOMBS was not used, the user must provide the location of where the `fpga` repository is installed.

If PyBOMBS used, run:

```
$ cmake ../
```

If PyBOMBS not used, run:

```
$ cmake [-DUHD_FPGA_DIR=/PATH/TO/FPGA/REPOSITORY] ../
```

Final output from the `$ cmake ../` command:

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/widow/rfnoc/src/rfnoc-tutorial/build
```

The following command will modify the necessary files and set the correct path to the simulation tools. From now on, every time a new block is added, this command will be run automatically. Remember, only run the following command once for each OOT module (not RFNoC block, but OOT module) created:

```
$ make test_tb
Scanning dependencies of target test_tb
Built target test_tb
```

Testbenches can be executed by running the command:

```
$ make noc_block_[name_of_your_block]_tb
```

The gain block testbench can be run by running the following command:

```
$ make noc_block_gain_tb
```

The simulation will start. Final output should look like this:

```
=======================================================
TESTBENCH STARTED: noc_block_gain
=======================================================
[TEST CASE   1] (t=000000000) BEGIN: Wait for Reset...
[TEST CASE   1] (t=000001002) DONE... Passed
[TEST CASE   2] (t=000001002) BEGIN: Check NoC ID...
Read GAIN NOC ID: 1111222233334444
[TEST CASE   2] (t=000001238) DONE... Passed
[TEST CASE   3] (t=000001238) BEGIN: Connect RFNoC blocks...
Connecting noc_block_tb (SID: 1:0) to noc_block_gain (SID: 0:0)
Connecting noc_block_gain (SID: 0:0) to noc_block_tb (SID: 1:0)
[TEST CASE   3] (t=000005457) DONE... Passed
[TEST CASE   4] (t=000005457) BEGIN: Write / readback user registers...
[TEST CASE   4] (t=000006888) DONE... Passed
[TEST CASE   5] (t=000006888) BEGIN: Test sequence...
[TEST CASE   5] (t=000007633) DONE... Passed
=======================================================
TESTBENCH FINISHED: noc_block_gain
 - Time elapsed:   7700 ns
 - Tests Expected: 5
 - Tests Run:      5
 - Tests Passed:   5
Result: PASSED
=======================================================
$finish called at time : 7700 ns : File "/home/widow/rfnoc/src/rfnoc-tutorial/rfnoc/testbenches/noc_block_gain_tb/noc_block_gain_tb.sv" Li
INFO: [USF-XSim-96] XSim completed. Design snapshot 'noc_block_gain_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000000000us
launch_simulation: Time (s): cpu = 00:00:10 ; elapsed = 00:00:12 . Memory (MB): peak = 966.387 ; gain = 54.848 ; free physical = 3080 ; fr
# if [string equal $vivado_mode "batch"] {
#     puts "BUILDER: Closing project"
#     close_project
# } else {
#     puts "BUILDER: In GUI mode. Leaving project open."
# }
BUILDER: Closing project
****** Webtalk v2015.4 (64-bit)
  **** SW Build 1412921 on Wed Nov 18 09:44:32 MST 2015
  **** IP Build 1412160 on Tue Nov 17 13:47:24 MST 2015
    ** Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.

source /home/widow/rfnoc/src/rfnoc-tutorial/rfnoc/testbenches/noc_block_gain_tb/xsim_proj/xsim_proj.hw/webtalk/labtool_webtalk.tcl -notrac
INFO: [Common 17-206] Exiting Webtalk at Tue Jan 10 23:26:20 2017...
INFO: [Common 17-206] Exiting Vivado at Tue Jan 10 23:26:22 2017...
Built target noc_block_gain_tb
```

With every custom block created, a `make` directive will be available to run the simulation from the `build` directory.

25.8.3

In this section steps are given on how to initiate an FPGA build while incorporating the user?s custom RFNoC block. The first sections give general information on building RFNoC images. The remaining two sections show how to initiate FPGA builds using a command line interface and using a graphical interface (coming out soon), respectively.

25.8.3.1

There is a maximum number of blocks that can be added for each device. The maximum amount of computation engines (CEs/RFNoC blocks) that each device can use is 16, but the amount of custom blocks that can be added depends on the device.

If using a device from the X3xx series, from the 16 CEs, there are 6 that will be always added and are not subject to direct customization: 1 CE for the AXI bus, 1 CE for the Ethernet Interface, 2 Radios and 2 Dma FIFOS. Because of this, the application will only allow a number of 10 custom blocks on the X3xx series.

If using a device from the E3xx series, 2 CE engines are always added and are not subject to direct customization: 1 CE for the AXI bus and 1 Radio. This would virtually allow 14 slots for custom blocks. However, given the size of the FPGA on the E3xx series of devices, the application only allows a number of 6 custom blocks.

NOTE: Blocks with higher resource utilization may fill up the FPGA and force the user to include less blocks.

Verify the current maximum values by running the `uhd_images_builder.py` utility from the scripts directory.

```
$ cd {USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts
$ uhd_image_builder.py --help
```

25.8.3.2

RFNoC target names follow the pattern `{DEVICE}_RFNOC_{BUILD_TYPE}` with the following build types:

- `HG`: 1GigE on SFP+ Port0, 10Gig on SFP+ Port
- `XG`: 10GigE on both SFP+ ports
- `HLS`: Vivado High Level Synthesis enabled
- `sgX`: Speed grade for E300 devices (1 or 3)

Some examples are:

- `X310_RFNOC_HG`
- `X300_RFNOC_HG`
- `X310_RFNOC_XG`
- `X300_RFNOC_XG`
- `X310_RFNOC_HLS_HG`
- `X300_RFNOC_HLS_HG`
- `E310_RFNOC` (this is for the speed grade 1 FPGA version of E310, append `_sg3` for speed grade 3)

NOTE: E310, E312 and E313 all have the same FPGA hardware and therefore will use the `E310_RFNOC_{BUILD_TYPE}` target. USRP E3xx devices have either `sg1` or `sg3` hardware, please visit here to find out how to differentiate.

Additional information about the build targets can be found at the build instructions for USRP3, found here.

25.8.3.3

The script `uhd_image_builder.py` is used to generate the NoC block instantiation file and build the FPGA image. Run the help menu by typing:

```
$ cd {USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts
$ ./uhd_image_builder.py --help

usage: uhd_image_builder.py [-h] [-I INCLUDE_DIR [INCLUDE_DIR ...]]
                            [-m MAX_NUM_BLOCKS] [--fill-with-fifos]
                            [-o OUTFILE] [-d DEVICE] [-t TARGET] [-g] [-c]
                            [blocks [blocks ...]]

Generate the NoC block instantiation file

positional arguments:
  blocks                List block names to instantiate.

optional arguments:
  -h, --help            show this help message and exit
  -I INCLUDE_DIR [INCLUDE_DIR ...], --include-dir INCLUDE_DIR [INCLUDE_DIR ...]
                        Path directory of the RFNoC Out-of-Tree module
  -m MAX_NUM_BLOCKS, --max-num-blocks MAX_NUM_BLOCKS
                        Maximum number of blocks (Max. Allowed for x310|x300:
                        10, for e300: 6)
  --fill-with-fifos     If the number of blocks provided was smaller than the
                        max number, fill the rest with FIFOs
  -o OUTFILE, --outfile OUTFILE
                        Output /path/filename - By running this directive, you
                        won't build your IP
  -d DEVICE, --device DEVICE
                        Device to be programmed [x300, x310, e310]
  -t TARGET, --target TARGET
                        Build target - image type [X3X0_RFNOC_HG,
                        X3X0_RFNOC_XG, E310_RFNOC_sg3...]
  -g, --GUI             Open Vivado GUI during the FPGA building process
  -c, --clean-all       Cleans the IP before a new build
```

Here are details on the usage of the script which is followed by an example:

Blocks: The first arguments are the names of RFNoC blocks that the user wants to have compiled into the new image which are separated by a space. They can be custom blocks from the user?s OOT module or from the ones that are provided from Ettus, or a combination. Blocks provided by Ettus Research are listed (among other sources necessary for the FPGA build) in the `{USER_PREFIX}/src/uhd-fpga/usrp3/lib/rfnoc/Makefile.srcs` file.

These blocks can be identified by the following pattern:

```
noc_block_{NAME}.v
```

However, as all the RFNoC blocks have the same `noc_block_` prefix, for simplicity this prefix is omitted when listing the blocks in the `uhd_image_builder.py` utility. As an example of the incorrect and correct way of adding blocks, consider the following examples when adding the `noc_block_null_source_sink` and `noc_block_siggen` blocks:

Incorrect method:

```
$ ./uhd_image_builder.py noc_block_null_source_sink noc_block_siggen ...
```

Correct method:

```
$ ./uhd_image_builder.py null_source_sink siggen ...
```

NOTE: Blocks generated by the RFNoC Modtool follow the same naming convention.

There is an increasing list of pre-built blocks. Here is a sample:

- axi_fifo_loopback
- axi_dma_fifo
- fir_filter
- fft
- null_source_sink
- schmidl_cox
- packet_resizer
- split_stream
- vector_iir
- addsub
- window
- keep_one_in_n
- pfb
- export_io
- conv_encoder_qpsk
- siggen
- logpwr
- fosphor
- moving_avg
- ddc
- duc

RFNoC related blocks generally reside in `fpga/usrp3/lib/rfnoc/`.

Block	Filename	Description
FIFO	noc_block_axi_fifo_loopback.v	Simple FIFO loopback / passthrough block.
FFT	noc_block_fft.v	Xilinx coregen based Fast Fourier Transform up to length 4096.
FIR	noc_block_fir_filter.v	Xilinx coregen based Finite Impulse Response Filter, 41 taps, reconfigurable tap coefficients.
Window	noc_block_window.v	Windowing block for use with FFT block.
Vector IIR	noc_block_vector_iir.v	Single pole IIR with configurable coefficients that filters data along vectors (i.e. parallel streams of samples). Useful with FFT output.
Keep One in N	noc_block_keep_one_in_n.v	Keeps one packet every N packets.
AddSub	noc_block_addsub.v	Example of using multiple block ports in a single RFNoC block to add and subtract streams.
Null Source Sink	noc_block_null_source_sink.v	Generates dummy packets and can consume packets at a configurable rate. Useful for testing.
Packet Resizer	noc_block_packet_resizer.v	Resizes input packets to a configurable size (larger or smaller than source packets).
Split Stream	noc_block_split_stream.v	Replicates an input stream to a configurable number of output streams.

NOTE: There is a restriction on the amount of blocks that can added into the FPGA image, see the section in this Application Note labeled Discussion on number of blocks in an FPGA image for more information.

`-I INCLUDE_DIR`: The `-I` directive provides the path to top OOT directory, which contains the users `rfnoc/fpga-src` directory which contains the custom blocks. This path is needed by the Xilinx Vivado tool. Inside the `fpga-src` directory there is a file called `Makefile.srcs` that contains the path of the OOT module and a list of all the custom OOT blocks. This is an auto generated file, which is amended every time a new block is added to the OOT module. Manually modifying this file is not recommended. If there are multiple OOT modules with various custom blocks that reside in different directories the way to include them all is by separating the different paths by a space (e.g. `-I /first/OOT/path/ /second/OOT/path/`).

IMPORTANT: Please be sure to terminate the path of your OOT with the "/" character. Otherwise the path might not be recognized.

`-d DEVICE`: The `-d` directive directs the script on which USRP device the build is for. If no `?d` is included the default is `?d x310`. Generation-3 USRPs and above all support RFNoC.

`-t TARGET`: The `?t` directive directs the script on which type of image to build for the chosen device. With each USRP device there are several build options to choose from. Detailed information about the build targets can be found at the build instructions for USRP3, found here. If `-t` is not included, a default target will be chosen for the given device. For example, the default `X310_RFNOC_HG` target builds for the `?d x310` device. More details on targets can be found in the section of this Application Note labeled Discussion on FPGA image targets.

`-m MAX_NUM_BLOCKS`: The `?m` directive specifies the max number of RFNoC blocks to build on the FPGA image. An RFNoC image does not need to fill all available slots with RFNoC blocks.

`--fill-with-fifos`: The `--fill-with-fifos` directive will fill the empty RFNoC block slots with FIFOS. As an example, if a user indicates three RFNoC blocks by name and also specifies `?m 5` then the other two slots will be filed with FIFOs.

`-o OUTFILE`: With the `-o` directive, the RFNoC blocks instantiation file is generated and saved at the desired path with the given name for the user to inspect. The FPGA image will NOT build if this directive is provided. The purpose of the `uhd_image_builder.py` script is to auto generate an instantiation file and populate the source files needed for the Xilinx Vivado tool to build the FPGA image, however, it may be desirable to only see the effect of adding a custom OOT module in the `fpga/` directory, or for inspecting the instantiation file. When the directive is not provided the `rfnoc_ce_auto_inst_x3x0.v` file is overwritten and the FPGA image build process will start automatically (standard use).

`-g, --GUI`: Open Vivado GUI during the FPGA building process

`-c, --clean-all`: Cleans the IP before a new build

Here is how to create an X310 FPGA image incorporating the `gain` block that was created earlier in this Application Note:

```
$ cd {USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts
$ ./uhd_image_builder.py gain ddc fft -I {USER_PREFIX}/src/rfnoc-tutorial/ -d x310 -t X310_RFNOC_HG -m 6 --fill-with-fifos
```

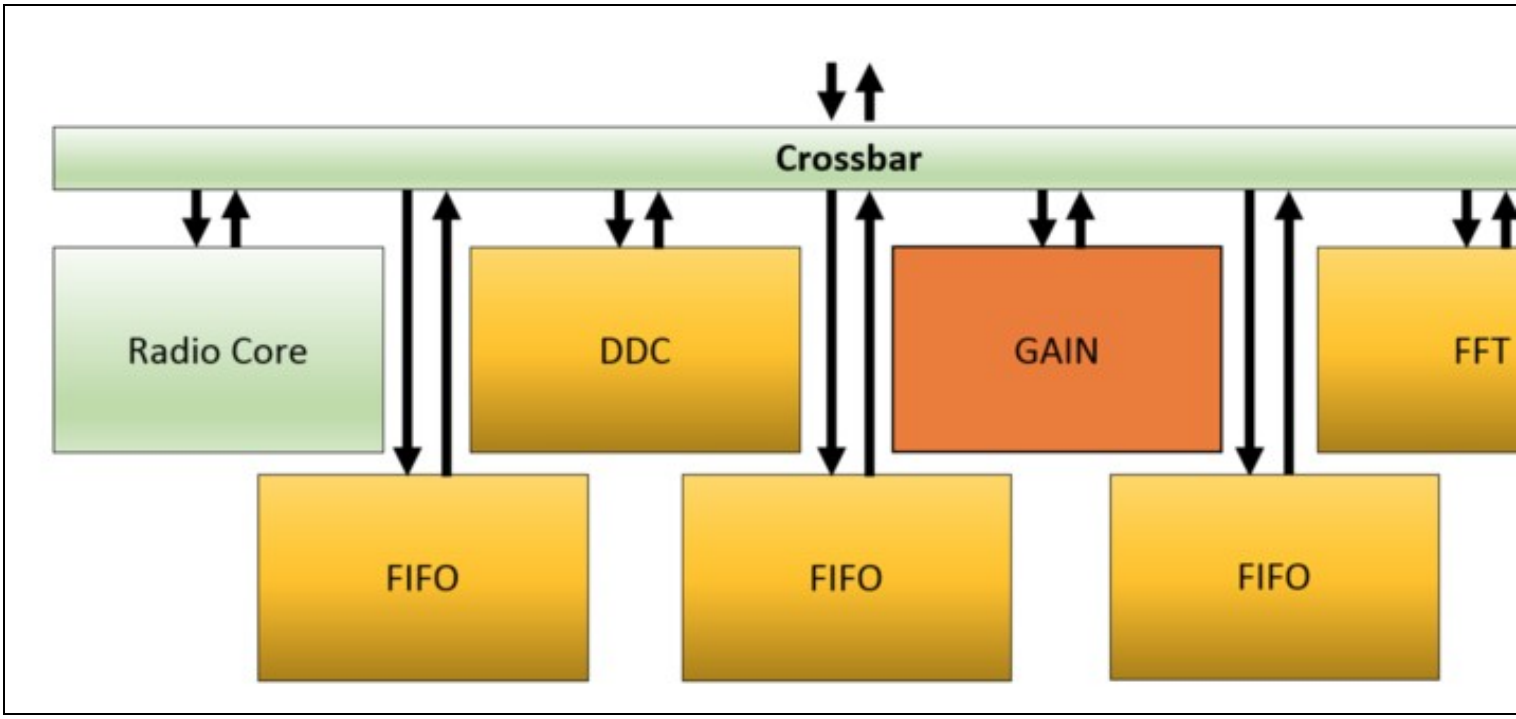
At the end of a successful compilation process, write the new image to a USRP. The following command will load the new image. Update the `{IP_Address}` of the USRP and `{USER_PREFIX}` to the appropriate values for your configuration before running the command.

```
$ uhd_image_loader --args "type=x300,addr={IP_ADDRESS}" --fpga-path {USER_PREFIX}/src/uhd-fpga/usrp3/top/x300/build/usrp_x310_fpga_RFNOC_H
```

NOTE:

- The FPGA image building process may take over an hour.
- FPGA images are specific to the USRP device NOT the USRP series. For example, a USRP X300 FPGA image will **NOT** work on a USRP X310 and vice versa. Loading an image that does not correspond to a USRP device will likely brick the device. Additional instructions on flashing a custom image to a device can be found here for X3xx series and here for E3xx series.
- [Environment setup] - The `uhd_image_builder.py` script will also set up the Xilinx Vivado environment by automatically running the `setupenv.sh` script located in the `{USER_PREFIX}/src/uhd-fpga/usrp3/top/{device}` directory. The `setupenv.sh` script assumes that Xilinx Vivado is installed in the default location of `/opt/Xilinx/Vivado`. If the installation is in a different directory the `setupenv_base.sh` script will need to be modified. The script is located at: `{USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts/setupenv_base.sh`

Besides the custom `gain` block, a `DDC` and `FFT` block are also being added along with three `FIFOs`. The `DDC`, `FFT` and `FIFO` blocks are already in the script's path and therefore do not need their path specified (they ship with the Ettus Research FPGA code). The reason three FIFOs are added is because the max number of blocks was specified to be 6 (`-m 6`) and since only 3 blocks were specifically named the other three slots are filled with FIFOs.



NOTE: Instructions on flashing the image to a device can be found here for X3xx series and here for E3xx series. FPGA images are specific to the USRP device **NOT** the USRP series. For example, a USRP X300 FPGA image will **NOT** work on a USRP X310 and vice versa. Loading an image that does not correspond to a USRP device will likely brick the device.

Once the newly compiled image is loaded onto a USRP X3xx running the following command will show what RFNoC blocks are available on the FPGA:

```
$ uhd_usrp_probe

|   |     _____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * Block_0
|   |   |   * DDC_0
|   |   |   * FFT_0
|   |   |   * FIFO_0
|   |   |   * FIFO_1
|   |   |   * FIFO_2
```

NOTE: The reason the custom block is called `Block_0` and not `gain_0` is because there is still host side software/files that need updated in order for this block to populate it?s proper name. A following section (UHD Integration) will step through the process of updating those host side files.

25.8.3.4

A graphical user interface for FPGA generation and building is shipped along with the `uhd_image_builder.py` script. This intuitive application aids in setting up a custom FPGA build.

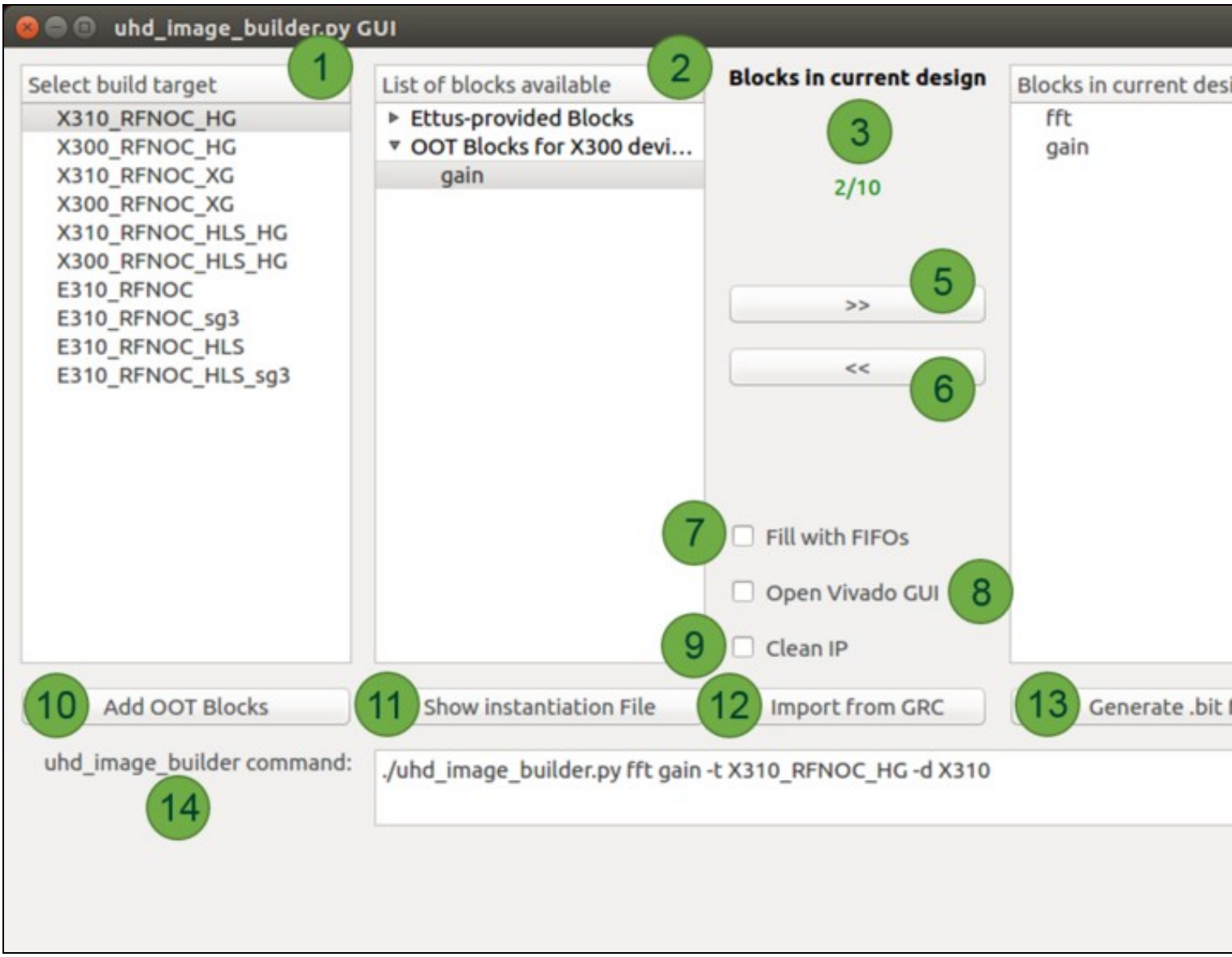
This utility is located in the same `scripts` directory as `uhd_image_builder.py`.

To run it, enter the following commands:

```
$ cd {USER_PREFIX}/src/uhd-fpga/usrp3/tools/scripts/
```

```
$ ./uhd_image_builder_gui
```

The application will then be launched:



1. Select build target: In this panel the available build targets are listed. This list may vary depending on which branch of the FPGA repository this user is using. Only RFNoC targets are listed. The build type descriptions are:

- `HG`: 1GigE on SFP+ Port0, 10Gig on SFP+ Port1
- `XG`: 10GigE on both SFP+ ports
- `HLS`: Vivado High Level Synthesis enabled
- `sgX`: Speed grade for E300 devices (1 or 3)

2. List of blocks available: In this panel the available blocks are listed that can be included into a custom design. This list separates the RFNoC blocks provided by Ettus Research and the OOT modules and corresponding blocks that the user adds. Given the hardware differences between the X3xx and E3xx devices, this list will dynamically change when a different device is selected from the panel on the left. This implies that it is necessary to add the OOT modules for each device independently. This is accomplished by using the `Add OOT Blocks` feature of the application, details of which are explained at #7 (`Add OOT Blocks`).

3. Blocks in current design: This section gives information on the MAX number of blocks for a given USRP (based on the target selection). There is a maximum number of blocks that can be added for each device. See the section in this App Note labeled "Discussion on number of blocks in an FPGA image" for more information.

4. Blocks in current design: This panel will be populated by adding elements from the available blocks. All the blocks listed in here will be compiled into the FPGA custom image. There is a maximum number of blocks that can be added for each device. See the section in this App Note labeled "Discussion on number of blocks in an FPGA image" for more information.

5. Add button (>>): Manually add the blocks from the central panel into your design.

6. Remove button (<<): Remove blocks from the current design (far-left panel)

7. Fill with FIFOs: By checking this box, the design will fill any available/unspecified block slots with FIFOs. The number of FIFO blocks that will be instantiated is based on the rules of amount of blocks explained at #3. When less than the max amount of blocks are needed for certain implementation, many users choose to fill their design with FIFO blocks.

8. Open Vivado GUI: Open Vivado GUI during the FPGA building process. This allows the user to save a Vivado project with all IP and work within the Vivado GUI for development.

9. Clean IP: Cleans the IP before a new build (recompiles all IP).

10. Add OOT blocks: Manually add RFNoC Modtool-generated OOT modules by pointing the application to the `Makefile.srcs` file, which is located in the `{USER_PREFIX}/src/{USER-OOT-moddir}/rfnoc/fpga-srcs/` directory. After adding this file, blocks will appear under ?`OOT blocks for XXXX devices`?

11. Show Instantiation File: The application auto-generates the instantiation file that is going to be used by Vivado to build the FPGA image. This instantiation file can be viewed and edited before starting the build by clicking this button.

12. Import from GRC: If the user has a GNU Radio flowgraph with RFNoC blocks already in it, this application can read what RFNoC blocks are in the flowgraph and populate the `Blocks in current design` section of the application with the necessary RFNoC blocks. **NOTE:** All RFNoC blocks pulled from a `.grc` file must be in the of `List of blocks available` before beginning the build.

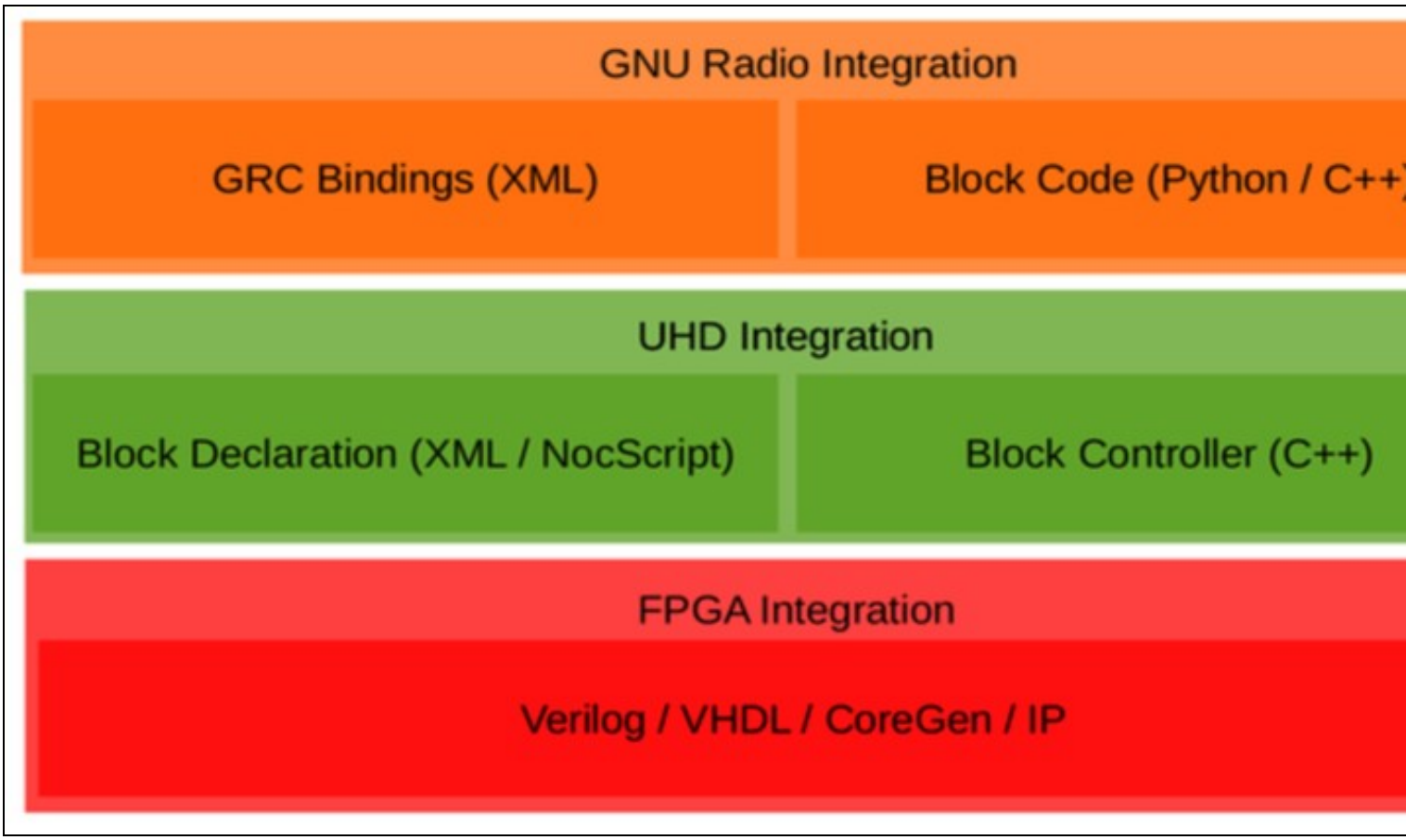
13. Generate .bit file: Start the build by clicking this button.

14. uhd_image_builder command: The command line command with arguments is dynamically build here as the user selects different options. The user could save this command to use next time they build/compile an FPGA image to avoid having to select all options again.

NOTE: See the latter end of the previous section for additional information on what to expect once the compile has started as well as final output.

25.9

Now that the FPGA portion is complete the next step is to add software integration to UHD and GNU Radio as depicted in the RFNoC Stack below.



25.9.1

Despite the data processing happening on the FPGA, the host software still has a lot of responsibilities in order for an RFNoC application to function. For example, it needs to know which settings registers are available within an RFNoC block, or what kind of input and output a block has. All of this information goes into the `Block Declaration`, which is an XML file that is readable by UHD. Often, some simple logic needs to be embedded in the XML file, which we can do by using a simple scripting language called Noc-Script. Changes to the block declaration file are immediately imported into UHD every time an application is executed, and therefore, no software development toolchain needs to be set up.

The list of things declared by the block declaration file includes:

- Block name and Noc-ID
- Registers
- Inputs and outputs (including types)

In some cases, additional C++ code is required to properly control a block from software. In this case, a `Block Controller` file is required as well as the declaration file. In most cases, the default block controller provided by UHD is sufficient, so no C++ code needs to be written. Writing custom block controllers requires more effort, and means having to set up a programming toolchain. A common reason to write custom C++ block controllers is if setting a register requires a lot of computation, which is not feasible to do within a block declaration file (e.g., using Noc-Script).

Skeleton code for both the block declaration and the block controller (if required) can be generated through RFNoC Modtool.

Because the `gain` block does not require anything other than simply reading and writing to a single register the default block controller will suffice for this example. However, we will need to add information about the register.

Open the `gain.xml` file located in the `/rfnoc-tutorial/rfnoc/blocks` directory and add the following:

```
<?xml version="1.0"?>

<nocblock>
  <name>gain</name>
  <blockname>gain</blockname>
  <ids>
    <id revision="0">1111222233334444</id>
  </ids>

  <registers>
    <setreg>
      <name>GAIN</name>
      <address>128</address>
    </setreg>
  </registers>

  <args>
    <arg>
      <name>gain</name>
      <type>double</type>
      <value>1.0</value>
      <check>GE($gain, 0.0) AND LE($gain, 32767.0)</check>
      <check_message>Invalid gain.</check_message>
      <action>
        SR_WRITE("GAIN", IROUND($gain))
      </action>
    </arg>
  </args>

  <ports>
    <sink>
      <name>in0</name>
      <type>sc16</type>
    </sink>
    <source>
      <name>out0</name>
      <type>sc16</type>
    </source>
  </ports>
</nocblock>
```

25.9.2

GNU Radio is built around the concept of blocks, similarly to RFNoC. When mapping RFNoC into an application, the simple constraint is made that every RFNoC block maps to a single GNU Radio block. Thus, when creating mixed GNU Radio/RFNoC applications, there is a very clear 1:1 mapping between what?s happening in RFNoC and GNU Radio.

Since most RFNoC blocks behave very similar to one another from GNU Radio?s perspective, it is generally not required to write C++ code for another block. Rather, a default block provided by RFNoC can be used with appropriate configuration. However, in some cases it may be desirable or even necessary to write a custom GNU Radio block for more specific controlling of the underlying RFNoC block. GNU Radio allows writing blocks in either C++ or Python, but since UHD and RFNoC do not have a Python API, a custom wrapper for an RFNoC block needs to be written in C++. RFNoC Modtool will create skeleton files for this purpose.

The most popular and effective way to use GNU Radio is through the graphical interface, the GNU Radio Companion (GRC). GRC requires a separate description of every GNU Radio block in order to become available in the graphical UI, and the same is true for an RFNoC block that is wrapped in a GNU Radio block (even if the generic RFNoC block wrapper is used). For GNU Radio 3.7 and earlier, GRC bindings for blocks are written as XML files with interspersed Cheetah or Python statements. For a more detailed tutorial on how to write these files, refer to the GNU Radio Documentation and associated tutorials.

25.9.2.1

- C++ or Python, although RFNoC blocks need to be written in C++ (if at all)
- How does GNU Radio interface to RFNoC?
    - via C++ infrastructure code in `gr-ettus`
    - `gr-ettus` provides a base RFNoC block class
    - Users extend base class for their RFNoC blocks
    - Many blocks can use base class ?as is?
    - No C++ or Python code!
- `rfnoc-tutorial/lib/gain_impl.cc`
    - The gain block does not need anything additional

25.9.2.2

- XML
- Describes GNU Radio blocks to GRC
- No recompilation
- Requirement of GNU Radio Companion
- Not strictly necessary for GNU Radio
- Tutorial on how to write them:
    - http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion
- Skeleton file generated by RFNoC Modtool

Open the `tutorial-gain.xml` file located in the `rfnoc-tutorial/grc` directory and edit as follows:

```
<?xml version="1.0"?>
<block>
  <name>RFNoC: gain</name>
  <key>tutorial_gain</key>
  <category>tutorial</category>
  <import>import tutorial</import>
  <make>tutorial.gain(
    self.device3,
    uhd.stream_args( \# TX Stream Args
        cpu_format="fc32",
        otw_format="sc16",
        args="gr_vlen={0},{1}".format(${grvlen}, "" if $grvlen == 1 else "spp={0}".format($grvlen)),
    ),
    uhd.stream_args( \# RX Stream Args
        cpu_format="fc32",
        otw_format="sc16",
        args="gr_vlen={0},{1}".format(${grvlen}, "" if $grvlen == 1 else "spp={0}".format($grvlen)),
    ),
    $block_index, $device_index,
  )
self.$(id).set_arg("gain", $gain)
  </make>
  <callback>set_arg("gain", $gain)</callback>


     .
     .
     .

    <option>
      <name>Byte</name>
      <key>u8</key>
    </option>
  </param>
  <param>
    <name>Gain</name>
    <key>gain</key>
    <value>1.0</value>
    <type>real</type>
  </param>


  <sink>
    <name>in</name>
    <type>complex</type>
    <vlen>$grvlen</vlen>
    <domain>rfnoc</domain>
  </sink>


  <source>
    <name>out</name>
    <type>complex</type>
    <vlen>$grvlen</vlen>
    <domain>rfnoc</domain>
  </source>
</block>
```

NOTE: Indentation spacing is important in the `<make>` section.

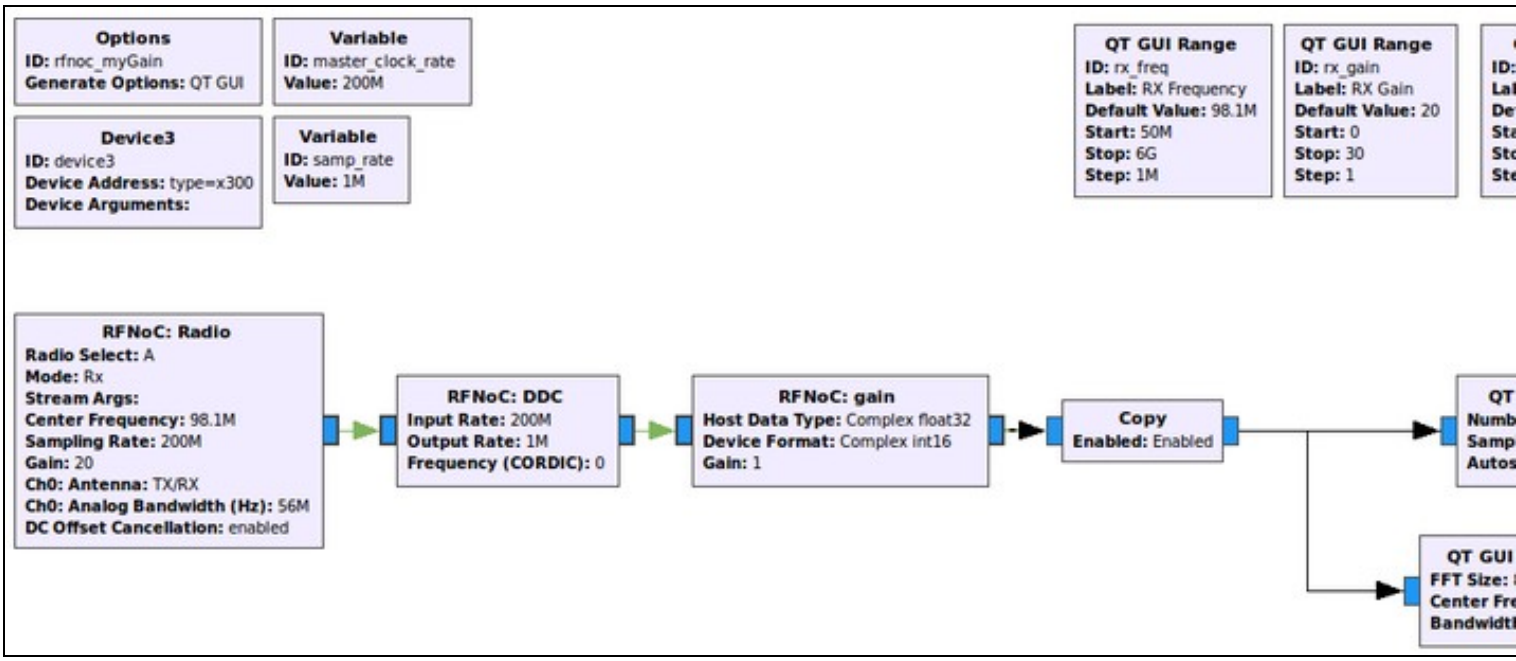
25.9.3

```
$ cd {USER_PREFIX}/src/rfnoc-tutorial/build
$ make install

$ uhd_usrp_probe

|   |     ____________________________________________________
|   |    /
|   |   |       RFNoC blocks on this device:
|   |   |
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * gain_0
|   |   |   * DDC_0
|   |   |   * FFT_0
|   |   |   * FIFO_0
|   |   |   * FIFO_1
|   |   |   * FIFO_2
```

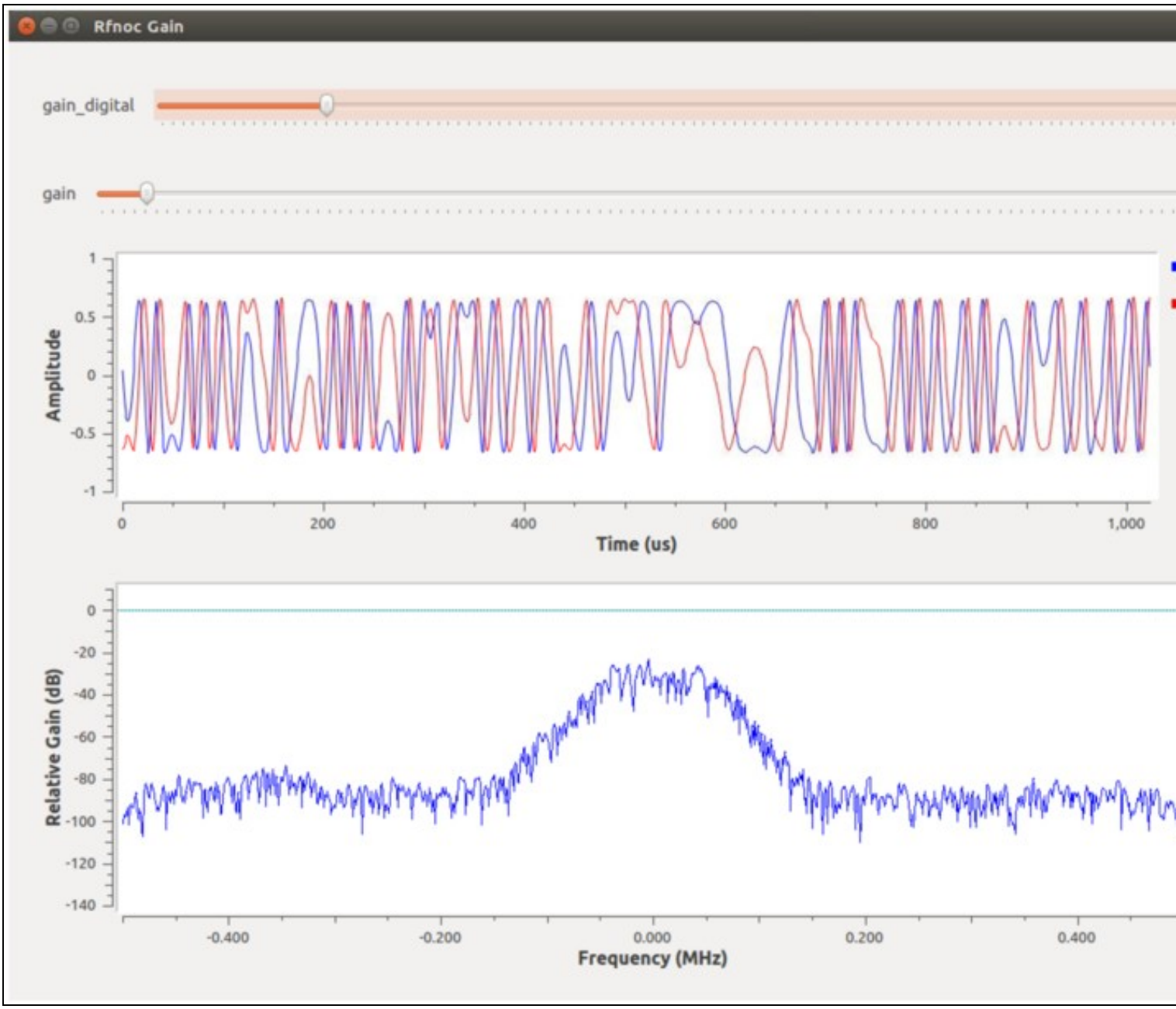
NOTE: In the case where the `gain_0` does not appear but `Block_0` does: Most likely, the XML block declaration file (see UHD Integration section) for the block contains a NoC-ID that does not match with any NoC-ID defined in the hardware part of the design. The user has to be certain that the description files are up-to-date and that the NoC-ID matches in the SW and HW side. See the UHD Integration section to update those host side files.

25.10

At this point the custom `gain` RFNoc Block (Computation Engine) can be used within a GNU Radio flowgraph. Below is an example GRC flowgraph using our new block as well as the output application it produces.



NOTE: Copy Block: In the RFNoC domain, streams of data can not be split as easily as they are in the GNU Radio domain. The ?copy? block depicted in the screenshot above serves the function as a stream splitter . It?s main purpose, when ?enabled?, is to copy the samples it is getting at its input and putting then into the output, but here it is also serving as a boundary between a RFNoC-domain and a GNURadio-domain. In the flowgraph above. after this boundary is passed, the data stream can easily be split into the two sinks to have them run simultaneously (standard GNU Radio functionality). It is possible to connect the GNU Radio blocks directly to RFNoC blocks without a ?copy? block, but only one would work at a time (the other ones would have to be disabled). Another way to split data streams from the RFNoC-domain is to use the ?RFNoC: split stream? block, which would split the streams in the RFNoC domain, but this is not very useful here as we are, in any case, moving into the GNURadio-domain.



25.11

25.11.1

25.11.1.1

25.11.1.1.1

Verify all the correct Xilinx prerequisite software is installed.

Additional helpful information can be found in the following Xilinx forum posts:

- https://forums.xilinx.com/t5/Synthesis/Synthesis-failed-without-reporting-any-error/td-p/686000
- https://forums.xilinx.com/t5/Installation-and-Licensing/Vivado-on-Linux-synthesis-fails-with-no-error-message/td-p/732143

25.11.1.2

The `uhd_image_builder.py` script will also set up the Xilinx Vivado environment by automatically running the `setupenv.sh` located in the `{USER_PREFIX}/src/uhd-fpga/usrp3/top/{device}` directory. The `setupenv.sh` script assumes that Xilinx Vivado is installed in the default location of `/opt/Xilinx/Vivado`. If the installation is in a different directory, then the `setupenv_base.sh` script will need to be modified. The script is located at: `{USER_PREFIX}/src/uhd-fpga/usrp3_rfnoc/tools/scripts/setupenv_base.sh`.

25.12

The following reference files are included within the gain_src.tar.gz archive linked below:

- gain.xml
- noc_block_gain.v
- noc_block_gain_tb.sv
- tutorial_gain.xml
- rfnoc_gain.grc

Media:gain src.tar.gz

25.13

25.13.1

- Video: RFNoC Getting Started Video Tutorial
- USRP Mailing List
- RFNoC Software Resources Page
- RFNoC Introduction
- RFNoC Deep Dive: FPGA
- RFNoC Deep Dive: Host side
- Video: RFNoC presented at Wireless @ Virginia Tech, 2015
    - Explaining the slides of Intro, FPGA and Host presentations above (in that order).
- Video: It's the RFNoC Life for Us by Martin Braun at GRCon16, 2016

25.13.2

- GNU Radio OutOfTree Modules tutorial
- GNU Radio Installation
- GNU Radio Tutorials

25.13.3

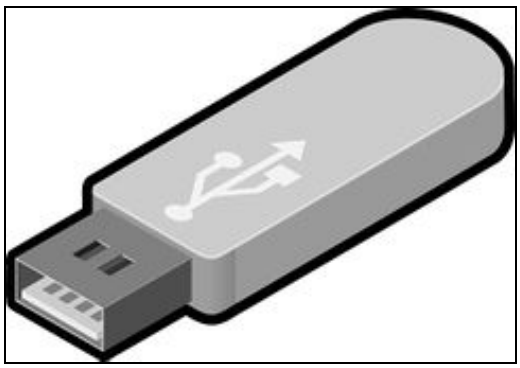
- USRP Mailing List
- UHD Software Resources Page
- USRP3 build instructions
- UHD Manual

25.13.4

- Xilinx - AXI reference guide
- UHD + GNU Radio Application Note (Linux)
- PyBOMBS

26 Live SDR Environment Getting Started Guides

The next major release of the Ettus Research Live SDR Environment, version 4.0, is scheduled for June 2016.



The Live SDR Environment is a bootable ISO image file that can be put onto a USB flash drive or a DVD, and runs directly from the USB or DVD, without having to install anything, and without modifying the local disk of the system. It contains a number of pre-installed applications as well as up-to-date versions of GNU Radio and UHD enabling support for the latest USRP SDR models. It is free and uses entirely open-source software, and is based on Linux Ubuntu 14.04.

The current version of the Ettus Research Live SDR Environment, version 3.0, contains older versions of UHD and GNU Radio, and does not support several recently released USRP SDR devices. Therefore, we recommend using the GNU Radio Live SDR Environment, which is maintained by the GNU Radio project. It is similarly configured to the Ettus Research Live SDR Environment and contains current versions of UHD and GNU Radio, but it is less USRP focused.

The current GNU Radio Live SDR Environment may be obtained from the GNU Radio Project at the link below:

https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioLiveDVD

A guide to Using the GNU Radio Live SDR Environment is available at:
http://gnuradio.org/blog/using-gnu-radio-live-sdr-environment/

Both Live SDR Environments may be put onto any standard DVD, or onto a USB 3.0 flash drive with at least 16 GB capacity. A USB 2.0 drive may also be used, but the system will take longer to boot, load and run programs, and respond to user interaction.

27 OctoClock CDA-2990 Getting Started Guides

27.1

- OctoClock
- Power Supply (6 Volt)

27.2

All Ettus Research products are individually tested before shipment. The USRP? is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP? can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

27.3

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

27.4

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

27.5

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

27.6

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

28 Using Ethernet-Based Synchronization on the USRP? N3xx Devices

28.1

AN-158 by Dan Baker, Wan Liu, and Michael Dickens

28.2

The USRP N3xx product family supports three different methods of baseband synchronization: external clock and time reference, GPSDO module, and Ethernet-based timing protocol. Using an external clock and time reference source, such as the CDA-2990 accessory, offers a precise and convenient method of baseband synchronization for high channel count systems where devices are located near each other, such as in a rackmount configuration. Using the GPSDO module enables synchronization when the devices are physically separated by large distances such as in small cell, RF sensor, TDOA, and distributed testbed applications. However, the GPSDO method typically has more skew than the other two methods and requires line of sight to satellites. Therefore, indoor, urban, or hostile environments restrict the use of GPSDO. Ethernet-based synchronization enables precise baseband synchronization over large distances in GPS-denied environments. However, this method consumes one of the SFP+ ports of the USRP N3xx devices and therefore reduces the number of connectors available for IQ streaming. This application note provides instructions for synchronizing multiple USRP N3xx devices using the Ethernet-based method.

28.3

The USRP N3xx product family supports Ethernet-based synchronization using an open source protocol known as White Rabbit. White Rabbit is a fully deterministic Ethernet-based network protocol for general purpose data transfer and synchronization[1]. This project is supported by a collaboration of academic and industry experts such as CERN and GSI Helmholtz Centre for Heavy Ion Research.

White Rabbit is an extension of the IEEE 1588 Precision Time Protocol (PTP) standard, which distributes time references over Ethernet networks. In addition, White Rabbit uses Synchronous Ethernet (SyncE) to distribute a common clock reference over the network across the Ethernet physical layer to ensure frequency syntonization between all nodes. This combination of SyncE and PTP, in addition to further measurements, provides sub-nanosecond synchronization over distances of up to 10 km. The White Rabbit extension of the IEEE 1588-2008 standard is in the final stages of becoming generalized as the IEEE 1588 High Accuracy profile[2].

The USRP N3xx product family implements the White Rabbit protocol using a combination of the FPGA and dedicated clocking resources. The USRP N3xx operates as a slave node, a White Rabbit master node is required in the network. Seven Solutions provides White Rabbit hardware that works with the USRP N3xx devices to create synchronous clock and time references that are precisely aligned across all devices in the network. See the ?Required Accessories? section for details on the required external hardware. The USRP N3xx devices do not support IQ sample streaming over this protocol. Therefore, only one of SFP+ ports is available for streaming when using White Rabbit synchronization.

For more information on the White Rabbit project, visit the links below:

White Rabbit documentation:

- https://www.ohwr.org/projects/white-rabbit/wiki

Standardization as IEEE1588 High Accuracy:

- https://www.ohwr.org/projects/wr-std/wiki

28.4

White Rabbit synchronization utilizes specific optical SFP transceivers and single mode fiber optic cables to achieve precise time alignment, as documented on the project website. The USRP N3xx was tested to work as a White Rabbit slave using the AXGE-1254-0531 SFP transceiver **marked in blue**, the AXGE-3454-0531 SFP transceiver **marked in purple**, and a G652 type single mode fiber optic cable.

Seven Solutions is a provider of White Rabbit equipment, including the WR-LEN and the White Rabbit Switch (WRS). The USRP N3xx was tested to work with both the WR-LEN and the WRS products. All accessories required for White Rabbit operation can be purchased directly from the Seven Solutions website. The AXGE SFP transceivers and fiber optic cables are only listed on the website as part of the ?KIT WR-LEN? product, but they can also be purchased individually by contacting Seven Solutions.

For more information on White Rabbit accessories, visit the links below:

White Rabbit SFP wiki:

- https://www.ohwr.org/projects/white-rabbit/wiki/SFP

Seven Solutions WR-LEN:

- http://sevensols.com/index.php/products/wr-len/

Seven Solutions KIT WR-LEN:

- http://sevensols.com/index.php/products/kit-wr-len/

Seven Solutions WRS:

- http://sevensols.com/index.php/products/white-rabbit-switch/

NOTE: There are transceivers other than the AXGE-1254-0531 and AXGE-3454-0531 that should be compatible with WR-LEN and WRS. The noted transceivers are known compatible with pre-existing calibration information in the WRS and WR-LEN. Calibrating any transceivers is beyond the scope of this document.

28.5

The White Rabbit feature of the USRP N3xx product family is based on standard networking technology, therefore many system topologies are possible. However, the USRP N3xx device only works as a downstream slave node and must receive its synchronization reference from an upstream master node. This section shows examples of typical configurations used to synchronize a network of multiple USRP N3xx devices.

Figure 1 shows a WRS operating as the master node connected to several USRP N3xx devices. Note that a master SFP port requires the purple SFP transceiver mentioned in the previous section, and a slave SFP port requires the blue SFP transceiver. The USRP N3xx use the SFP+ 0 port for White

Rabbit and SFP+ 1 port for IQ streaming. This port configuration requires the White Rabbit ?WX? FPGA bitfile.

Download all FPGA images for the version of the USRP Hardware Driver (UHD) installed on the host PC by running the following command in a terminal:

```
uhd_images_downloader
```

Load the WX bitfile by running:

```
uhd_image_loader --args type=n3xx,addr=ni-n3xx-<DEVICE_SERIAL> --fpga-path=?<UHD_INSTALL_DIRECTORY>/share/uhd/images/usrp_n310_fpga_WX.bit
```

Using the UHD API, configure the USRP application to use ?internal? clock source and ?sfp0? time source:

```
usrp->set_clock_source("internal")
usrp->set_time_source("sfp0")
```

The White Rabbit IP running on the FPGA disciplines the internal VCXO of the USRP N3xx to the clock reference from the upstream master node in the network. See the USRP N3xx block diagram for reference.

The WRS/WR-LEN device needs to be configured as a master on the ports connected to the USRP N3xx modules. Users can make this configuration with the WR-GUI application provided by Seven Solutions, or with a serial console connection to the WRS/WR-LEN device. See the WRS/WR-LEN manual for detailed instructions. After White Rabbit lock is achieved, the standard USRP N3xx synchronization process completes and the devices are ready for use.

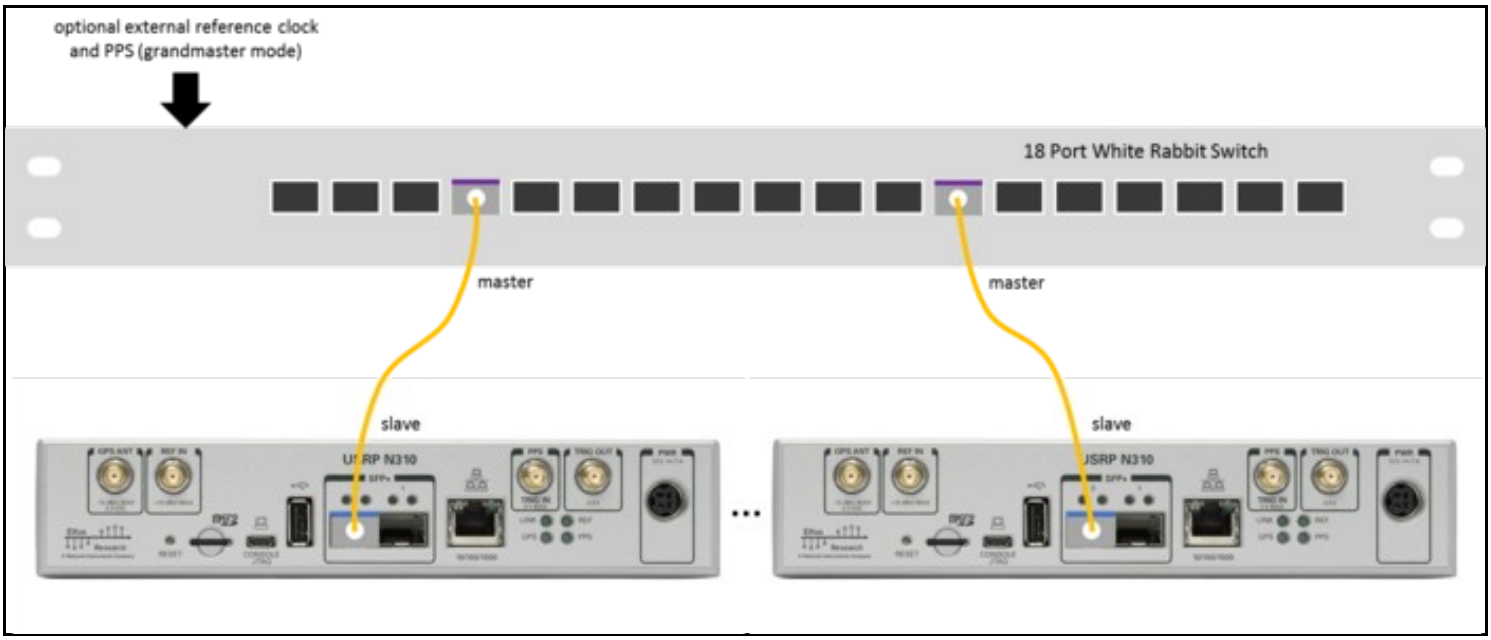


Figure 1. USRP N3xx synchronization using a White Rabbit Switch

In addition to operating as a master, the WRS and WR-LEN devices can operate as a grandmaster by receiving clock and time references from an external source. This feature is useful for situations where the entire White Rabbit network needs to be disciplined to GPS or other high accuracy synchronization equipment such as a rubidium source. See the WRS/WR-LEN documentation for more information on grandmaster mode.

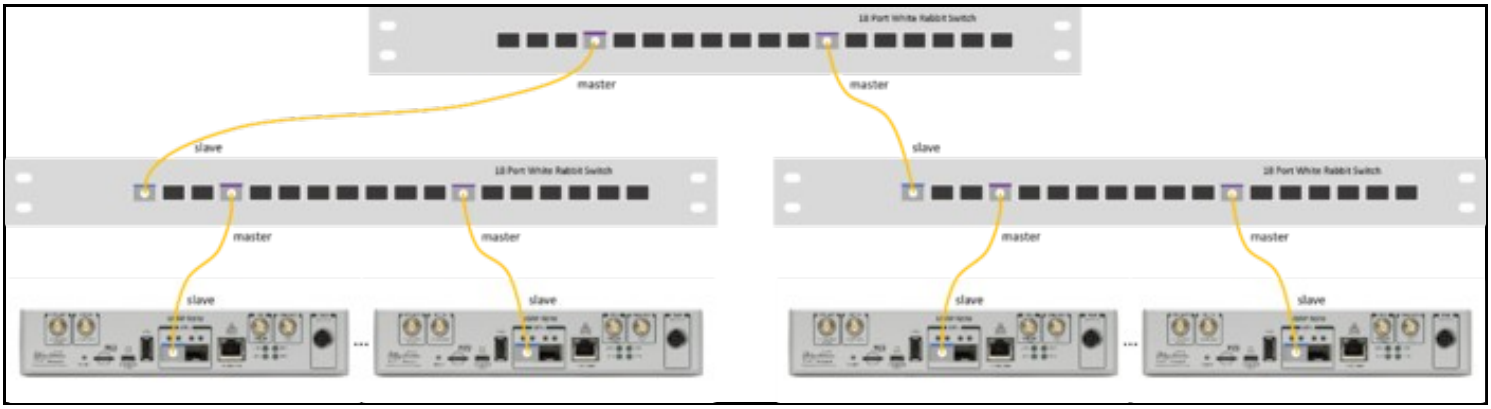


Figure 2 shows multiple WRS devices cascaded to expand the network beyond 18 USRP N3xx devices.

28.6

This section provides an example measurement of the timing alignment between multiple USRP N3xx devices synchronized using White Rabbit, with varying fiber cable lengths. As shown in Figure 3, a White Rabbit Switch in master mode is connected to one USRP N3xx device using a 5 km spool of fiber, and to another USRP N3xx device using 1 m of fiber. The synchronization performance was measured by probing the exported PPS signal, which is in the sample clock domain on both USRP N3xx devices thereby demonstrating sample clock and timestamp alignment. The time difference between each PPS edge was measured with an oscilloscope at room temperature in a laboratory environment. As shown in Figure 4, the resulting measurement shows about 222 ps of skew between the two USRP N3xx devices, thereby demonstrating the sub-nanosecond synchronization of White Rabbit over long distances.

The frequency accuracy of the internal oscillator of each USRP N3xx slave node is derived from the frequency accuracy of the upstream master node, in a manner similar to disciplining to an external clock reference source connected to the REF IN port. By connecting a high accuracy frequency source such as a rubidium reference to the master White Rabbit device in grandmaster mode, all USRP N3xx devices in the White Rabbit network would inherit this frequency accuracy.

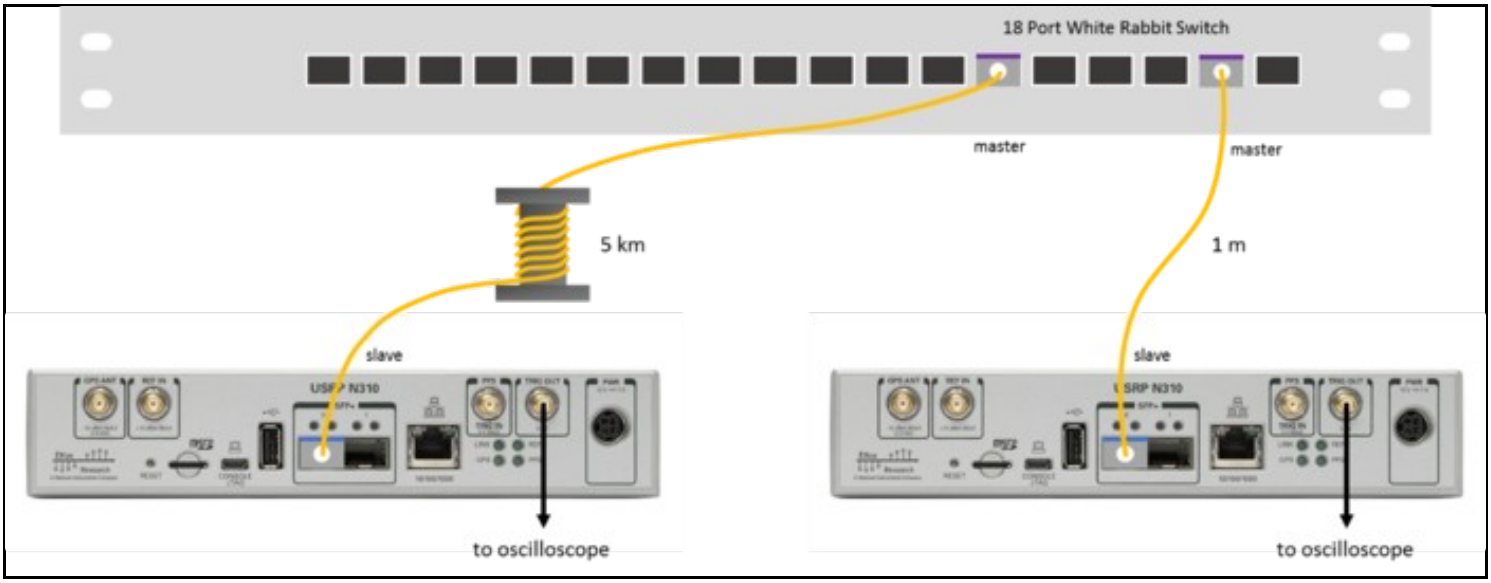


Figure 3. Test setup to measure the alignment of two USRP N3xx devices separated by 5 km of fiber

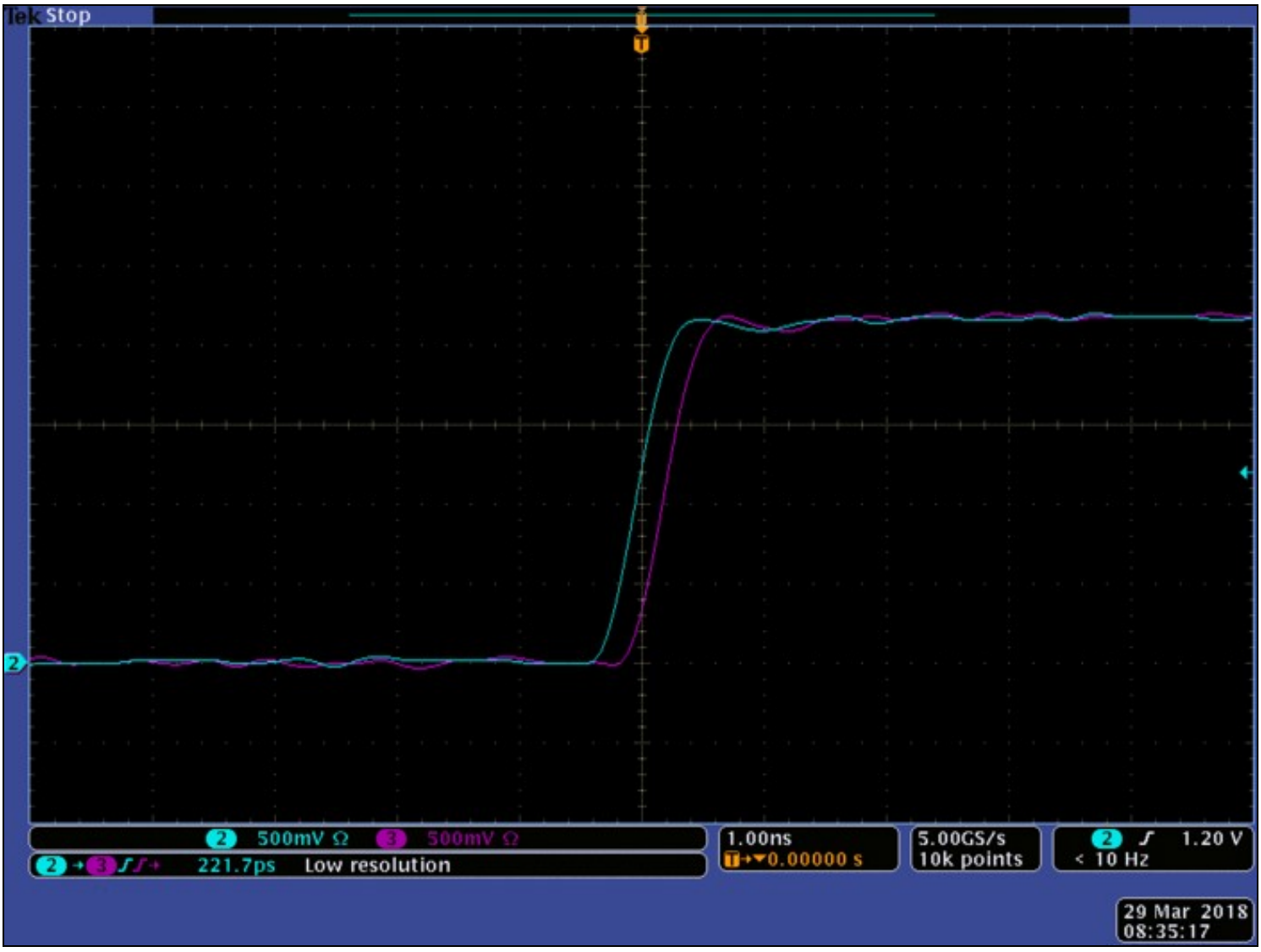


Figure 4. Example showing USRP N3xx measured skew of about 222 ps using 5 km of fiber

28.7

There are 3 primary ways to check the status of White Rabbit connectivity on the USRP.

28.7.1

After logging into the USRP and while one of the White Rabbit FPGA images is loaded, execute the following command:

```
n3xx_bist --skip-fpga-reload -v whiterabbit
```

If the USRP has White Rabbit lock, then it will return the following after around 5 seconds of command issue:

```
Executing test method: bist_whiterabbit
{
    "whiterabbit": {
         "error_msg": "",
         "lock_status": 1,
         "status": true
    }
}
BIST complete.
```

If there is no White Rabbit lock, then the test will run for roughly 45 seconds and return:

```
Executing test method: bist_whiterabbit
{
    "whiterabbit": {
         "error_msg": "",
         "lock_status": 0,
         "status": false
    }
}
BIST complete.
```

28.7.2

All White Rabbit devices provide a core that can be logged into for the purposes of viewing information and setting calibration parameters (among other things). In UHD, the White Rabbit core is integrated into the FPGA image, and is made available from the USRP's login as a Linux TTY device. To access the White Rabbit core, from the USRP's login issue the following command:

```
screen /dev/ttyUL0 115200
```

and then hit the `return` key to get the `wrc#` prompt. Issue the command `help` for a list of command available within the White Rabbit core. For example, the `gui` command will show the status of the device via refreshing text-based "window". When the USRP has White Rabbit lock, the window will show something like the following

```
WR PTP Core Sync Monitor wrpc-v4.2-2-g97f418e
Esc = exit

TAI Time:                  Thu, Feb 5, 1970, 00:30:52

Link status:
wru1: Link up   (RX: 4350, TX: 2475) IPv4: BOOTP running
Mode: WR Slave   Locked Calibrated

PTP status: slave

Synchronization status:
Servo state:               TRACK_PHASE
Phase tracking:            ON
Aux clock 0 status:        enabled

Timing parameters:
Round-trip time (mu):                907377 ps
Master-slave delay:                  452548 ps
Master PHY delays:           TX:    224037 ps, RX:    227977 ps
Slave PHY delays:            TX:         0 ps, RX:      1600 ps
Total link asymmetry:                  2281 ps
Cable rtt delay:                     453763 ps
Clock offset:                             1 ps
Phase setpoint:                        3848 ps
Skew:                                     1 ps
Update counter:                         737
```

whereas if there is no White Rabbit lock, the window will show something like

```
WR PTP Core Sync Monitor wrpc-v4.2-2-g97f418e
Esc = exit

TAI Time:                  Thu, Jan 1, 1970, 00:12:39

Link status:
wru1: Link up   (RX: 760, TX: 1162) IPv4: BOOTP running
Mode: WR Off

PTP status: listening

Sync info not valid
```

NOTE: Viewing information to see the status of the White Rabbit core is recommended when debugging White Rabbit connectivity. Changing any parameters within the White Rabbit FPGA core is an advanced topic that is beyond the scope of this document, and done at your own risk.

28.7.3

Assuming that the prior 2 lock tests work, then this test shows that UHD is working with White Rabbit signals by executing actual data transfer via the `benchmark_rate` utility. Note that this utility does not verify data integrity nor data phase any synchronization.

For 2 USRPs, here is an example `benchmark_rate` command showing `addr#` for both data and management, as well as basic settings for White Rabbit:

```
/usr/local/lib/uhd/examples/benchmark_rate --args type=n3xx,mgmt_addr0=10.9.8.202,addr0=10.9.10.2,mgmt_addr1=10.9.8.203,addr1=10.9.20.2,ma
```

If there is no White Rabbit lock, then this command will error out showing something like the following:

```
[ERROR] [RPC] Failed to lock SFP timebase.
[ERROR] [MPM.PeriphManager] sfp0 timebase failed to lock within 40 seconds. Status: 0x0
[ERROR] [MPM.RPCServer] init() failed with error: Failed to lock SFP timebase.
Error: RuntimeError: Error during RPC call to `init'. Error message: Failed to lock SFP timebase.
```

If there is White Rabbit lock, then if this command fails any issue should not be related to White Rabbit -- most likely networking issues.

28.8

- [1] https://www.ohwr.org/projects/white-rabbit
- [2] https://www.ohwr.org/projects/wr-std/wiki

29 Getting Started with DPDK and UHD

29.1

AN-500 by Nate Temple, Alex Williams, Wade Fife, Matt Prost, and Michael Dickens

29.2

This application note walks through the process to get started with the Data Plane Development Kit (DPDK) driver within UHD.

29.3

Up until now, UHD's only support for networked devices was backed by the kernel's sockets implementation. Every call to `send()` or `recv()` would cause a context switch and invite the kernel's scheduler to replace our thread with something else. Because the typical scheduler is optimized to distribute CPU time fairly across multiple loads, the timing-critical threads might sporadically be hit with sleeping time, and the thread might be migrated off its current CPU and forced to run on another. The overhead and random latency spikes make it difficult to enable reliable real-time streaming at higher rates.

DPDK is a high-speed packet processing framework that enables a kernel bypass for network drivers. By putting the entire driver in user space, avoiding context switches, and pinning I/O threads to cores, UHD and DPDK combine to largely prevent the latency spikes induced by the scheduler. In addition, the overall overhead for packet processing lowers.

29.4

29.4.1

DPDK is supported on the following USRP devices:

- E320
- N300 / N310
- N320 / N321
- X300 / X310
- X410
- X440

29.4.2

DPDK is supported on many Intel and Mellanox based 10Gb and 100Gb NICs and lots of other NICs. Below is a list of NICs Ettus Research has tested. For a full list of NICs supported by DPDK, please see the DPDK manual.

- Intel X520-DA1 (1x10Gb)
- Intel X520-DA2 (2x10Gb)
- Intel X710-DA2 (2x10Gb)
- Intel X710-DA4 (4x10Gb)
- Intel XL710-QDA2 (2x40Gb breakout to 4x10Gb)
- Intel E810-CQDA1 & E810-CQDA2 (2x100Gb and 2x4x10Gb)
- Mellanox MCX4121A-ACAT ConnectX-4 Lx (2x10Gb)
- Mellanox MCX515A-CCAT ConnectX-5 EN (2x100Gb or 2x10Gb)
- Mellanox MCX516A-CCAT ConnectX-5 EN (2x100Gb or 2x10Gb)
- Mellanox MCX516A-CDAT ConnectX-5 Ex EN (2x100Gb or 2x10Gb)
- Mellanox MCX623106AN-CDAT ConnectX-6 Dx EN (2x100Gb or 2x10Gb)
- NI Dual 100 Gigabit Ethernet PCIe Interface Kit (PN 788216-01)

29.5

- DPDK: https://www.dpdk.org/
    - https://doc.dpdk.org/guides-17.11
    - https://doc.dpdk.org/guides-18.11
    - https://doc.dpdk.org/guides-19.11
    - https://doc.dpdk.org/guides-20.11
    - https://doc.dpdk.org/guides-21.11
    - https://doc.dpdk.org/guides-22.11
    - https://doc.dpdk.org/guides-23.11
- UHD Manual: https://files.ettus.com/manual/page_dpdk.html

29.6

- UHD 3.x requires DPDK 17.11
- UHD 4.0 and 4.1 require DPDK 18.11
- UHD 4.2 through 4.6 can use any version of DPDK from 18.11 through and including 21.11

29.7

We recommend installing DPDK via the system-provided installer; for example with Ubuntu:

```
sudo apt install dpdk dpdk-dev
```

While it is possible to install DPDK from source, we recommend using the system-provided install unless there is a very good reason to not do so. DPDK can be challenging to *correctly* build from source. If you require installing DPDK from source, the install guide for various versions is noted below:

- DPDK 17.11
- DPDK 18.11
- DPDK 19.11

- DPDK 20.11
- DPDK 21.11
- DPDK 22.11
- DPDK 23.11

DPDK releases come twice per year in April (version X.04) and November (version X.11). The first release is more of a beta while the second is more of a formal release. While either *can* be used with UHD, we recommend just the formal releases with version ending in 11.

29.8

Once the `dpdk` and `dpdk-dev` packages are installed, UHD will locate them during a build and you should see DPDK in the enabled components lists when running `cmake`.

29.9

Edit your grub configuration file, `/etc/default/grub` and add the follow parameters to `GRUB_CMDLINE_LINUX_DEFAULT`:

```
iommu=pt intel_iommu=on hugepages=2048
```

On a vanilla Ubuntu system it should look like this:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash iommu=pt intel_iommu=on hugepages=2048"
```

Close `/etc/default/grub` and at the command prompt, update your grub configuration with the command:

```
sudo update-grub
```

For these settings to take effect, reboot your host machine.

29.10

You must note the MAC addresses for your NICs before proceeding.

The MAC addresses for your NICs can be found by running the command:

```
ip a
```

You must then create a UHD configuration file at the location `/root/.uhd/uhd.conf`.

```
sudo su
mkdir -p /root/.uhd
nano /root/.uhd/uhd.conf
```

29.10.1

An example `uhd.conf` file is listed below. Note that field names in UHD 3.x are slightly different from UHD 4.0.

You should update the following fields for your configuration from this example:

- Update the MAC address variables, `dpdk-mac`, to match your NIC
- Update the `dpdk-driver` if the location is different on your system. `/usr/lib/x86_64-linux-gnu/dpdk-17.11-drivers/` is the default location on Ubuntu 18.04.x when `dpdk` is installed via `apt`.
- Update the `dpdk-corelist` and `dpdk-io-cpu` fields. In this example, a two port NIC is used. There should be one core for the main `dpdk` thread (in this example `core #2`), and then separate cores assigned to each NIC (in this example `core #3` for the first port on the NIC, `core #4` for the second port on the NIC)
- Update the `dpdk-ipv4` fields to your desired IP range.
  - `192.168.30.2`, `192.168.40.2` on a default X3xx system
  - `192.168.10.2`, `192.168.20.2` on a default N3xx system
  - `192.168.10.2` on a default E320 system

```
[use_dpdk=1]
dpdk-mtu=9000
dpdk-driver=/usr/lib/x86_64-linux-gnu/dpdk-17.11-drivers/
dpdk-corelist=2,3,4
dpdk-num-mbufs=4095
dpdk-mbufs-cache-size=315

[dpdk-mac=aa:bb:cc:dd:ee:f1]
dpdk-io-cpu = 3
dpdk-ipv4 = 192.168.10.1/24

[dpdk-mac=aa:bb:cc:dd:ee:f2]
dpdk-io-cpu = 4
dpdk-ipv4 = 192.168.20.1/24
```

Note: Additional information on the UHD configuration file can be found here:
https://files.ettus.com/manual_archive/v3.15.0.0/html/page_dpdk.html#dpdk_nic_config

29.10.2

An example `uhd.conf` file is listed below. Note that the field names in UHD 4.x are slightly different from UHD 3.x.

You must verify and/or update the following fields for your configuration from this example:

- The MAC address variables, `dpdk_mac`, to match your NIC(s) link(s)
- The `dpdk_driver` if the location is different on your system. `/usr/local/lib/` is the default location on when DPDK is built and installed from source.

- The `dpdk_corelist` and `dpdk_lcore` fields. In this example, a two port NIC and both links are used. There must be one core for the main `dpdk` thread (in this example `core #2` because it is not otherwise used in the file) and then separate cores assigned to each NIC link (in this example `core #3` for the one of the NIC links and `core #4` for the other NIC link).
- The `dpdk_ipv4` fields to your desired IP range(s).
  - `192.168.30.2`, `192.168.40.2` on a default X3xx system
  - `192.168.10.2`, `192.168.20.2` on a default N3xx system
  - `192.168.10.2` on a default E320 system
  - `192.168.10.2` and `192.168.20.2` on a default X4xx systems

```
[use_dpdk=1]
dpdk_mtu=9000
dpdk_driver=/usr/lib/x86_64-linux-gnu/dpdk/pmds-20.0/
dpdk_corelist=2,3,4
dpdk_num_mbufs=4095
dpdk_mbuf_cache_size=315

[dpdk_mac=aa:bb:cc:dd:ee:f1]
dpdk_lcore = 3
dpdk_ipv4 = 192.168.10.1/24

[dpdk_mac=aa:bb:cc:dd:ee:f2]
dpdk_lcore = 4
dpdk_ipv4 = 192.168.20.1/24
```

Notes:

- Additional information on the UHD DPDK configuration file is in the UHD manual.
- The number of cores listed must be used exactly within the file.
- The number of NIC link(s) listed must exactly match those specified in the UHD instantiation args.
- A simple way to move between different DPDK configurations is to create different files and then symlink from the desired file to `uhd.conf`

29.11

The process for this step is different for Intel and Mellanox NICs and is detailed in individual sections below.

29.11.1

The Intel based NICs will use the `vfio-pci` driver which must be loaded:

```
sudo modprobe vfio-pci
```

Next, you will need to rebind the NIC to the `vfio-pci` drivers.

First, identify the PCI address your NIC is at:

```
dpdk-devbind --status
```

Note the PCI address that your NIC is connected to for the next step.

Before the next step, you will need to turn off the NIC first before doing the rebind.

In Ubuntu under `System` -> `Network` -> click the switches to `off` for the 10Gb ports, then run the `dpdk-devbind` commands:

Note: Your PCI address will likely be different than `02:00.0` as shown in the example below.

```
sudo dpdk-devbind --bind=vfio-pci 02:00.0
sudo dpdk-devbind --bind=vfio-pci 02:00.1
```

Now if you run `dpdk-devbind --status` again, you should see the NICs listed under DPDK devices

```
# dpdk-devbind --status

Network devices using DPDK-compatible driver
============================================
0000:02:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' drv=vfio-pci unused=ixgbe
0000:02:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' drv=vfio-pci unused=ixgbe
```

Note: More info can be found here on the rebinding process:
https://doc.dpdk.org/guides-17.11/linux_gsg/linux_drivers.html#binding-and-unbinding-network-ports-to-from-the-kernel-modules

29.11.2

The Mellanox NICs do not require rebinding using the `vfio-pci` driver. Mellanox provides additional drivers for DPDK.

Install and activate the Mellanox drivers:

```
sudo apt install librte-pmd-mlx5-17.11
sudo modprobe -a ib_uverbs mlx5_core mlx5_ib
```

For 18.11 you can download and install the latest Mellanox drivers from the mellanox website (https://www.mellanox.com/products/infiniband-drivers/linux/mlnx_ofed).

The MLX5 poll mode driver library (librte_pmd_mlx5) in DPDK provides support for Mellanox ConnextX-4 and ConnectX-5 cards. This driver must be enabled manually with the build option `CONFIG_RTE_LIBRTE_MLX5_PMD=y` when building DPDK.

29.12

UHD based application (including GNU Radio flowgraphs) can now be ran using a DPDK transport by passing in the Device Argument: `use_dpdk=1`.

Important Note: In order for UHD to use DPDK, the UHD application *must* be ran as the `root` user. Using `sudo` will not work, you should switch to the `root` user by running `sudo su`.

For example, running the `benchmark_rate` utility:

```
# cd /usr/local/lib/uhd/examples

# ./benchmark_rate --rx_rate 125e6 --rx_subdev "A:0 B:0" --rx_channels 0,1 --tx_rate 125e6 --tx_subdev "A:0 B:0" --tx_channels 0,1 --args "ad

[INFO] [UHD] linux; GNU C++ version 7.3.0; Boost_106501; UHD_3.14.0.HEAD-0-gabf0db4e
EAL: Detected 8 lcore(s)
EAL: Some devices want iova as va but pa will be used because.. EAL: IOMMU does not support IOVA as VA
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: PCI device 0000:02:00.0 on NUMA socket -1
EAL:   Invalid NUMA socket, default to 0
EAL:   probe driver: 8086:10fb net_ixgbe
EAL:   using IOMMU type 1 (Type 1)
EAL: Ignore mapping IO port bar(2)
EAL: PCI device 0000:02:00.1 on NUMA socket -1
EAL:   Invalid NUMA socket, default to 0
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: Ignore mapping IO port bar(2)
PMD: ixgbe_dev_link_status_print():  Port 0: Link Down
EAL: Port 0 MAC: aa bb cc dd ee f1
EAL: Port 0 UP: 1
PMD: ixgbe_dev_link_status_print():  Port 1: Link Down
EAL: Port 1 MAC: aa bb cc dd ee f2
EAL: Port 1 UP: 1
EAL: Init DONE!
EAL: Starting I/O threads!
USER2: Thread 1 started
[00:00:00.000003] Creating the usrp device with: addr=192.168.10.2,second_addr=192.168.20.2,mgmt_addr=10.2.1.19,master_clock_rate=125e6,use_d
[INFO] [MPMD] Initializing 1 device(s) in parallel with args: mgmt_addr=10.2.1.19,type=n3xx,product=n310,serial=313ABDA,claimed=False,addr=19
[INFO] [MPM.PeriphManager] init() called with device args 'product=n310,time_source=internal,master_clock_rate=125e6,clock_source=internal,us
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000004)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1344 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1341 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1348 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1347 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000011312)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000011312)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000002)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000002)
Using Device: Single USRP:
  Device: N300-Series Device
  Mboard 0: ni-n3xx-313ABDA
  RX Channel: 0
    RX DSP: 0
    RX Dboard: A
    RX Subdev: Magnesium
  RX Channel: 1
    RX DSP: 0
    RX Dboard: B
    RX Subdev: Magnesium
  TX Channel: 0
    TX DSP: 0
    TX Dboard: A
    TX Subdev: Magnesium
  TX Channel: 1
    TX DSP: 0
    TX Dboard: B
    TX Subdev: Magnesium

[00:00:03.728707] Setting device timestamp to 0...
[INFO] [MULTI_USRP]     1) catch time transition at pps edge
[INFO] [MULTI_USRP]     2) set times next pps (synchronously)
[00:00:05.331920] Testing receive rate 125.000000 Msps on 2 channels
[00:00:05.610789] Testing transmit rate 125.000000 Msps on 2 channels
[00:00:15.878071] Benchmark complete.


Benchmark rate summary:
  Num received samples:     2557247854
  Num dropped samples:      0
  Num overruns detected:    0
  Num transmitted samples:  2504266704
  Num sequence errors (Tx): 0
  Num sequence errors (Rx): 0
  Num underruns detected:   0
  Num late commands:        0
  Num timeouts (Tx):        0
  Num timeouts (Rx):        0


Done!
```

29.13

29.13.1

Perform the general host performance tuning tips and tricks.

29.13.2

If you experience `Overflows` at higher data rates, adding the device argument `num_recv_frames=512` can help.

29.13.3

If you're streaming data at the full master clock rate, and there is no interpolation or decimation being performed on the FPGA, you can skip the DUC and DDC blocks within the FPGA with the following parameters:

```
skip_ddc=1
```

```
skip_duc=1
```

29.13.4

If you're streaming two transmit channels at full rate (200e6) on the X3xx platform, you should additionally set the following device arg:

```
enable_tx_dual_eth=1
```

29.13.5

Isolating the cores that are used for DPDK can improve performance. This can be done by adding the `isolcpus` parameter to your `/etc/default/grub` file in the `GRUB_CMDLINE_LINUX_DEFAULT=""` (the "`GRUB_CONFIG`"). For example, to isolate cores 2 through and including 4, add this entry:

```
isolcpus=2-4
```

NOTE: After saving the `GRUB_CONFIG` file, execute `sudo update-grub` and then reboot the computer for the changes to take effect. It is OK to isolate core used by DPDK via this method.

29.13.6

Disabling system interrupts can improve the jitter and performance generally by 1-3%. This can be done by adding the parameters `nohz_full` and `rcu_nocbs` to your `GRUB_CONFIG`. For example, to disable system interrupts on cores 2 through and including 4, add this entry:

```
nohz_full=2-4 rcu_nocbs=2-4
```

NOTE: After saving the `GRUB_CONFIG` file, execute `sudo update-grub` and then reboot the computer for the changes to take effect. It is OK to isolate core used by DPDK via this method.

29.13.7

If you're streaming on multiple channels simultaneously, you can create multiple streamer objects on separate threads. This can be accomplished with the `benchmark_rate` example by using the parameter `--multi_streamer`.

29.13.8

In UHD 4, streaming thread priorities can be elevated with the `uhd::set_thread_priority_safe()` function call. This can be accomplished with the `benchmark_rate` example by using parameter `--priority high`. Note that if the Thread Schedule Priority has not been enabled for the current user then this parameter requires `sudo` to work.

29.13.9

Beyond elevating streaming thread priority, one can also increase the `nice` priority level to maximum to increase the amount of CPU time for the process and its thread by prepending the following to the command being issued:

```
sudo nice -n -20
```

29.13.10

With Linux one can limit the cores being used by the threads in a process via a `taskset` prepended to a command being executed. When combining with the other techniques listed here, one can fairly well constrain a process and its thread to specific cores and give them maximum CPU time. Note that when using DPDK the MAC cores specified in `uhd.conf` will already be included as part of the `taskset` (but those cores will not be isolated nor have system interrupts disabled on them unless using that setting as noted above); it generally won't hurt to include the DPDK cores as part of the overall `taskset` but it's not required. For example, to limit process/thread execution to cores 2 through and including 4, one would prepend the following (note that `sudo` is *not* required):

```
taskset -c "2-4"
```

NOTE: For best performance do *not* include the cores used by DPDK in the `dpdk_corelist` argument in the `taskset`.

29.13.11

The Linux kernel spawns a number of processes and threads that spend most of their time sleeping; one cannot stop these processes/threads. That said, in a typical Linux install (for example Ubuntu) there will be a graphical desktop (e.g., `gdm`) and various daemons started up by user request (e.g., `containerd`, `docker`, `snapd`). Most of these daemons can be controlled by `systemctl`; some must be manually quit. Any process that runs regularly stands a chance of interrupting the UHD and DPDK threads, and thus stopping, quitting, or disabling those processes can increase performance. For example, the following commands stop various daemons that execute regularly on Ubuntu; these need to be executed with `sudo` and some might not exist on your specific system and for these the command will do nothing so it's OK to run it:

```
systemctl stop containerd containerd.socket
systemctl stop docker docker.socket
systemctl stop dbus dbus.socket
systemctl stop snapd snapd.socket
systemctl stop udev systemd-udevd-control.socket systemd-udevd-kernel.socket
```

The following command stops the main Ubuntu desktop GUI, so running it will render the system accessible only via networking (e.g., `ssh`), so make sure network access is enabled before executing this command:

```
systemctl stop gdm
```

29.13.12

In some applications which require the highest possible CPU performance per core, disabling hyper-threading can provide roughly a 10% increase in core performance, at the cost of having fewer core threads. Hyper-threading is disabled within the BIOS and how to do this varies by motherboard manufacturer. With other techniques listed here, disabling hyper-threading should only be done as a last resort to eek absolute maximum performance from the CPU.

29.13.13

- Performance report from Intel on DPDK 17.11: https://fast.dpdk.org/doc/perf/DPDK_17_11_Intel_NIC_performance_report.pdf

- How to get best performance with NICs on Intel platforms: https://doc.dpdk.org/guides/linux_gsg/nic_perf_intel_platform.html

29.14

29.14.1

On Multi-CPU systems each CPU is a NUMA node. The NUMA node contains all of the cores on the CPU. For best performance when using CPU/core isolation ("isocpus" per AAA above) make sure that all of the cores are in the same NUMA node.

29.14.2

With Linux kernels 5.10 and beyond, we have observed periodic underruns on systems that otherwise have no issues. These Linux kernel versions are the default for Ubuntu 20.04.3 LTS and later. The underrun issue is due to the `RT_RUNTIME_SHARE` feature being disabled by default in these versions of the Linux kernel (shown as `NO_RT_RUNTIME_SHARE`). The following procedure can be used to enable this feature. This process was tested on Linux kernel version 5.13; the procedure may be slightly different on other kernel versions. To determine the Linux kernel version of your system, in a terminal issue the command `uname -r`.

```
$ sudo -s
$ cd /sys/kernel/debug/sched
$ cat features | tr ' ' '\n' | grep RUNTIME_SHARE
NO_RT_RUNTIME_SHARE
$ echo RT_RUNTIME_SHARE > features
$ cat features | tr ' ' '\n' | grep RUNTIME_SHARE
RT_RUNTIME_SHARE
```

30 B205mini

30.1

The USRP Bus Series provides a fully integrated, single board, Universal Software Radio Peripheral platform with continuous frequency coverage from 70 MHz ? 6 GHz. Designed for low-cost experimentation, it combines a fully integrated direct conversion transceiver providing up to 56MHz of real-time bandwidth, an open and reprogrammable Spartan6 FPGA, and fast and convenient bus-powered SuperSpeed USB 3.0 connectivity.

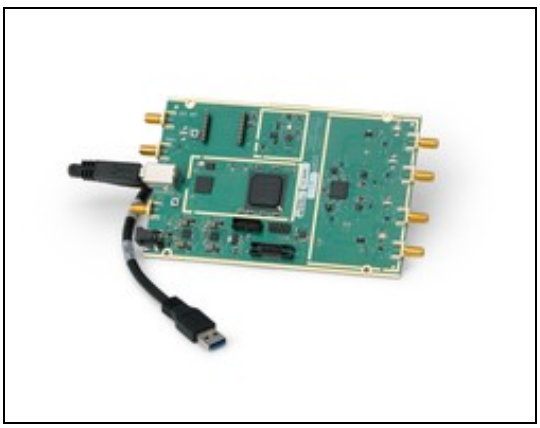
30.2

30.2.1

- Xilinx Spartan 6 XC6SLX75 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Full duplex, SISO (1 Tx & 1 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity
- Optional Board Mounted GPSDO

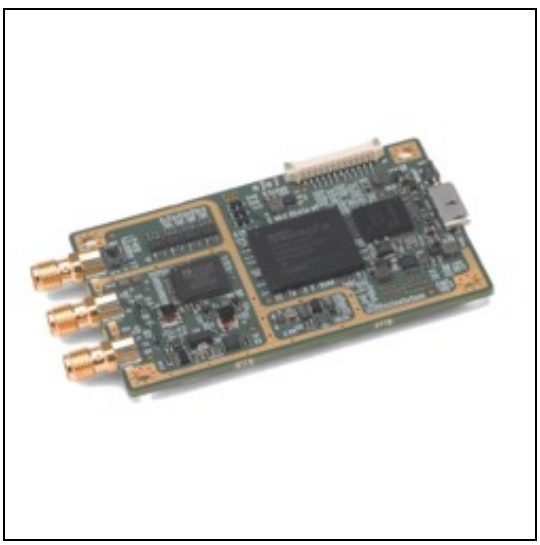
30.2.2

- Xilinx Spartan 6 XC6SLX150 FPGA
- Analog Devices AD9361 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth (61.44MS/s quadrature)
- Full duplex, MIMO (2 Tx & 2 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity
- Optional Board Mounted GPSDO



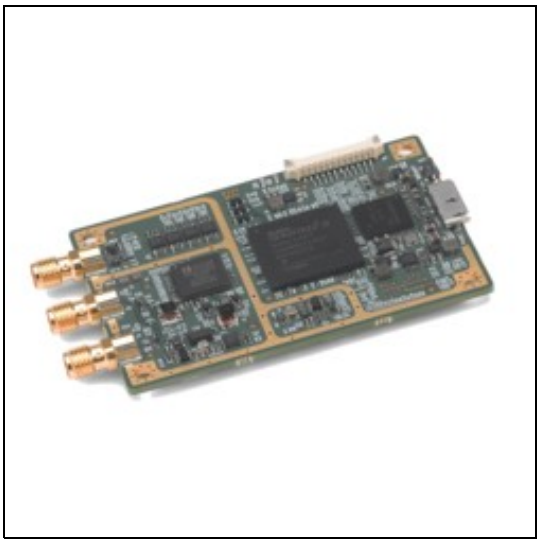
30.2.3

- Xilinx Spartan-6 XC6SLX75 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Full duplex, SISO (1 Tx & 1 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity



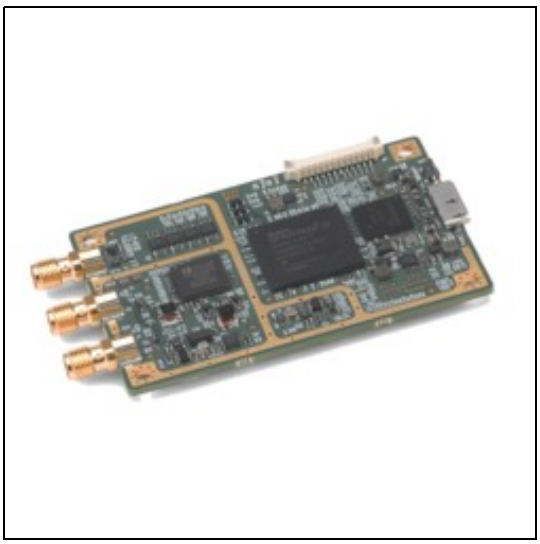
30.2.4

- Industrial-grade Xilinx Spartan-6 XC6SLX75 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Full duplex, SISO (1 Tx & 1 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity



30.2.5

- Industrial-grade Xilinx Spartan-6 XC6SLX150 FPGA
- Analog Devices AD9364 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- Full duplex, SISO (1 Tx & 1 Rx)
- Fast and convenient bus-powered USB 3.0 connectivity



30.3

30.3.1

The RF frontend has individually tunable receive and transmit chains. On the B200 and B200 mini, there is one transmit and one receive RF frontend. On the B210, both transmit and receive can be used in a MIMO configuration. For the MIMO case on the B210 only, both receive frontends share the RX LO, and both transmit frontends share the TX LO. Each LO is independently tunable between 50 MHz and 6 GHz and can be used with 1 or 2 channels; all channels using the same LO must use the same sampling parameters, including the sample rate and RF center frequency.

30.3.2

All frontends have individual analog gain controls. The receive frontends have 76 dB of available gain; and the transmit frontends have 89.8 dB of available gain. Gain settings are application specific, but it is recommended that users consider using at least half of the available gain to get reasonable dynamic range.

30.3.3

The analog frontend has a seamlessly adjustable bandwidth of 200 kHz to 56 MHz.

Generally, when requesting any possible master clock rate, UHD will automatically configure the analog filters to avoid any aliasing (RX) or out-of-band emissions whilst letting through the cleanest possible signal.

If you, however, happen to have a very strong interferer within half the master clock rate of your RX LO frequency, you might want to reduce this analog bandwidth. You can do so by calling uhd::usrp::multi_usrp::set_rx_bandwidth(bw).

The property to control the analog RX bandwidth is bandwidth/value.

UHD will not allow you to set bandwidths larger than your current master clock rate.

30.4

The USRP B200/B210/B200mini/B205mini are derived from the Analog devices AD936x integrated transceiver chip, the overall RF performance of the device is largely governed by the transceiver chip itself.

30.4.1

- SSB/LO Suppression -35/50 dBc
- Phase Noise 3.5 GHz 1.0 deg RMS
- Phase Noise 6 GHz 1.5 deg RMS
- Power Output >10dBm
- IIP3 (@ typ NF) -20dBm
- Typical Noise Figure <8dB
- Maximum Input Power: -15 dBm

30.4.2

All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

30.4.3

- The maximum input power for the B200/B210/B200mini/B205mini is 0 dBm.

30.4.4

30.4.4.1

- Media:B200mini B205 RF Performance Data 20160119.pdf

30.4.4.2

- Media:B200 RF Performance.pdf

30.5

- Ettus Research recommends to always use the latest stable version of UHD

30.5.1

- Current Hardware Revision: 6
- Minimum version of UHD required: 3.8.4
- B200 Rev 5 (AD9364-based board) requires minimum UHD 3.8.4

30.5.2

- Current Hardware Revision: 5
- Minimum version of UHD required: 3.6.0

30.5.3

- Current Hardware Revision: 2
- Minimum version of UHD required: 3.9.0

30.5.4

- Current Hardware Revision: 2
- Minimum version of UHD required: 3.9.0

30.5.5

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.9.2

30.6

30.6.1

- B200mini/B205mini 5.0 x 8.4 cm
- B200/B210 9.7 x 15.5 x 1.5 cm

30.6.2

- B200mini 24.0 g
- B200/B210 350 g

30.6.3

30.6.3.1

- Board only
- B20xmini Enclosure

30.6.3.2

- Board only

30.6.3.3

- Board only

30.6.3.4

- Enclosure

30.6.4

30.6.4.1

- B200mini with Enclosure
- Enclosure only
- Board only

30.6.4.2

- B20xmini-i Thermal Insert

30.6.4.3

- Board only

30.6.4.4

- Board only

30.6.4.5

- Enclosure

30.7

30.7.1

- B200 / B210: 25 °C
- B200mini - Board Only: 0 - 40 °C
- B200mini - With Enclosure: -20 - 60°C
- B200mini-i / B205mini-i - Board Only: 0 - 45 °C
- B200mini-i / B205mini-i - With I-Grade Enclosure: -40 - 75°C

30.7.2

- 10% to 90% non-condensing

30.8

30.8.1

B200mini/B200mini-i/B205mini-i Schematics

30.8.2

B200/B210 Schematics

30.9

Part Number	Description	Schematic ID (Page)
Mini-Circuits TCM1-63AX+	Transformer	T1 (1,3); T2 (1,3)
Analog Devices AD9364	RF Transceiver	U1 (2)
Analog Devices AD9361	RF Transceiver	U2 (2,8)
AD9361/AD9364 Product Page	RF Transceiver	-
Xilinx Spartan-6 Product Page XC6SLX75 / XC6SLX150	FPGA FPGA	U1 (2,3,4,6); PG1 (6); U18B, U18C (7); U18D (8); U18E, U18F (9); U18G, U18H (10)
ADF4001	Frequency Synthesizer	U101 (1)
CYUSB3014 EZ-USB FX3? Product Page	FX3: SuperSpeed USB Controller	U3 (5,6); U13 (5)
SKY13317	Antenna Switch	U801, U810 (8)
BD3150L50100A00	Balun	U802, U808, U809, U815 (8)
PGA?102+	Amplifier	U804, U817 (8)
VCTCXO	VCTCXO (B200mini only)	-
525L20DA40M0000	VCTCXO (B200/B210 only)	X100 (1)
Jackson Labs LC_XO Spec Sheet Manual	Optional GPSDO (B200/B210 only)	U100 (1)

30.10

- SMA connectors should be torqued to 4 inch-pounds

30.10.1

- B200mini C-Grade Enclosure
- B200mini I-Grade Enclosure

30.10.2

- B205mini I-Grade Enclosure

30.10.3

- USRP B200/B210 Enclosure
  - Full Steel Enclosure
  - Compatible with green USRP B200 and B210 devices (revision 6 or later)
  - Front and rear K-Slots for anti-theft protection

30.11

- Utilization statistics are subject to change between UHD releases. This information is current as of UHD 3.9.4.

30.11.1

```
Device utilization summary:
---------------------------

Selected Device : 6slx75fgg484-3


Slice Logic Utilization:
 Number of Slice Registers:           15781  out of  93296    16%
 Number of Slice LUTs:                19987  out of  46648    42%
    Number used as Logic:             15983  out of  46648    34%
    Number used as Memory:             4004  out of  11072    36%
       Number used as RAM:              972
       Number used as SRL:             3032

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  24062
   Number with an unused Flip Flop:    8281  out of  24062    34%
   Number with an unused LUT:          4075  out of  24062    16%
   Number of fully used LUT-FF pairs: 11706  out of  24062    48%
   Number of unique control sets:       434

IO Utilization:
 Number of IOs:                         172
 Number of bonded IOBs:                 155  out of    280    55%
    IOB Flip Flops/Latches:             124

Specific Feature Utilization:
 Number of Block RAM/FIFO:              144  out of    172    83%
    Number using Block RAM only:        144
 Number of BUFG/BUFGCTRLs:                4  out of     16    25%
 Number of DSP48A1s:                     76  out of    132    57%
```

30.11.2

```
Device utilization summary:
---------------------------

Selected Device : 6slx150fgg484-3


Slice Logic Utilization:
 Number of Slice Registers:           29310  out of  184304   15%
 Number of Slice LUTs:                36486  out of  92152    39%
    Number used as Logic:             29279  out of  92152    31%
    Number used as Memory:             7207  out of  21680    33%
       Number used as RAM:             1752
       Number used as SRL:             5455

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  43635
   Number with an unused Flip Flop:   14325  out of  43635    32%
   Number with an unused LUT:          7149  out of  43635    16%
   Number of fully used LUT-FF pairs: 22161  out of  43635    50%
   Number of unique control sets:       723

IO Utilization:
 Number of IOs:                         180
 Number of bonded IOBs:                 163  out of    338    48%
    IOB Flip Flops/Latches:             148

Specific Feature Utilization:
 Number of Block RAM/FIFO:              186  out of    268    69%
    Number using Block RAM only:        186
 Number of BUFG/BUFGCTRLs:                4  out of     16    25%
 Number of DSP48A1s:                    152  out of    180    84%
```

30.11.3

```
Device utilization summary:
---------------------------

Selected Device : 6slx75csg484-3


Slice Logic Utilization:
 Number of Slice Registers:           15949  out of  93296    17%
 Number of Slice LUTs:                19963  out of  46648    42%
    Number used as Logic:             16140  out of  46648    34%
    Number used as Memory:             3823  out of  11072    34%
       Number used as RAM:              972
       Number used as SRL:             2851

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  23859
```

```
   Number with an unused Flip Flop:    7910  out of  23859    33%
   Number with an unused LUT:          3896  out of  23859    16%
   Number of fully used LUT-FF pairs: 12053  out of  23859    50%
   Number of unique control sets:       429

IO Utilization:
 Number of IOs:                        123
 Number of bonded IOBs:                114  out of    328    34%
    IOB Flip Flops/Latches:            147

Specific Feature Utilization:
 Number of Block RAM/FIFO:             110  out of    172    63%
    Number using Block RAM only:       110
 Number of BUFG/BUFGCTRLs:               6  out of     16    37%
 Number of DSP48A1s:                    76  out of    132    57%
 Number of PLL_ADVs:                     1  out of      6    16%
```

30.11.4

```
Device utilization summary:
---------------------------

Selected Device : 6slx150csg484-3


Slice Logic Utilization:
 Number of Slice Registers:           15949  out of  184304     8%
 Number of Slice LUTs:                19963  out of  92152    21%
    Number used as Logic:             16140  out of  92152    17%
    Number used as Memory:             3823  out of  21680    17%
       Number used as RAM:              972
       Number used as SRL:             2851

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  23859
   Number with an unused Flip Flop:    7910  out of  23859    33%
   Number with an unused LUT:          3896  out of  23859    16%
   Number of fully used LUT-FF pairs: 12053  out of  23859    50%
   Number of unique control sets:       429

IO Utilization:
 Number of IOs:                        123
 Number of bonded IOBs:                114  out of    338    33%
    IOB Flip Flops/Latches:            147

Specific Feature Utilization:
 Number of Block RAM/FIFO:             110  out of    268    41%
    Number using Block RAM only:       110
 Number of BUFG/BUFGCTRLs:               6  out of     16    37%
 Number of DSP48A1s:                    76  out of    180    42%
 Number of PLL_ADVs:                     1  out of      6    16%
```

30.12

B200/B210/B200mini - USB 3.0

30.12.1

30.12.1.1

The hardware power on state and UHD initial state for the front-panel GPIOs is high-Z. For the B2xx, B2xxmini there are no external pull-ups/pull-downs for the GPIO pins, but the FPGAs do have them and they are configured as follows: B2xx: pull-up, B2xxmini: pull-up.

30.12.1.2

The GPIOs are configured as LVCMOS33 outputs with pull-ups on the B2xx. The strength for LVCMOS and LVTTL on Spartan 6 is 12 mA if not otherwise specified.

30.12.2

30.12.2.1

- 1-PPS or 10 MHz input

30.12.2.1.1

- Maximum: -5V / +5V
- Minimum: 0V / +2.5V

30.12.2.1.2

- Maximum: 0V / +5V
- Minimum: 0V / +1.8V

OR

- +10dBm ~ +27dBm

30.12.2.2

30.12.2.2.1

- Maximum: 5V

30.12.2.2.2

- Maximum: 15dBm (3.5Vpp into 50 ohms)

30.13

30.13.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

30.13.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

30.13.3

In order to ensure compliance with EU certifications for radio equipment, a ferrite bead (included in kits with NI part number 785825-01 and 785826-01) should be affixed onto the GPIO cable, if in use. This is achieved by opening the snap-on ferrite bead and enclosing it around the GPIO cable(s).

In addition to the part numbers listed above, these ferrite beads can be sourced through Fair-Rite using part number 0443164251.

30.14

Found on the NI Product Certifications lookup tool here.

30.15

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

30.16

This is a list of frequently asked questions on the USRP B200/B210/B200mini. If you have questions that are not answered in this document, please contact us - info@ettus.com.

Will the USRP B200/B210 work with USB 2.0?

Yes, both the USRP B200 and USRP B210 will fall back to the USB 2.0 standard if a USB 3.0 port is not available. There are several things to consider. First, the USB 2.0 data rates are slower. Depending on the USB controller, operating system, and other factors, you may achieve a sample rate up to 8 MS/s with USB 2.0. Also, you may not be able to bus-power the USRP B200/B210 in USB 2.0 mode.

What samples rates should I expect with USB 3.0? USB 2.0?

The performance and throughput of USB 3.0 can vary between host controllers. Ettus Research recommends using the Intel Series 7, 8, and 9 USB controllers. In Linux, the command `lspci` will show the USB controller on the system.

When can I power the USRP B200/B210/B200mini off USB?

The experience will vary across various controllers. Generally speaking, bus-power is ideal for SISO operation. If you are using both channels of a USRP B210 we recommend an external power supply. We sell an external power supply that works with a variety of USRPs.

MIMO operation with the USRP B210 is not recommended when using the USRP B210 on bus-power. It is also not recommended to run the B210 on bus-power if a GPS-disciplined oscillator is installed.

How much power does the USRP consume?

The table below shows power consumption (Watts) of a USRP B210 run with a 6V power supply. Figures on a 5V supply (USB power), or with a USRP B200 will be moderately lower. The sample rates shown are aggregate sample rates on the USB 3.0 interface.

	5 Msps	15.36 Msps	30.72 Msps	56 Msps	61.44 Msps
1 RX	1.92	2.112	2.184	2.508	
2 RX	2.148	2.436	2.508	2.64	
1 TX	2.184	2.34	2.352	2.22	
2 TX	2.76	2.88	2.904	2.64	
Full Duplex (1x1)	2.508	2.736	2.796	3.168	
2x2 MIMO	3.252	3.588	3.672	4.11	4.092

Can I build a multi-unit system with the USRP B200/B210?

It is possible to synchronize multiple USRP B200/B210 devices using the 10 MHz/1 PPS inputs and an external distribution system like to the OctoClock-G. However, USB 3.0/2.0 performance varies dramatically when multiple devices are streaming through the same controller. Generally, we recommend using the USRP N200/N210 if you need to build a high-channel count system.

Can I access the source code for the USRP B200/B210?

Yes. The USRP B200/B210 is supported by the USRP Hardware DriverTM software. You can find the driver and FPGA source code for the USRP B200/B210, and all other USRP models, in the UHD git repository:

http://files.ettus.com/manual/page_build_guide.html

What operating systems does the USRP B200/B210 work on?

The USRP B200/B210 is supported on Linux, OSX (MacOSX / macOS) and Windows.

Does the USRP B200/B210 work with GNU Radio?

Yes. The USRP B200/B210 work with our GNU Radio plugin - gr-uhd.

Does the USRP B200/B210 work with MATLAB and Simulink?

Yes. You need to install the Communications System Toolbox Support Package for USRP Radio.

Does the USRP B200/B210 work with OpenBTS?

Yes. This is a third-party application and you can find instructions here: OpenBTS - Build, Install, Run.

For support, please sign up and contact the OpenBTS mailing list.

What tools do I need to program the FPGA?

The USRP B200 and USRP B210 include a Spartan 6 XC6SLX75 and XC6S150, respectively. The USRP B200 can be programmed with the free version of Xilinx tools, while the larger FPGA on the USRP B210 requires a licensed seat.

Can I use a GPSDO with the USRP B200/B210?

Ettus Research offers a Board-Mounted GPS-Disciplined OCXO and a Board-Mounted GPS-Disciplined TCXO, which are compatible with the USRP B200/B210. These provide a high-accuracy XO, which can be disciplined to the global GPS standard. Please note: When the GPSDO OCXO model is integrated on the USRP B200/B210, the device should be powered with an external supply instead of USB bus power. The TCXO version can be USB bus powered.

31 Ettus USRP E300 Embedded Family Hardware Resources

31.1

The USRP E31x offers a portable stand-alone SDR platform designed for field deployment. The flexible 2x2 MIMO AD9361 transceiver from Analog Devices provides up to 56 MHz of instantaneous bandwidth and spans frequencies from 70 MHz ? 6 GHz to cover multiple bands of interest.

31.2

31.2.1

- Xilinx Zynq 7020 SoC: 7 Series FPGA with ARM Cortex A9 667 MHz (SG1) or 866 MHz (SG3) dual-core processor
- Analog Devices AD9361 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- 2x2 MIMO transceiver
- Up to 10 MS/s sample data transfer rate to ARM processor
- RX, TX filter banks
- Integrated GPS receiver
- 9-axis inertial measurement unit
- RF Network on Chip (RFNoC?) FPGA development framework support

31.2.2

- Battery Operated
- Xilinx Zynq 7020 SoC: 7 Series FPGA with ARM Cortex A9 866 MHz dual-core processor

31.2.3

- Rugged and weatherproof for outdoor deployment
- Fully assembled IP67-rated enclosure with USRP E310 inside
- Extensive environmental testing
- Power over Ethernet (PoE) with surge and lightning protection
- Xilinx Zynq 7020 SoC: 7 Series FPGA with ARM Cortex A9 866 MHz dual-core processor

31.3

31.3.1

The USRP E31x MIMO XCVR daughterboard features an integrated MIMO capable RF frontend.

31.3.2

The RF frontend has individually tunable receive and transmit chains. Both transmit and receive can be used in a MIMO configuration. For the MIMO case, both receive frontends share the RX LO, and both transmit frontends share the TX LO. Each LO is tunable between 50 MHz and 6 GHz.

31.3.3

All frontends have individual analog gain controls. The receive frontends have 76 dB of available gain; and the transmit frontends have 89.8 dB of available gain. Gain settings are application specific, but it is recommended that users consider using at least half of the available gain to get reasonable dynamic range.

31.3.4

The frontends provide a lo-locked sensor that can be queried through the UHD API.

31.3.5

The transmit and receive filter banks uses switches to select between the available filters. These paths are also dependent on the antenna switch settings. Incorrectly setting the switches generally results in attenuated input / output power. Receive filters are band pass (series high & low pass filters), transmit filters are low pass.

Source code related to controlling the filter band and antenna switches resides in `e300_impl.c`. Specifically, refer to methods `e300_impl::_update_bandsel`, `e300_impl::_update_atrs`, `e300_impl::_update_gpio`, and `e300_impl::_update_enables`. Generally, these methods set the switches depending on the state of transmit and receive streams.

The following sections provide switch setting tables for antenna and filter selection for frontends A & B receive and transmit paths. For futher details refer to the schematics.

31.3.6

Note: X = don't care, T = If full duplex, set bits according to transmit table, otherwise don't care. Filter range A ? B will be selected if A <= freq < B.

Receive

RX Port	RX Filter (MHz)	VCTXRX2_V1,V2	VCRX2_V1,V2	RX2_BANDSEL[2:0]	RX2B_BANDSEL[1:0]	RX2C_BANDSEL[1:0]
TRX-A	< 450	01	10	101	XX	01
TRX-A	450 – 700	01	10	011	XX	11
TRX-A	700 – 1200	01	10	001	XX	10
TRX-A	1200 – 1800	01	10	000	01	XX
TRX-A	1800 – 2350	01	10	010	11	XX
TRX-A	2350 – 2600	01	10	100	10	XX
TRX-A	2600 – 6000	01	01	XXX	XX	XX
RX2-A	70 – 450	TT	01	101	XX	01
RX2-A	450 – 700	TT	01	011	XX	11
RX2-A	700 – 1200	TT	01	001	XX	10
RX2-A	1200 – 1800	TT	01	000	01	XX
RX2-A	1800 – 2350	TT	01	010	11	XX
RX2-A	2350 – 2600	TT	01	100	10	XX
RX2-A	>= 2600	TT	10	XXX	XX	XX

Transmit

TX Port	TX Filter (MHz)	VCTXRX2_V1,V2	TX_ENABLE2A,2B	TX_BANDSEL[2:0]
TRX-A	< 117.7	10	01	111
TRX-A	117.7 – 178.2	10	01	110
TRX-A	178.2 – 284.3	10	01	101
TRX-A	284.3 – 453.7	10	01	100
TRX-A	453.7 – 723.8	10	01	011
TRX-A	723.8 – 1154.9	10	01	010
TRX-A	1154.9 – 1842.6	10	01	001
TRX-A	1842.6 – 2940.0	10	01	000
TRX-A	>= 2940.0	11	10	XXX

Note: Although the transmit filters are low pass, this table describes UHD's tuning range for selecting each filter path. The table also includes the required transmit enable state.

31.3.7

Note: X = don't care, T = If full duplex, set bits according to transmit table, otherwise don't care. Filter range A ? B will be selected if A <= freq < B.

Receive

RX Port	RX Filter (MHz)	VCTXRX1_V1,V2	VCRX1_V1,V2	RX1_BANDSEL[2:0]	RX1B_BANDSEL[1:0]	RX1C_BANDSEL[1:0]
TRX-B	< 450	10	01	100	XX	10
TRX-B	450 – 700	10	01	010	XX	11
TRX-B	700 – 1200	10	01	000	XX	01
TRX-B	1200 – 1800	10	01	001	10	XX
TRX-B	1800 – 2350	10	01	011	11	XX
TRX-B	2350 – 2600	10	01	101	01	XX
TRX-B	2600 – 6000	10	10	XXX	XX	XX
RX2-B	70 – 450	TT	10	100	XX	10
RX2-B	450 – 700	TT	10	010	XX	11
RX2-B	700 – 1200	TT	10	000	XX	01
RX2-B	1200 – 1800	TT	10	001	10	XX
RX2-B	1800 – 2350	TT	10	011	11	XX
RX2-B	2350 – 2600	TT	10	101	01	XX
RX2-B	>= 2600	TT	01	XXX	XX	XX

Transmit

TX Port	TX Filter (MHz)	VCTXRX1_V1,V2	TX_ENABLE1A,1B	TX1_BANDSEL[2:0]
TRX-B	< 117.7	00	01	111
TRX-B	117.7 – 178.2	00	01	110
TRX-B	178.2 – 284.3	00	01	101
TRX-B	284.3 – 453.7	00	01	100
TRX-B	453.7 – 723.8	00	01	011
TRX-B	723.8 – 1154.9	00	01	010
TRX-B	1154.9 – 1842.6	00	01	001
TRX-B	1842.6 – 2940.0	00	01	000
TRX-B	>= 2940.0	11	10	XXX

Note: Although the transmit filters are low pass, the following table describes UHD's tuning range for selecting each filter path. The table also includes the required transmit enable states.

31.4

31.4.1

- SSB/LO Suppression -35/50 dBc
- Phase Noise 3.5 GHz 1.0 deg RMS
- Phase Noise 6 GHz 1.5 deg RMS
- Power Output >10dBm
- IIP3 (@ typ NF) -20dBm
- Typical Noise Figure <8dB

31.4.2

- The maximum input power for the E310/E312/E313 is 0 dBm.

31.5

- Ettus Research recommends to always use the latest stable version of UHD
- Required version on the host computer must match what is running on the E31x

31.5.1

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

31.5.2

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.5

31.5.3

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

31.6

31.6.1

31.6.1.1

- 133 x 68 x 26.4 mm

31.6.1.2

- 133 x 68.2 x 31.8 mm

31.6.1.3

- 186 x 280 x 106 mm

31.6.2

31.6.2.1

- 375 g

31.6.2.2

- 446 g

31.6.2.3

- 2.5 kg

31.6.3

31.6.3.1

- Media:E310_Dimensional_Sketches.pdf
- Media:cu e310 motherboard cca.pdf
- Media:cu E310 daughtercard cca.pdf
- Media:cu usrp-e310.pdf

31.6.3.2

- Media:cu e312 motherboard cca.pdf
- Media:cu e312 daughtercard cca.pdf
- Media:cu ettus-e312.pdf

31.6.3.3

- Media:cu usrp-e313 enclosure.pdf
- Media:USRP E313 Dimension Pole Mount.pdf
- Media:USRP E313 Dimension Surface Mount.pdf
- Media:USRP_E313_Mounting_Accessory_Assembly_Guide.pdf
- Media:USRP E313 USB conduit interface.png

31.6.4

31.6.4.1

- Motherboard

31.6.4.2

- Top Cover
- Rear Plate
- Front Plate
- Middle Frame
- Bottom Cover

31.6.4.3

- Motherboard
- Daughtercard
- Enclosure

31.6.4.4

- Enclosure

31.7

31.7.1

- E310 0-40 °C
- E312 0-40 °C
- E313 -40-71 °C

31.7.2

- 10% to 90% non-condensing

31.8

31.8.1

E310 Schematics

E310 DB

E310 Architecture

31.9

Part Number	Description	Schematic ID (Page)
	Motherboard	
TXS02612RTWR	SDIO PORT EXPANDER	U23 (2)
XC7Z020-1CLG484CES9919	FPGA	U11 (2,3,4,8,11,13)
Xilinx Zynq Product Page	FPGA	-
USB3340-EZK-TR	ULPI Transceiver	U33 (5)
AK4571VQP	Audio CODEC	U30 (6)
FT230XQ-R	UART Interface	U32 (6)
88E1512	Gigabit Ethernet Transceiver	U13 (7)
24LC024/SN	EEPROM	U5 (9)
DS1339,SM	Real-Time Clock	U6 (9)
ADT7408	Temperature Sensor	U8 (9)
MPU-9150	Motion Processing Unit	U3 (9)
InvenSense MPU-9150 Product Page	Motion Processing Unit	U3 (9)
BMP180	Digital pressure sensor	U4 (9)
BQ24192	Adapter Charger	U1 (10)
TPS54478	Step-Down Switcher	U20 (10)
MAX6510HAUT-T	Temperature Switches	U35 (10)
ATTINY88-MU	Microcontroller	U18 (10)
TPS61253YFF	Step-Up Converter	U19 (10)
AMY-6M	GPS Module	U12 (6)
525L20DA40M0000	VCTCXO	-
	Daughterboard	
Part Number	**Description**	**Schematic ID (Page)**
AD9361 Product Page	2 x 2 RF Agile Transceiver	U8 (3)
24AA256	EEPROM	U15 (2)
TC-1-43A+	RF Transformer	T6 (3); T5 (3); T4 (3)
TC1-1-13M+	RF Transformer	T7 (3); T10 (3); T1 (3)
TPS62140	Step-Down Converter	U19 (4)
ADP1753ACPZ-R7	Linear Regulator	U17 (4); U18 (4)
SGA-4563Z	MMIC AMPLIFIER	U12 (5); U4 (5)
SKY13418-485LF	Antenna Switch	U13 (5); U3 (5); U16 (5); U2 (5); U10 (6); U5 (6)
SKY13373-460LF	SP3T Switch	U11 (6); U9 (6); U6 (6); U7 (6); SW4 (7); SW1 (7)
MGA-81563	Amplifier	U14 (5); U1 (5)
LFCN-5850+	Low Pass Filter	FL32 (5); FL1 (5)
LFCN-2750+	Low Pass Filter	FL37 (5); FL4 (5)
LFCN-2250+	Low Pass Filter	FL23 (6); FL20 (6)
LFCN-1700+	Low Pass Filter	FL40 (5); FL2 (5)
LFCN-1575+	Low Pass Filter	FL25 (6); FL17 (6)
LFCN-1000+	Low Pass Filter	FL33 (5); FL9 (5); FL27 (6); FL15 (6)
LFCN-575+	Low Pass Filter	FL36 (5); FL5 (5)
LFCN-530+	Low Pass Filter	FL29 (6); FL13 (6)
LFCN-400+	Low Pass Filter	FL38 (5); FL3 (5); FL30 (6); FL11 (6)
LFCN-225	Low Pass Filter	FL39 (5); FL6 (5)
LFCN-160+	Low Pass Filter	FL34 (5); FL8 (5)
LFCN-80+	Low Pass Filter	FL35 (5); FL7 (5)
HFCN-1600+	High Pass Filter	FL22 (6); FL19 (6)
HFCN-1100+	High Pass Filter	FL24 (6); FL16 (6)
HFCN-650+	High Pass Filter	FL26 (6); FL14 (6)
HFCN-440+	High Pass Filter	FL28 (6); FL12 (6)
BFCN-2435+	Bandpass Filter	FL21 (6); FL18 (6)
FDG6301N	Dual N-Channel, Digital FET	Q8 (7); Q5 (7)
HSMS-8202	Mixer Diodes	CR1 (7); CR2 (7); CR3 (7); CR4 (7)
LP5900TL	Linear Regulator	U25 (8)
ADP150AUJZ-3.0	Linear Regulator	U22 (8)

AD5662RBJ	16-Bit nanoDAC	U21 (8)
SN74AUP1T57	Voltage Translator	U27 (8); U28 (8); U29 (8)

Request a detailed whitepaper covering features and components from info@ettus.com

31.10

- Utilization statistics are subject to change between UHD releases. This information is current as of UHD 3.9.4 and was taken directly from Xilinx Vivado 2014.4. However, keep in mind that Vivado 2015.4 is recommended for FPGA design involving this device.

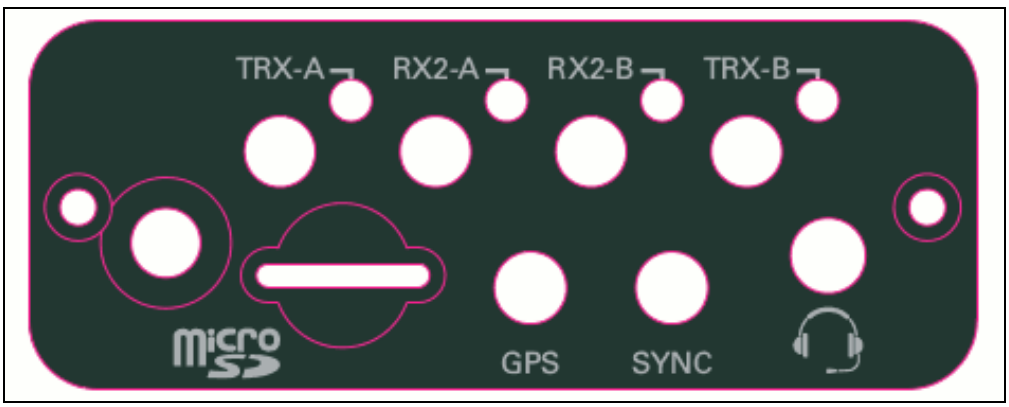
31.10.1

```
1. Slice Logic
--------------

+----------------------------+-------+-----------+-------+
|          Site Type         |  Used | Available | Util% |
+----------------------------+-------+-----------+-------+
| Slice LUTs                 | 36203 |     53200 | 68.05 |
|   LUT as Logic             | 28108 |     53200 | 52.83 |
|   LUT as Memory            |  8095 |     17400 | 46.52 |
|     LUT as Distributed RAM |   870 |           |       |
|     LUT as Shift Register  |  7225 |           |       |
| Slice Registers            | 36562 |    106400 | 34.36 |
|   Register as Flip Flop    | 36562 |    106400 | 34.36 |
|   Register as Latch        |     0 |    106400 |  0.00 |
| F7 Muxes                   |   376 |     26600 |  1.41 |
| F8 Muxes                   |   125 |     13300 |  0.93 |
+----------------------------+-------+-----------+-------+

3. Memory
---------

+-------------------+------+-----------+-------+
|     Site Type     | Used | Available | Util% |
+-------------------+------+-----------+-------+
| Block RAM Tile    |   97 |       140 | 69.28 |
|   RAMB36/FIFO*    |   90 |       140 | 64.28 |
|     RAMB36E1 only |   90 |           |       |
|   RAMB18          |   14 |       280 |  5.00 |
|     RAMB18E1 only |   14 |           |       |
+-------------------+------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a


4. DSP
------

+----------------+------+-----------+-------+
|    Site Type   | Used | Available | Util% |
+----------------+------+-----------+-------+
| DSPs           |  120 |       220 | 54.54 |
|   DSP48E1 only |  120 |           |       |
+----------------+------+-----------+-------+
```

31.11

- 10/100/1000 BASE-T Ethernet
- Stereo audio out, mono mic in
- Integrated GPS receiver
- Host USB support
- 9-axis IMU

31.11.1



- **RF A Group**
  - **TX/RX LED:** Indicates that data is streaming on the TX/RX channel on frontend side A
  - **RX2 LED:** Indicates that data is streaming on the RX2 channel on frontend side A
- **RF B Group**
  - **TX/RX LED:** Indicates that data is streaming on the TX/RX channel on frontend B
  - **RX2 LED:** Indicates that data is streaming on the RX2 channel on frontend B
- **PWR:** Power switch with integrated status LED, for status description see below.
- **SYNC:** Input port for external PPS signal
- **GPS:** Connection for the GPS antenna
- **AUDIO:** Audio input / output

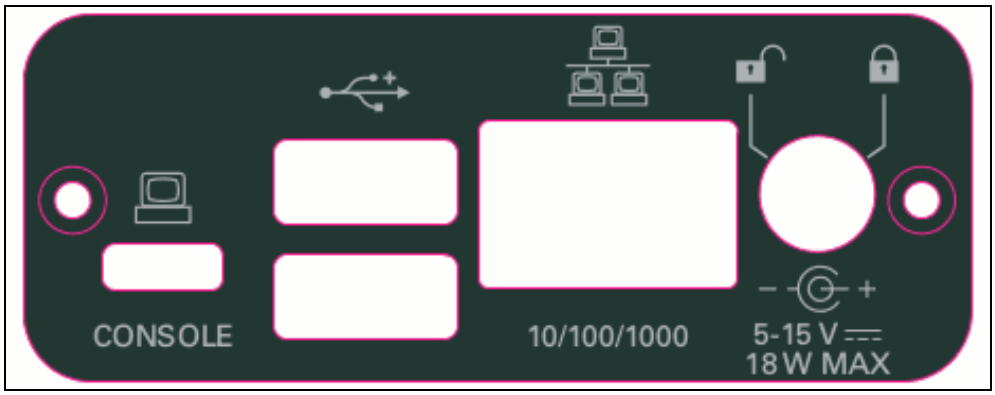
The status LED in the power switch indicates the power and charge status. It's behavior is firmware version dependent.

- **Version 1** (original E310)
  - **Off:** Indicates device is off and not charging
  - **Solid Red:** Indicates device is charging
  - **Solid Green:** Indicates device is on
  - **Fast Blinking Red:** Indicates an error code
    - 1 - Low voltage error
    - 2 - Regulator low voltage error
    - 3 - FPGA power error
    - 4 - DRAM power error
    - 5 - 1.8V rail power error
    - 6 - 3.3V rail power error
    - 7 - Daughterboard / TX power error
    - 9 - Temperature error

- **Version 2** (E312 and upgraded E310)
  - **Off:** Indicates device is off and not charging
  - **Slow Blinking Green:** Indicates device is off and charging
  - **Fast Blinking Green:** Indicates device is on and charging
  - **Solid Green:** Indicates device is on (and not charging, if E312)
  - **Solid Orange:** Indicates device is on and discharging
  - **Fast Blinking Orange:** Indicates device is on, discharging, and charge is below 10% charge
  - **Fast Blinking Red:** Indicates an error code
    - 1 - Low voltage error
    - 2 - Regulator low voltage error
    - 3 - FPGA power error
    - 4 - DRAM power error
    - 5 - 1.8V rail power error
    - 6 - 3.3V rail power error
    - 7 - Daughterboard / TX power error
    - 8 - Charger error
    - 9 - Charger temperature error
    - 10 - Battery low error
    - 11 - Fuel Gauge temperature error
    - 12 - Global (case) temperature error

31.11.2

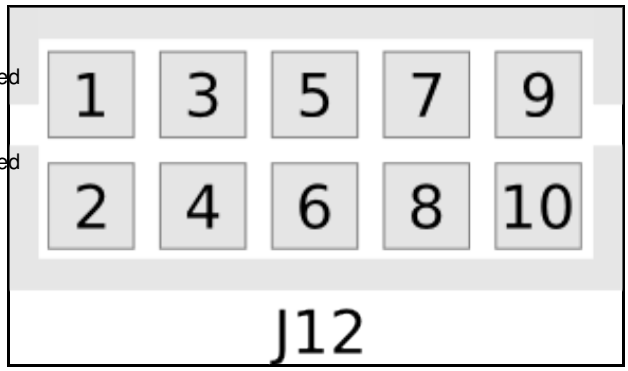
- **PWR:** Locking connector (Kycon KLDHCX-0202-A-LT) for the USRP-E Series power supply
- **1G ETH:** RJ45 port for Ethernet interfaces
- **USB:** USB 2.0 Port
- **SERIAL:** Micro USB connection for serial uart console



31.11.3

Pin Mapping

- Pin 1: +3.3V
- Pin 2: Reserved
- Pin 3: Data[5]
- Pin 4: Reserved
- Pin 5: Data[4]
- Pin 6: Data[0]
- Pin 7: Data[3]
- Pin 8: Data[1]
- Pin 9: 0V
- Pin 10: Data[2]



31.11.3.1

The hardware power on state and UHD initial state for the front-panel GPIOs is high-Z. For the E3xx, there are no external pull-ups/pull-downs for the GPIO pins, but the FPGAs do have them and they are configured as follows: E3xx: pull-down.

- Please see the E3x0/X3x0 GPIO API for information on configuring and using the GPIO bus.

31.11.3.2

Crimp connector pins and sockets can be found at the links below:

- Crimp Connector Pins
- Socket

Complete cable assemblies can be found here:

- https://uk.farnell.com/multicomp/cass-0842/cable-assembly-crimp-socket-150mm/dp/2506397

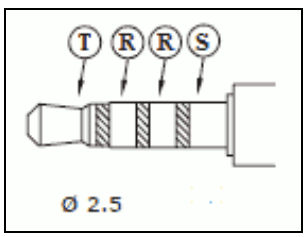
Note: You must modify the the E31x case in order to provide access for the cable to pass through. This modification is not covered under the warranty.

Depending upon your application requirements, a FTDI->UART board such as the Adafruit FT232H board linked below, connected to the USB port provides an option for basic GPIO functionality without any modfiication to the E31x case.

- https://www.adafruit.com/product/2264

31.11.4

- The E3x0 2.5 mm Audio Jack TRRS pins are assigned as follows: Tip=Mic, Ring1=Right, Ring2=Left, Sleeve=GND.
- The Left/Right audio outputs are compatible with typical low-impedance headphones (16 to 32 Ohms). The Microphone pin provides approximately 2 mA bias at 2.2 V when not suspended. A variety of pin configurations can be found on commonly available headsets, so an adapter may be required.



31.12

31.12.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

31.12.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

31.13

Found on the NI Product Certifications lookup tool here.

31.14

- http://files.ettus.com/e3xx_images/

This folder linked above contains SD card images and the SDK (OE cross-compiler build environment) for the USRP E31x. There is a manifest file that shows which packages, and which versions, are included in the OE build within each folder.

The "`alpha`", "`beta`", "`e3xx-release-001`", "`e310-release-002`", "`e3xx-release-3`" folders contain older versions which are currently obsolete. We do not suggest that customers use these files. These versions are no longer supported. They are provided here for archival purposes only.

The current version is Release 4, which located in the "`e3xx-release-4`" folder. We recommend the customers use this version. It is fine if you are already successfully using an older version, but at some point it is recommended that you upgrade to this current version so that you benefit from the latest bug fixes, new features, stability improvements, and other enhancements.

The Release 4 image includes UHD 3.9.2 and GNU Radio 3.7.9, and also includes the corresponding FPGA image file.

Note: An 8 GB SD card is required for the Release 4 image.

The SD card image contains both the FPGA image and the OS for the E31x. The FPGA images are located in the file system of the E31x in the `/usr/local/share/uhd/images` folder.

There are two SD card image files for each version of the image, which include the text "`-dev`" and "`-demo`" in the filename. The "`-dev`" flavor lacks some graphical packages, such as X Windows and QT, which the "`-demo`" flavor includes. The two flavors are otherwise functionally equivalent, although the "`-demo`" flavor takes some additional space on the SD card and some additional memory to run.

The Release 4 image comes in two varieties. The variety that you will need depends on the product number of your E31x, which is printed on the bottom of the device.

For the E310, the product number will be `156333X-01L`, where X is a letter from A to Z. For devices where X is A, B, C, D, the images under the "`ettus-e3xx-sg1`" folder should be used. For devices where X is E or later, the images under the "`ettus-e3xx-sg3`" folder should be used. You must use the appropriate image for your specific device. The incorrect image will not work, and will only boot as far as the U-Boot boot loader before stopping.

For the E312, the product number will be `140605X-01L`, where X is a letter from A to Z. The images under the "`ettus-e3xx-sg3`" folder should be used for all E312 devices.

You can burn the image to an SD card using either the "`dd`" or the "`bmaptool`" tool. Instructions on how to use these tools are at the links below.

- https://files.ettus.com/manual_archive/release_003_009_007/html/page_usrp_e3x0.html#e3x0_upgrade_sd_card
- https://gnuradio.org/redmine/projects/gnuradio/wiki/Copy_an_image_file_to_the_SD_card

The SD image files have an *.xz extension, as they are compressed using the LZMA/LZMA2 compression algorithms. You can uncompress these files with tools such as 7-Zip and the XZ Utils. Please see the links below for further information.

7-Zip

- http://www.7-zip.org/
- https://en.wikipedia.org/wiki/7-Zip

XZ Utils

- http://tukaani.org/xz/
- https://en.wikipedia.org/wiki/XZ_Utils

The folder structure is listed below.

```
.
|-- alpha
|   |-- dizzy-test
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|   |   |-- sdimage-gnuradio-demo.direct.xz
|   |   `-- sdimage-gnuradio-dev.direct.xz
|   |-- fido-rfnoc-test
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|   |   |-- sdimage-gnuradio-demo.direct.xz
|   |   |-- sdimage-gnuradio-demo.direct.xz.md5
|   |   |-- sdimage-gnuradio-dev.direct.xz
|   |   `-- sdimage-gnuradio-dev.direct.xz.md5
|   |-- fido-test
|   |   |-- ettus-e3xx-sg1
|   |   |   |-- sdimage-gnuradio-demo.direct.xz
|   |   |   |-- sdimage-gnuradio-demo.direct.xz.md5
|   |   |   |-- sdimage-gnuradio-dev.direct.xz
|   |   |   `-- sdimage-gnuradio-dev.direct.xz.md5
|   |   |-- ettus-e3xx-sg3
|   |   |   |-- sdimage-gnuradio-demo.direct.xz
|   |   |   |-- sdimage-gnuradio-demo.direct.xz.md5
```

```
|   |   |   |-- sdimage-gnuradio-dev.direct.xz
|   |   |   `-- sdimage-gnuradio-dev.direct.xz.md5
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
|   |   `-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|   `-- fosphor-testing
|       |-- fosphor.direct.xz
|       |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.host.manifest
|       |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|       |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.target.manifest
|       |-- sdimage-gnuradio-demo.direct.xz
|       |-- sdimage-gnuradio-demo.direct.xz.md5
|       |-- sdimage-gnuradio-dev.direct.xz
|       `-- sdimage-gnuradio-dev.direct.xz.md5
|-- beta
|   |-- dizzy-test
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
|   |   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|   |   |-- sdimage-gnuradio-demo.direct.xz
|   |   `-- sdimage-gnuradio-dev.direct.xz
|   `-- dizzy-test-wifi
|       `-- sdimage-gnuradio-dev.direct.xz
|-- e310-release-002
|   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
|   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|   |-- sdimage-gnuradio-demo.direct.xz
|   |-- sdimage-gnuradio-demo.direct.xz.md5sum
|   |-- sdimage-gnuradio-dev.direct.xz
|   `-- sdimage-gnuradio-dev.direct.xz.md5sum
|-- e3xx-release-001
|   |-- e300-gnuradio-dev-image-release1.bz
|   `-- nodistro-eglibc-x86_64-gnuradio-dev-image-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|-- e3xx-release-3
|   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
|   |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
|   |-- sdimage-gnuradio-demo.direct.xz
|   `-- sdimage-gnuradio-dev.direct.xz
`-- e3xx-release-4
    |-- ettus-e3xx-sg1
    |   |-- sdimage-gnuradio-demo.direct.xz
    |   |-- sdimage-gnuradio-demo.direct.xz.md5
    |   |-- sdimage-gnuradio-dev.direct.xz
    |   `-- sdimage-gnuradio-dev.direct.xz.md5
    |-- ettus-e3xx-sg3
    |   |-- sdimage-gnuradio-demo.direct.xz
    |   |-- sdimage-gnuradio-demo.direct.xz.md5
    |   |-- sdimage-gnuradio-dev.direct.xz
    |   `-- sdimage-gnuradio-dev.direct.xz.md5
    |-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.manifest
    `-- oecore-x86_64-armv7ahf-vfp-neon-toolchain-nodistro.0.sh
```

31.15

Below are instructions on setting up a USB WiFi adapter with the E3xx. We have tested the "Edimax EW-7811Un", however many USB based WiFi adapters should be supported.

First verify the USB WiFI adapter is found by running `lsusb`.

Example output from `lsusb` for the "Edimax EW-7811Un" WiFi adapter:

```
Bus 001 Device 003: ID 7392:7811 Edimax Technology Co., Ltd EW-7811Un 802.11n Wireless Adapter [Realtek RTL8188CUS]
```

If the USB adapter is found, proceed with installation and configuration procedure:

1. Run the command below. Enter the passphrase and hit `<Enter>`.

```
wpa_passphrase <SSID> >> /etc/wpa_supplicant.conf`
```

2. In the file `/etc/wpa_supplicant.conf`, edit the entry created above to look like the text below (just add what is missing).

```
network={
       ssid="YOUR_SSID"
       psk=HASH_VALUE
       key_mgmt=WPA-PSK
       proto=RSN WPA
       pairwise=CCMP TKIP
       group=CCMP TKIP
}
```

3. Run the command:

```
wpa_supplicant -B -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf
```

4. Run the command:

```
udhcpc -i wlan0
```

31.15.1

At this time, Ettus Research does not support compiling drivers on the E3xx device. In the Release-4 image there is the drivers for several wifi adapters built-in. Most wireless adapters labeled for the Raspberry-Pi should work with the included drivers.

31.16

31.16.1

- Software Development on the E310 and E312
- Resolving Audio Codec Enumeration Issues On The E31x

31.16.2

- Compass Heading Using Magnetometers (Honeywell Application Note)

31.17

FPGA Images

FPGA Images Read Me

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

31.18

31.18.1

Q: What data rate is supported for continuous sample streaming to a desktop or laptop host-PC? A: Unlike host-based SDRs such as the USRP B, N, and X Series devices, the USRP E Series devices are not intended for continuous streaming of high bandwidth data to a desktop or laptop host-PC. The SDR application runs on an embedded CPU with limited processing capability. Users should leverage the FPGA using tools such as RFNoC to offload compute intensive algorithms that process high bandwidth samples.

Q: What data rate is supported between FPGA to the ARM processor? A: Due to the limited performance of the embedded processor, the maximum data rate from the FPGA to ARM cores is approximately 10 MS/s, and will vary with the processing load on the CPU. The AD9631 RFIC is capable of capturing 56 MHz of bandwidth. Processing signals at full bandwidth requires implementing algorithms on the FPGA.

Q: What is the purpose of the 1 GbE port? A: The E Series can run a DHCP client on the 1 GigE interface to enable connection to a larger network for remote access by another computer. Power over Ethernet is also supported on the E313.

31.18.2

Q: Which environmental tests were performed? A: The USRP E313 is tested against multiple environmental standards to ensure operation in outdoor conditions. These test include ingress protection, temperature, humidity, mechanical shock, random vibration, and altitude. Detailed specifications are provided in the [link data sheet].

Q: How do I protect external devices connected to the host USB port? ?

A: A circular conduit interface is provided in the kit. This component has large and small threaded ends. The small end connects to the USB port on the USRP E313. The user will need to connect their own waterproof structure to the large end to protect the USB device. Since requirements for external devices vary greatly, the conduit interface serves as a flexible starting point for users to build their own solution. See the Media:USRP E313 Dimension Pole Mount.pdf of the conduit interface for details.

Q: How do I protect unused ports? A: All ports come with protective end caps that should be left in place on unused ports.

Q: Why is a DC power supply not included? A: The RJ45 port supports Power over Ethernet and is intended as the primary power supply option. However, users can still use a DC power supply. Since requirements for supplying DC power in outdoor scenarios can vary greatly, a waterproof sleeve for the DC port is provided as a starting point for users to design their own solution. A waterproof sleeve for the RJ45 port is also provided.

Q: Do the RF inputs have lightning protection? A: The DC and PoE power inputs have surge and lightning protection, but the N-type RX/TX inputs and SMA GPS input do not. Users should design antenna lightning protection based on their application requirements.

32 E320

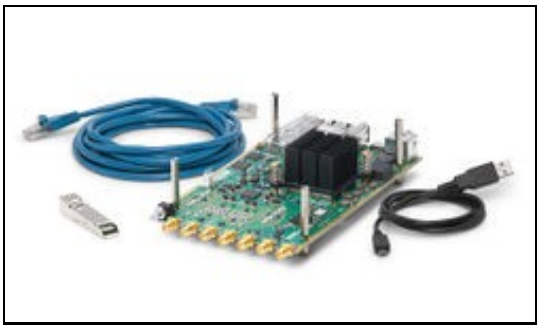
32.1

The USRP E320 brings performance to embedded software defined radios by offering four times more FPGA resources compared to the USRP E31x devices. The USRP E320 also introduces improvements in streaming, synchronization, integration, fault-recovery, and remote management capability. This field deployable SDR continues to use the flexible 2x2 MIMO AD9361 transceiver from Analog Devices, which covers frequencies from 70 MHz ? 6 GHz and provides up to 56 MHz of instantaneous bandwidth.

32.2

32.2.1

- Xilinx Zynq 7045 SoC
  - 7 Series FPGA with 2GB DDR3 RAM
  - ARM Cortex A9 800 MHz dual-core processor
- Analog Devices AD9361 RFIC direct-conversion transceiver
- Frequency range: 70 MHz - 6 GHz
- Up to 56 MHz of instantaneous bandwidth
- 2x2 MIMO transceiver
- Up to 10 MS/s sample data transfer rate to ARM processor
- Up to 61.44 MS/s sample data transfer to host (10Gb/SFP+)
- RX, TX filter banks
- Integrated GPSDO
- 9-axis inertial measurement unit
- RF Network on Chip (RFNoC?) FPGA development framework support
- Board-only and Full Enclosure Options



32.3

32.3.1

- Power Output >10dBm
- IIP3 (@ typical NF) -20dBm
- Typical Receive Noise Figure <8dB

32.3.2

- The maximum input power for the E320 is -15 dBm.

32.4

- Ettus Research recommends to always use the latest stable version of UHD
- UHD version on the host computer must match what is running on the E320

32.4.1

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.14.0.0

32.5

32.5.1

32.5.1.1

- 173 x 100 x 36 mm

32.5.1.2

- 175 x 106 x 38 mm

32.5.2

32.5.2.1

- 0.16 kg

32.5.2.2

- 0.86 kg

32.5.3

32.5.3.1

- Board only

32.5.3.2

- Enclosure

32.5.4

32.5.4.1

- Board only

32.5.4.2

- Enclosure

32.6

32.6.1

- 0-45 °C

32.6.2

- -40-85 °C

32.6.3

- 10% to 90% non-condensing

32.6.4

- 5% to 95% non-condensing

32.7

- E320: File:Neon Public.pdf

32.8

- Support GPSDO NMEA Strings

32.8.1

You can query the lock status with the `gps_locked` sensor, as well as obtain raw NMEA sentences using the `gps_gprmc`, and `gps_gpgga` sensors. Location information can be parsed out of the `gps_gpgga` sensor by using `gpsd` or another NMEA parser.

32.8.2

	Module Specifications
1 PPS Timing Accuracy from GPS receiver	<8ns to UTC RMS (1-Sigma) GPS Locked
Holdover Stability (1 week with GPS)	<±50us over 3 Hour Period @+25°C (No Motion, No Airflow)
1 PPS Output	3.3VDC CMOS
Serial Port	TTL Level, GPS NMEA Output with 1Hz or 5Hz update rate, Integrated into UHD
GPS Frequency	L1, C/A 1574MHz
GPS Antenna	Active (3V compatible) or Passive (0dB to +30dB gain)
GPS Receiver	65 Channels, QZSS, SBAS WAAS, EGNOS, MSAS capable Supports Position and Hold over-determined clock mode
Sensitivity	Acquisition -148dBm, Tracking -165dBm
TTFF	Cold Start: <32 sec, Warm Start: 1 sec, Hot Start: 1 sec
ADEV	10s: <7E-011 10Ks: <2E-012 (GPS Locked, 25°C, no motion, no airflow)
Warm Up Time / Stabilization Time	<10 min at +25C to 1E-09 Accuracy
Supply Voltage (Vdd)	3.3V Single-Supply, +0.2V/-0.15V
Power Consumption	<0.16W

Operating Temperature	-10°C to +70°C
Storage Temperature	-45C to 85C
Oscillator Specifications (internal)	
Frequency Output of lock	20MHz CMOS 3Vpp
Phase Noise crystal	
20MHz Reference	1E-08 After 1 Hour @ +25°C without GPS
RF Output Amplitude	3Vpp CMOS
20MHz Phase Jitter (100Hz to 10MHz)	<0.135ps rms
Frequency Stability Over Temperature (0°C to +60°C)	+0.1ppm (internal TCXO without GPS)
Warm Up Time	< 1 min at +25C
Phase Noise at 20MHz	
1Hz	-65 dBc/Hz
10Hz	-97 dBc/Hz
100Hz	-116 dBc/Hz
1kHz	-136 dBc/Hz
10kHz	<-148 dBc/Hz
100 kHz	<-155 dBc/Hz

32.8.3

- Spec Sheet: http://www.jackson-labs.com/assets/uploads/main/LTE-Lite_specsheet_20MHz.pdf
- User Manual: http://www.jackson-labs.com/assets/uploads/main/LTE-Lite.pdf

32.9

32.9.1

The RF frontend has individually tunable receive and transmit chains. Both transmit and receive can be used in a MIMO configuration. For the MIMO case, both receive frontends share the RX LO, and both transmit frontends share the TX LO. Each LO is tunable between 50 MHz and 6 GHz.

32.9.2

All frontends have individual analog gain controls. The receive frontends have 76 dB of available gain; and the transmit frontends have 89.8 dB of available gain. Gain settings are application specific, but it is recommended that users consider using at least half of the available gain to get reasonable dynamic range.

32.10

- RJ45 (1 GbE)
- SFP+ (1/10 GbE, Aurora)
- Type A USB Host
- Micro-USB (serial console, JTAG)
- GPIO
- Integrated GPSDO
- 9-axis IMU

32.10.1

- **PWR:** Power button
- **RF A Group**
  - **TX/RX LED:** Indicates that data is streaming on the TX/RX channel on frontend side A
  - **RX2 LED:** Indicates that data is streaming on the RX2 channel on frontend side A
- **RF B Group**
  - **TX/RX LED:** Indicates that data is streaming on the TX/RX channel on frontend B
  - **RX2 LED:** Indicates that data is streaming on the RX2 channel on frontend B
- **PPS IN:** Input port for external PPS signal

- **REF IN:** Input port for external 10 MHz signal
- **GPS ANT:** Connection for the GPS antenna

32.10.2

- **POWER:** 10-14v DC Power connector
- **GPIO:** Mini-HDMI connector for GPIO
- **USB:** USB 2.0 Port
- **1G ETH:** RJ45 port for remote management
- **SFP+:** SFP+ connection for sample streaming
- **CONSOLE JTAG:** Micro USB connection for serial UART/JTAG console

32.10.3

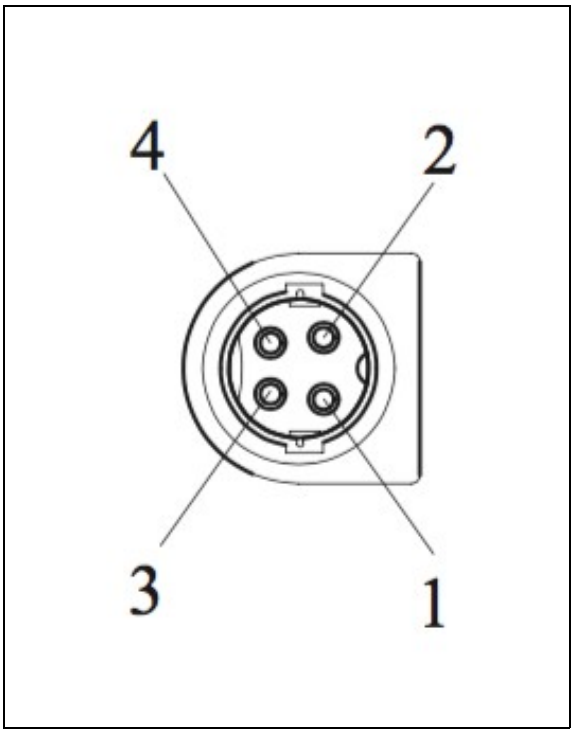
Model: PDP-40 by CUI Inc.

Power plug connectors for custom power harnesses can be purchased here:
https://www.digikey.com/products/en?KeyWords=CP-7340-ND&WT.z_cid=sp_102_buynow

Assembly instructions: Media:pdp-40.pdf

32.10.3.1

- Pins #1 / #2: 12v
- Pins #3 / #4: Ground



32.11

Ettus Research currently offers direct-connect, copper cabling accessories for the USRP E320. However, it is also possible to use multi-mode fiber instead of copper connections for these devices. In this section, we will provide general guidance on the types of fiber adapters and cables that can be used with these products.

The USRP E320 USRP is compatible with most brands of SFP+ fiber adapters. In some cases, other equipment in the systems such as 1/10 Gigabit Ethernet switches are only compatible with specific brands of SFP+ adapters and cables. As a general rule, we recommend checking compatibility with the switches and network cards in your system before purchasing an adapter.

Ettus Research does test the USRP E320 USRP devices with our 10 Gigabit Ethernet Connectivity Kit and a Blade Networks G8124 1/10 GigE switch. Here are is a list of known-good cables and adapters.

Ettus Research has only tested multi-mode fiber accessories.

32.11.1

- Approved Optics BN-CKM-SP-SR-A

32.11.2

- Elpeus 10GbE SFP+ AOC Cable, 3 meters

32.12

Many new motherboards come equipped with an onboard 10Gb RJ45 NIC. It is possible to use a SFP+ to RJ45 adapter and operate at 10Gb speeds using a Cat6/7 Ethernet cables.

Ettus Research has tested the adapters linked below.

32.12.1

- 10Gtek SFP+ to RJ45 Copper Module
- ProLabs 10G-SFPP-T-C

32.13

32.13.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

32.13.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

32.14

Found on the NI Product Certifications lookup tool here.

33 N210

33.1

The USRP Network Series offers high-bandwidth, high-dynamic range processing capability. The Gigabit Ethernet interface of the USRP Network Series allows high-speed streaming capability up to 50 MS/s in both directions (8-bit samples). These features, combined with plug-and-play MIMO capability make the USRP Network an ideal candidate for software defined radio systems with demanding performance requirements.

33.2

33.2.1

- 50 MHz of RF bandwidth with 8 bit samples
- 25 MHz of RF bandwidth with 16 bit samples
- Gigabit Ethernet connectivity
- MIMO capable - requires two or more USRP N200 devices as motherboard has one daughterboard slot (1 RX + 1 TX connectors)
- Onboard FPGA processing
- FPGA: Xilinx® Spartan® 3A-DSP XC3SD1800A
- ADCs: 14-bits 100 MS/s
- DACs: 16-bits 400 MS/s
- Ability to lock to external 5 or 10 MHz clock reference
- TCXO Frequency Reference (~2.5ppm)
- Optional internal GPS locked reference oscillator
- FPGA code can be changed with Xilinx® ISE® WebPACK? tools
- Frequency range: DC - 6 GHz with suitable daughterboard

33.2.2

- 50 MHz of RF bandwidth with 8 bit samples
- 25 MHz of RF bandwidth with 16 bit samples
- Gigabit Ethernet connectivity
- MIMO capable - requires two or more USRP N210 devices as motherboard has one daughterboard slot (1 RX + 1 TX connectors)
- Onboard FPGA processing
- FPGA: Xilinx® Spartan® 3A-DSP XC3SD3400A
- ADCs: 14-bits 100 MS/s
- DACs: 16-bits 400 MS/s
- Ability to lock to external 5 or 10 MHz clock reference
- TCXO Frequency Reference (~2.5ppm)
- Optional internal GPS locked reference oscillator
- FPGA code can only be changed with the paid version of the Xilinx® ISE® Design Suite tools
- Frequency range: DC - 6 GHz with suitable daughterboard

33.3

- SBX-40
- UBX-40
- WBX-40
- CBX-40
- LFRX / LFTX
- BasicRX / BasicTX
- DBSRX2 (EOL)
- RFX Series (EOL)
- TVRX2 (EOL)

33.4

33.4.1

- SSB/LO Suppression -35/50 dBc
- Phase Noise 1.8 GHz 10kHz -80 dBc/Hz
- Phase Noise 1.8 GHz 100kHz -100 dBc/Hz
- Phase Noise 1.8 GHz 1MHz -137 dBc/Hz
- Power Output 15 dBm
- IIP3 (@ typ NF) 0 dBm
- Typical Noise Figure 5 dB

33.5

- Ettus Research recommends to always use the latest stable version of UHD

33.5.1

- Current Hardware Revision: 4
- Minimum version of UHD required: 3.8.0

33.5.2

- Current Hardware Revision: 4
- Minimum version of UHD required: 3.8.0

33.6

33.6.1

22 x 16 x 5 cm

33.6.2

1.2 kg

33.6.3

- File:cu usrp-n2x0 motherboard.pdf
- File:cu ettus-usrp-n2x0.pdf

33.6.4

33.6.4.1

- Motherboard

33.6.4.2

- Enclosure

33.7

33.7.1

- N200/N210: 25 °C

33.7.2

- 10% to 90% non-condensing

33.8

33.8.1

N200/N210 Schematics

33.9

Part Number	Description	Schematic ID (Page)
AD9777	Dual Channel, 16-Bit DAC	U3 (1)
ADS62P4X	Dual Channel, 14-Bit ADC	U2 (1)
XC3SD3400AFG676	FPGA	U1 (2,8,9,10,11,12)
AD9510	Clock Distribution IC	U9 (4)
ET1011C2	Gigabit Ethernet Transceiver	U12 (6)
CY7C1354C	Pipelined SRAM	U19 (7)
MAX232	Drivers/Receiver	U25 (10)

33.10

- Utilization statistics are subject to change between UHD releases. This information is current as of UHD 3.9.4 and was taken directly from Xilinx Vivado 2014.4.

33.10.1

```
Device utilization summary:
---------------------------

Selected Device : 3sd1800afg676-5

 Number of Slices:                    18356  out of  16640   110% (*)
 Number of Slice Flip Flops:          20466  out of  33280    61%
 Number of 4 input LUTs:              32968  out of  33280    99%
    Number used as logic:             28511
    Number used as Shift registers:    3945
    Number used as RAMs:                512
 Number of IOs:                         338
 Number of bonded IOBs:                 331  out of    519    63%
    IOB Flip Flops:                     342
 Number of BRAMs:                        41  out of     84    48%
 Number of GCLKs:                         6  out of     24    25%
 Number of DCMs:                          1  out of      8    12%
 Number of DSP48s:                       31  out of     84    36%
```

33.10.2

```
Device utilization summary:
---------------------------

Selected Device : 3sd3400afg676-5

 Number of Slices:                    18349  out of  23872    76%
 Number of Slice Flip Flops:          20475  out of  47744    42%
 Number of 4 input LUTs:              32986  out of  47744    69%
    Number used as logic:             28529
    Number used as Shift registers:    3945
    Number used as RAMs:                512
 Number of IOs:                         338
 Number of bonded IOBs:                 331  out of    469    70%
    IOB Flip Flops:                     342
 Number of BRAMs:                       41  out of    126    32%
 Number of GCLKs:                        6  out of     24    25%
 Number of DCMs:                         1  out of      8    12%
 Number of DSP48s:                      31  out of    126    24%
```

33.11

33.11.1

- Gigabit Ethernet

33.12

33.12.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

33.12.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

33.13

Found on the NI Product Certifications lookup tool here.

33.14

For a detailed guide to recovering the N200/N210, please see the N200/N210 Device Recovery application note.

33.15

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

34 N310

34.1

When you receive a brand-new device, it is strongly recommended that you download the most recent filesystem image from the Ettus Research website and write it to the SD card that comes with the unit. It is not recommended that you use the SD card from the factory as-is. Instructions on downloading the latest filesystem image and writing it to the SD card are listed below.

Note that if you are operating the device in Network Mode, then the versions of UHD running on the host computer and on the USRP N300/N310 device must match.

34.2

The USRP N310 is a networked software defined radio that provides reliability and fault-tolerance for deployment in large scale and distributed wireless systems. This device simplifies control and management of a network of radios by introducing the unique capability to remotely perform tasks such as debugging, updating software, rebooting, factory resetting, self-testing, and monitoring system health. The USRP N310 is an all-in-one device that includes two AD9371 transceivers, the Zynq-7100 SoC baseband processor, two SFP+ ports, a built-in GPSDO module, and various other peripheral and synchronization features.

34.3

34.3.1

- Xilinx Zynq-7035 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 2 RX, 2TX in half-wide RU form factor
- 10 MHz ? 6 GHz extended frequency range
- Up to 100 MHz of instantaneous bandwidth per channel
- RX, TX filter bank
- 16 bit ADC, 14 bit DAC
- Configurable sample rates: 122.88, 125, and 153.6 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Watchdog timer
- OpenEmbedded Linux
- High channel density
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation
- USRP N300 **does not** contain a Trusted Platform Module

34.3.2

- Xilinx Zynq-7100 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 4 RX, 4TX in half-wide RU form factor
- 10 MHz ? 6 GHz extended frequency range
- Up to 100 MHz of instantaneous bandwidth per channel
- RX, TX filter bank
- 16 bit ADC, 14 bit DAC
- Configurable sample rates: 122.88, 125, and 153.6 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora)

- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- External RX, TX LO input ports
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Trusted Platform Module (TPM) v1.2
- Watchdog timer
- OpenEmbedded Linux
- High channel density
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation

34.4

34.4.1

- Number of channels: 4
- Frequency Range: 10 MHz to 6 GHz
- Maximum instantaneous bandwidth: 100 MHz
- Minimum frequency step
    - 7.32 Hz @ 122.88 MHz sample rate
    - 7.45 Hz @ 125 MHz sample rate
    - 9.15 Hz @ 153.6 MHz sample rate

- Maximum output power (P? out?): See Table 1
- Gain range
    - -30 dB to 25 dB (10 MHz to 300 MHz)
    - -30 dB to 20 dB (300 MHz to 6 GHz)

- Gain step: 1 dB
- Supported I/Q sample rates:
    - 122.88 MHz, 125 MHz, 153.6 MHz

- Spurious-free dynamic range (SFDR) > 50 dBc
- Output third-order intercept (OIP3) See Table 2

Frequency	Maximum Output Power
10 MHz to 500 MHz	+16 dBm
500 MHz to 1 GHz	+18 dBm
1 GHz to 4 GHz	+18 dBm
4 GHz to 6 GHz	+12 dBm

Table 1: Maximum Output Power

Frequency	Output Third-Order Intercept (IP3)
10 MHz to 2 GHz	> 30 dBm
2 GHz to 4 GHz	> 20 dBm
4 GHz to 6 GHz	> 10 dBm

Table 2: Third-Order Intercept (IP3)

34.4.2

- Number of channels: 4
- Frequency Range: 10 MHz to 6 GHz
- Maximum instantaneous bandwidth: 100 MHz
- Minimum frequency step
    - 7.32 Hz @ 122.88 MHz sample rate
    - 7.45 Hz @ 125 MHz sample rate
    - 9.15 Hz @ 153.6 MHz sample rate

- Gain step: 1
- Maximum recommended input power (P? in?) 1 dB: -15 dBm

- Noise figure: See Table 3
- Spurious-free dynamic range (SFDR): > 50 dBc

- Third-order intermodulation distortion (IMD3) See Table 4

- Supported I/Q sample rates
    - 122.88 MHz, 125 MHz, 153.6 MHz

Frequency	TX/RX Noise Figure	RX2 Noise Figure
1.8 GHz	6.8 dB	5.8 dB
2.4 GHz	7.5 dB	6.5 dB
4.4 GHz	7.0 dB	5.5 dB
5.8 GHz	6.4 dB	6.4 dB

Table 3: Noise Figure

Frequency	RX IMD3
0.5 GHz to 3 GHz	< -80 dBc
3 GHz to 4 GHz	< -74 dBc
4 GHz to 6 GHz	< -81 dBc

Table 4: RX Third-Order Intermodulation Distortion (IMD3)

- Noise figure is measured at maximum gain state on receiver signal path.

34.4.3

- DDR3 Memory size
  - 2,048 MB (PL)
  - 1,024 MB (PS)

34.5

You must use either the Level VI Efficiency power supply provided in the shipping kit, or another UL listed ITE power supply marked ?`LPS`, with the USRP N310.

- Input voltage: 12 VDC
- Input current: 7.0 A, maximum
- Typical power consumption: 50 W to 80 W, varies by application

34.6

- Ettus Research recommends to always use the latest stable version of UHD
- If you need to clean the module, wipe it with a dry towel.

34.6.1

- Current Hardware Revision: D
- Minimum version of UHD required: 3.11.0.0
- Due to product compliance restrictions on products with TPM (Trusted Platform Module) components to a few countries, the USRP N310 is available in two variants:
  - Standard variant with TPM (P/N 785067-01)
  - Non-TPM variant (P/N 786465-01)

34.6.2

There are three master clock rates (MCR) supported on the N310: 122.88 MHz; 125.0 MHz; 153.6 MHz.

The sampling rate must be an integer decimation rate of the MCR. Ideally, this decimation factor should be an even number. An odd decimation factor will result in additional unwanted attenuation (roll-off from the CIC filter in the DUC and DDC blocks in the FPGA). The valid decimation rates are between 1 and 1024.

For the MCR of 122.88 MHz, the achievable sampling rates using an even decimation factor are 122.88, 61.44, 30.72, 20.48, 15.36, 12.288, 10.24, 8.777, 7.68 Msps, ... 120.0 Ksps.

For the MCR of 125.0 MHz, the achievable sampling rates using an even decimation factor are 125.0, 62.5, 31.25, 20.833, 15.625, 12.5, 10.41666, 8.9286 Msps, ... 122.07 Ksps.

For the MCR of 153.6 MHz, the achievable sampling rates using an even decimation factor are 156.3, 78.15, 39.075, 26.05, 19.5375, 15.63, 13.025, 11.16429 Msps, ... 152.637 Ksps

If the desired sampling rate is not directly supported by the hardware, then it will be necessary to re-sample in software. This can be done in C++ using libraries such as Liquid DSP [1], or can be done in GNU Radio, in which there are three blocks that perform sampling rate conversion.

34.7

34.7.1

(L × W × H)

- 35.71 cm × 21.11 cm × 4.37 cm
- 14.06 in. × 8.31 in. × 1.72 in.

34.7.2

- 3.13 kg

34.7.3

34.7.3.1

- File:CU USRP-N300.pdf

34.7.3.2

- File:CU USRP-N310.pdf

34.7.4

If you want any CAD / STP models beyond those found here, please send an email to Ettus Support at support@ettus.com noting your request and your use case for any such model. We will determine on a case-by-case basis whether we have any such requested model and, if so, whether to release it -- possibly requiring an NDA for any such release. Note that we do not have models on all USRPs and daughterboards, and requesting any model does not guarantee that either Ettus Research or NI will honor any such request.

34.7.4.1

- Enclosure Bottom Panel
- Enclosure Top Panel
- Faceplate
- Full Enclosure

34.8

34.8.1

- N310: 0 to 50 °C

34.8.2

- N310: -40 to 70 °C

34.8.3

- 10% to 90% non-condensing

34.9

34.9.1

- Motherboard: File:USRP N310 N300 MB Schematic.pdf
- Daughterboard: File:USRP N310 N300 DB Schematic.pdf

34.10

- Support GPSDO NMEA Strings

34.10.1

You can query the lock status with the `gps_locked` sensor, as well as obtain raw NMEA sentences using the `gps_gprmc`, and `gps_gpgga` sensors. Location information can be parsed out of the `gps_gpgga` sensor by using `gpsd` or another NMEA parser.

34.10.2

	Module Specifications
1 PPS Timing Accuracy from GPS receiver	<8ns to UTC RMS (1-Sigma) GPS Locked
Holdover Stability (1 week with GPS)	<±50us over 3 Hour Period @+25°C (No Motion, No Airflow)
1 PPS Output	3.3VDC CMOS
Serial Port	TTL Level, GPS NMEA Output with 1Hz or 5Hz update rate, Integrated into UHD
GPS Frequency	L1, C/A 1574MHz
GPS Antenna	Active (3V compatible) or Passive (0dB to +30dB gain)
GPS Receiver	65 Channels, QZSS, SBAS WAAS, EGNOS, MSAS capable Supports Position and Hold over-determined clock mode
Sensitivity	Acquisition -148dBm, Tracking -165dBm
TTFF	Cold Start: <32 sec, Warm Start: 1 sec, Hot Start: 1 sec
ADEV	10s: <7E-011 10Ks: <2E-012 (GPS Locked, 25°C, no motion, no airflow)
Warm Up Time / Stabilization Time	<10 min at +25C to 1E-09 Accuracy
Supply Voltage (Vdd)	3.3V Single-Supply, +0.2V/-0.15V
Power Consumption	<0.16W
Operating Temperature	-10°C to +70°C
Storage Temperature	-45C to 85C
Oscillator Specifications (internal)	

Frequency Output of low Phase Noise crystal	20MHz CMOS 3Vpp
20MHz Reference	±2E-08 After 1 Hour @ +25°C without GPS
RF Output Amplitude	3Vpp CMOS
20MHz Phase Jitter (100Hz to 10MHz)	<0.135ps rms
Frequency Stability Over Temperature (0°C to +60°C)	±0.1ppm (internal TCXO without GPS)
Warm Up Time	≤ 1 min at +25C
Phase Noise at 20MHz	1Hz -65 dBc/Hz 10Hz -97 dBc/Hz 100Hz -116 dBc/Hz 1kHz -136 dBc/Hz 10kHz <-148 dBc/Hz 100 kHz <-155 dBc/Hz

34.10.3

- Spec Sheet: http://www.jackson-labs.com/assets/uploads/main/LTE-Lite_specsheet_20MHz.pdf
- User Manual: http://www.jackson-labs.com/assets/uploads/main/LTE-Lite.pdf

34.11

34.11.1

The Verilog code for the FPGA in the USRP N300/N310 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Specifically, changing the I/O interface of the FPGA in any way, or modifying the pin and timing constraint files, could result in physical damage to other components on the motherboard, external to the FPGA, and doing this will void the warranty. Also, even if the PCIe interface is not being used, you cannot remove or reassign these pins in the constraint file. The constraint files should not be modified. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

34.12

34.12.1

- **PWR**: Power switch
- **RF 0 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 0, channel 0
  - **RX2 SMA/LED**:

RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 0, channel 0

- **RF 1 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 0, channel 1.
  - **RX2 SMA/LED**: RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 0, channel 1.
- **RF 2 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 1, channel 0.
  - **RX2 SMA/LED**: RF Input Port

/ Indicates that data is streaming on the RX2 channel on daughterboard 1, channel 0.

- **RF 3 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 1, channel 1.
  - **RX2 SMA/LED**: RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 1, channel 1.

- **LO IN 0/1**
  - **TX**: Input port for TX LO of Daughterboard 0. The LO input frequency must be twice the frequency of the desired RF output frequency. An LO input

frequency range of 600 MHz to 8000 MHz corresponds to an RF output frequency of 300 MHz to 4000 MHz. LO inputs above 8000 MHz, and RF outputs above 4000 MHz, are not supported. The LO input signal level should be +3 dBm, but may be between 0 dBm and +6 dBm.

- **RX**: Input port for RX LO of Daughterboard 0. The LO input frequency must be twice the frequency of the desired RF output frequency. An LO input frequency range of 600 MHz to 8000 MHz corresponds to

an RF output frequency of 300 MHz to 4000 MHz. LO inputs above 8000 MHz, and RF outputs above 4000 MHz, are not supported. The LO input signal level should be +3 dBm, but may be between 0 dBm and +6 dBm.

- **LO IN 2/3**
  - **TX**: Input port for TX LO of Daughterboard 1. The LO input frequency must be twice the frequency of the desired RF output frequency. An LO input frequency range of 600 MHz to 8000 MHz corresponds to an RF output frequency of 300

MHz
to
4000
MHz.
LO
inputs
above
8000
MHz,
and
RF
outputs
above
4000
MHz,
are
not
supported.
The
LO
input
signal
level
should
be
+3
dBm,
but
may
be
between
0
dBm
and
+6
dBm.

- **RX**:
Input
port
for
RX
LO
of
Daughterboard
1.
The
LO
input
frequency
must
be
twice
the
frequency
of
the
desired
RF
output
frequency.
An
LO
input
frequency
range
of
600
MHz
to
8000
MHz
corresponds
to
an
RF
output
frequency
of
300
MHz
to
4000
MHz.
LO
inputs
above
8000
MHz,
and

RF outputs above 4000 MHz, are not supported. The LO input signal level should be +3 dBm, but may be between 0 dBm and +6 dBm.

- **GPIO**
  - **GPIO**: DB15 GPIO Interface. Additional details below.

34.12.2

- **GPS ANT**: Connection for the GPS antenna
- **REF IN**: Reference clock input
- **PPS/TRIG IN**: Input port for the PPS signal
- **TRIG OUT**: Output port for the exported reference clock
- **PWR**: Connector for the USRP N310 Series power supply
- **RESET**: Input button to reset device
- **MicroSD**: MicroSD Card

for OE Linux File System
- **Console JTAG**: Micro USB connector for the on-board USB-JTAG programmer as well as TTY login to the console
- **USB 2.0**: Host USB connector to ARM CPU
- **SFP+**: 1/10Gb SFP+ ports for Ethernet interfaces
- **10/1000/1000**: 10/100/1000 Mb Ethernet interface to ARM CPU

34.12.3

Using an external 10 MHz reference clock, a square wave will offer the best phase noise performance, but a sinusoid is acceptable. The power level of the reference clock cannot exceed +10 dBm.

34.12.4

Using a PPS signal for timestamp synchronization requires a square wave signal (a typical PPS signal has a 20%-25% duty cycle) with a 5Vpp amplitude.

To test the PPS input, you can use the following tool from the UHD examples:

- `<args>` are device address arguments (optional if only one USRP device is on your machine)

```
cd <install-path>/lib/uhd/examples ./test_pps_input ?args=<args>
```

34.12.5

The GPIO port is not meant to drive big loads. You should not try to source more than 5mA per pin.

The +3.3V is for ESD clamping purposes only and not designed to deliver high currents.

34.12.5.1

The hardware power on state and UHD initial state for the front-panel GPIOs is high-Z. For the N310, there are no external pull-ups/pull-downs for the GPIO pins, but the FPGAs do have them and they are configured as follows: N310: pull-down.

34.12.5.2

- Pin 1: +3.3V
- Pin 2: Data[0]
- Pin 3: Data[1]
- Pin 4: Data[2]
- Pin 5: Data[3]
- Pin 6: Data[4]
- Pin 7: Data[5]
- Pin 8: Data[6]
- Pin 9: Data[7]
- Pin 10: Data[8]
- Pin 11: Data[9]

- Pin 12: Data[10]
- Pin 13: Data[11]
- Pin 14: 0V
- Pin 15: 0V

Note: Please see the E3x0/X3x0/N3x0 GPIO API for information on configuring and using the GPIO bus.

34.12.6

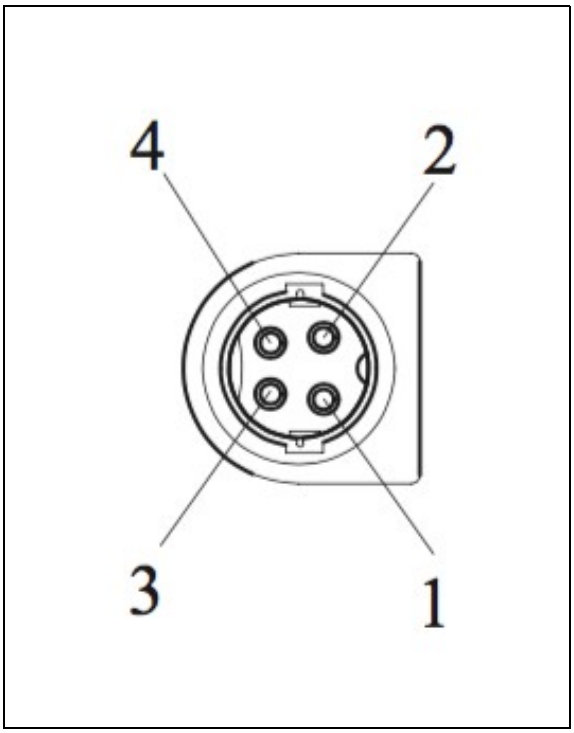
Model: PDP-40 by CUI Inc.

Power plug connectors for custom power harnesses can be purchased here:
https://www.digikey.com/products/en?KeyWords=CP-7340-ND&WT.z_cid=sp_102_buynow

Assembly instructions: Media:pdp-40.pdf

34.12.6.1

- Pins #1 / #2: 12v
- Pins #3 / #4: Ground



34.13

34.13.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at
http://ettus.com/legal/rohs-information

34.13.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

34.14

Found on the NI Product Certifications lookup tool here.

34.15

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

34.16

34.16.1

Recommended 10 Gigabit Ethernet Cards

- Intel X520-DA2
  - Intel® Ethernet Converged Network Adapter X520-DA2
- Intel X520-DA1
  - Intel® Ethernet Converged Network Adapter X520-DA1
- Intel X710-DA2
  - Intel® Ethernet Converged Network Adapter X710-DA2
- Intel X710-DA4
  - Intel® Ethernet Converged Network Adapter X710-DA4
- Mellanox MCX4121A-ACAT
  - Mellanox MCX4121A-ACAT

34.17

The power supply provided with the USRP N310 kit is packaged with a power cord that is compatible with power outlets in the US/Japan. If you are not using the USRP N310 in the US/Japan, we recommend purchasing the International USRP N310 Power Cord set.

34.18

The USRP N300/N310 was designed to be used with a 1U Rackmount Assembly for building high-density MIMO systems in a compact and well-organized setup. This mount *requires* two compatible USRPs, and provides rubber standoffs between the USRPs to avoid both direct contact and surface scratching. If the user will be developing in a laboratory environment or building a high-channel count USRP system, then a 1U Rackmount Assembly is highly recommended. This specific mount is compatible with only the USRP N300, N310, N320, and N321, and allows the integration of up to eight bidirectional RF channels per 1U.

34.19

Ettus Research currently offers direct-connect, copper cabling accessories for the USRP N300/N310. However, it is also possible to use multi-mode fiber instead of copper connections for these devices. In this section, we will provide general guidance on the types of fiber adapters and cables that can be used with these products.

The USRP N300/N310 USRP is compatible with most brands of SFP+ fiber adapters. In some cases, other equipment in the systems such as 1/10 Gigabit Ethernet switches are only compatible with specific brands of SFP+ adapters and cables. As a general rule, we recommend checking compatibility with the switches and network cards in your system before purchasing an adapter.

Ettus Research does test the USRP N300/N310 USRP devices with our 10 Gigabit Ethernet Connectivity Kit and a Blade Networks G8124 1/10 GigE switch. Here are is a list of known-good cables and adapters.

Ettus Research has only tested multi-mode fiber accessories.

34.19.1

- Approved Optics BN-CKM-SP-SR-A

34.19.2

- Elpeus 10GbE SFP+ AOC Cable, 3 meters

34.20

Many new motherboards come equipped with an onboard 10Gb RJ45 NIC. It is possible to use a SFP+ to RJ45 adapter and operate at 10Gb speeds using a Cat6/7 Ethernet cables.

Ettus Research has tested the adapters linked below.

34.20.1

- 10Gtek SFP+ to RJ45 Copper Module
- ProLabs 10G-SFPP-T-C

34.21

As of UHD version 3.14.0.0, the following data rates have been tested and validated.

Number of TX Channels	Number of RX Channels	Maximum Sampling Rate
1	0	153.6 MS/s
0	1	153.6 MS/s
1	1	153.6 MS/s
2	0	125 MS/s
0	2	125 MS/s
2	2	125 MS/s
3	0	62.5 MS/s
0	3	125 MS/s
3	3	62.5 MS/s TX, 125 MS/s RX
4	0	62.5 MS/s
0	4	125 MS/s
4	4	62.5 MS/s TX, 125 MS/s RX

35 N321

35.1

When you receive a brand-new device, it is strongly recommended that you download the most recent filesystem image from the Ettus Research website and write it to the SD card that comes with the unit. It is not recommended that you use the SD card from the factory as-is. Instructions on downloading the latest filesystem image and writing it to the SD card are listed below.

Note that if you are operating the device in Network Mode, then the versions of UHD running on the host computer and on the USRP N320/N321 device must match.

35.2

The USRP N320 is a networked software defined radio that provides reliability and fault-tolerance for deployment in large scale and distributed wireless systems. This is a high performance SDR that uses a unique RF design by Ettus Research to provide 2 RX and 2 TX channels in a half-wide RU form factor. Each channel provides up to 200 MHz of instantaneous bandwidth, and covers a frequency range from 3 MHz to 6 GHz. The baseband processor uses the Xilinx Zynq-7100 SoC to deliver a large user programmable FPGA for real-time, low latency processing and a dual-core ARM CPU for stand-alone operation.Support for 1 GbE, 10 GbE, and Aurora interfaces over two SFP+ ports and 1 QSFP+ port enables high throughput IQ streaming to a host PC or FPGA coprocessor. A flexible synchronization architecture with support for LO sharing for TX and RX, 10 MHz clock reference, PPS time reference, GPSDO, and White Rabbit enables implementation of phase coherent MIMO testbeds. The USRP N320 leverages recent software developments in UHD to simplify control and management of multiple devices over the network with the unique capability to remotely administrate tasks such as debugging, updating software, rebooting, resetting to factory state, and monitoring system health.

35.3

35.3.1

- Xilinx Zynq-7100 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 2 RX, 2 TX in half-wide RU form factor
- 3 MHz ? 6 GHz frequency range
- Up to 200 MHz of instantaneous bandwidth per channel
- Sub-octave RX, TX filter bank
- 14 bit ADC, 16 bit DAC
- Configurable sample rates: 200, 245.76, 250 MS/s
- Two SFP+ ports (1 GbE, 10 GbE, Aurora, White Rabbit)
- One QSFP+ port (2x 10Gb / Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- External RX, TX LO input ports
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Trusted Platform Module (TPM) v1.2
- Watchdog timer
- OpenEmbedded Linux
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation

35.3.2

- Xilinx Zynq-7100 FPGA SoC
- Dual-core ARM A9 800 MHz CPU
- 2 RX, 2 TX in half-wide RU form factor
- 3 MHz ? 6 GHz frequency range
- Up to 200 MHz of instantaneous bandwidth per channel
- Sub-octave RX, TX filter bank
- 14 bit ADC, 16 bit DAC
- Configurable sample rates: 200, 245.76, 250 MS/s

- Two SFP+ ports (1 GbE, 10 GbE, Aurora, White Rabbit)
- One QSFP+ port (2x 10Gb / Aurora)
- RJ45 (1 GbE)
- 10 MHz clock reference
- PPS time reference
- External RX, TX LO input ports
- LO Distribution for up to 128x128 MIMO
- Built-in GPSDO
- 1 Type A USB host port
- 1 micro-USB port (serial console, JTAG)
- Trusted Platform Module (TPM) v1.2
- Watchdog timer
- OpenEmbedded Linux
- Reliable and fault-tolerant deployment
- Remote management capability
- Stand-alone operation

35.4

35.4.1

- Number of channels: 2
- Frequency Range: 3 MHz to 6 GHz
- Maximum instantaneous bandwidth: 200 MHz

- Maximum output power (P? out?): See Table 1
- Gain range
    - 0 dB to 60 dB (1 MHz to 6 GHz)

- Gain step: 1 dB
- Supported I/Q sample rates:
    - 200 MHz, 245.76 MHz, 250 MHz

- Output third-order intercept (OIP3) See Table 2

- Tuning Time: 245 us

- TX/RX Switching Time: 750 ns

- Filter Banks
    - 450 ? 650 MHz
    - 650 ? 1000 MHz
    - 1000 ? 1350 MHz
    - 1350 ? 1900 MHz
    - 1900 ? 3000 MHz
    - 3000 ? 4100 MHz
    - 4100 ? 6000 MHz

- External LO Frequency Range: 450 MHz - 6.0 GHz

Frequency	Maximum Output Power
3 MHz - 450 MHz	+10 dBm
450 MHz - 1000 MHz	+20 dBm
1 GHz - 4.25 GHz	+18 dBm
4.25 GHz - 6 GHz	+15 dBm

Table 1: Maximum Output Power

Frequency	Output Third-Order Intercept (OIP3)
3 MHz - 450 MHz	> 15 dBm
450 MHz - 1.6 GHz	> 28 dBm
1.6 GHz - 5.8 GHz	> 25 dBm
5.8 GHz - 6.0 GHz	> 23 dBm

Table 2: Output Third-Order Intercept (OIP3)

Frequency Offset	1.0 GHz	2.0 GHz	3.0 GHz	5.5 GHz
10 kHz	-117	-110	-108	-103
100 kHz	-117	-110	-108	-104
1 MHz	-145	-137	-135	-130

Table 3: TX Phase Noise (dBc/Hz)

35.4.2

- Number of channels: 2
- Frequency Range: 3 MHz to 6 GHz
- Maximum instantaneous bandwidth: 200 MHz
- Gain range
    - 0 dB to 60 dB (1 MHz to 6 GHz)
- Gain step: 1 dB
- Maximum recommended input power (P? in?) 1 dB: -15 dBm

- Noise figure: See Table 3

- Third-order intermodulation distortion (IMD3) See Table 4

- Supported I/Q sample rates
    - 200 MHz, 245.76 MHz, 250 MHz

- Tuning Time: 245 us

- TX/RX Switching Time: 750 ns

- Filter Banks
    - 450 ? 760 MHz
    - 760 ? 1100 MHz
    - 1100 ? 1410 MHz
    - 1410 ? 2050 MHz
    - 2050? 3000 MHz
    - 3000 ? 4500 MHz
    - 4500 ? 6000 MHz

- External LO Frequency Range: 450 MHz - 6.0 GHz

Frequency	TX/RX Port Noise Figure	RX2 Port Noise Figure
< 800 MHz	11.0 dB	10.0 dB
800 MHz - 1.8 GHz	6.5 dB	5.5 dB
1.8 GHz - 2.8 GHz	7.0 dB	6.0 dB
2.8 GHz - 3.8 GHz	7.5 dB	6.5 dB
3.8 GHz - 5.0 GHz	8.5 dB	7.5 dB
5.0 GHz - 6.0 GHz	11.0 dB	10.0 dB

Table 3: RX Noise Figure

Frequency	RX Input Third-Order Intercept (IIP3) (dBm)
450 MHz - 1.0 GHz	> 13 dBm
1.0 GHz - 4.5 GHz	> 17 dBm
4.5 GHz - 6.0 GHz	> 16 dBm

Table 4: RX Input Third-Order Intercept (IIP3) (dBm)

35.4.3

- DDR3 Memory size
    - 2,048 MB (PL)
    - 1,024 MB (PS)

35.5

You must use either the Level VI Efficiency power supply provided in the shipping kit, or another UL listed ITE power supply marked ?`LPS`, with the USRP N320/N321.

- Input voltage: 12 VDC
- Input current: 7.0 A, maximum
- Typical power consumption: 60 W to 75 W, varies by application

35.6

- Ettus Research recommends to always use the latest stable version of UHD
- If you need to clean the module, wipe it with a dry towel.

35.6.1

- Current Hardware Revision: A
- Minimum version of UHD required: 3.14.0.0
- Due to product compliance restrictions on products with TPM (Trusted Platform Module) components to a few countries, the USRP N320/N321 is available in two variants:
    - Standard variant with TPM
    - Non-TPM variant

35.6.2

- Current Hardware Revision: A
- Minimum version of UHD required: 3.14.0.0

- Due to product compliance restrictions on products with TPM (Trusted Platform Module) components to a few countries, the USRP N320/N321 is available in two variants:
    - Standard variant with TPM
    - Non-TPM variant

35.6.3

There are three master clock rates (MCR) supported on the N320/N321: 200 MHz, 245.76 MHz, 250 MHz

The sampling rate must be an integer decimation rate of the MCR. Ideally, this decimation factor should be an even number. An odd decimation factor will result in additional unwanted attenuation (roll-off from the CIC filter in the DUC and DDC blocks in the FPGA). The valid decimation rates are between 1 and 1024.

If the desired sampling rate is not directly supported by the hardware, then it will be necessary to re-sample in software. This can be done in C++ using libraries such as Liquid DSP [1], or can be done in GNU Radio, in which there are three blocks that perform sampling rate conversion.

35.7

35.7.1

(L × W × H)

- 35.71 cm × 21.11 cm × 4.37 cm
- 14.06 in. × 8.31 in. × 1.72 in.

35.7.2

- 3.13 kg

35.7.3

35.7.3.1

- Media:cu usrp-n320.pdf

35.7.3.2

- Media:cu usrp-n321.pdf

35.7.4

- Media:USRP_N320.stp.7z
- Media:USRP_N321.stp.7z

35.8

35.8.1

- N320 / N321: 0 to 50 °C

35.8.2

- N320 / N321: -40 to 70 °C

35.8.3

- 10% to 90% non-condensing

35.9

35.9.1

- Motherboard: File:USRP N3XX MB Schematic.pdf
- Daughterboard: File:N32X Daughterboard Schematic.pdf

35.10

- Support GPSDO NMEA Strings

35.10.1

You can query the lock status with the `gps_locked` sensor, as well as obtain raw NMEA sentences using the `gps_gprmc`, and `gps_gpgga` sensors. Location information can be parsed out of the `gps_gpgga` sensor by using `gpsd` or another NMEA parser.

35.10.2

Module Specifications	
1 PPS Timing Accuracy from GPS receiver	<8ns to UTC RMS (1-Sigma) GPS Locked
Holdover Stability (1 week with GPS)	<±50us over 3 Hour Period @+25°C (No Motion, No Airflow)
1 PPS Output	3.3VDC CMOS
Serial Port	TTL Level, GPS NMEA Output with 1Hz or 5Hz update rate, Integrated into UHD

GPS Frequency	L1, C/A 1574MHz
GPS Antenna	Active (3V compatible) or Passive (0dB to +30dB gain)
GPS Receiver	65 Channels, QZSS, SBAS WAAS, EGNOS, MSAS capable Supports Position and Hold over-determined clock mode
Sensitivity	Acquisition -148dBm, Tracking -165dBm
TTFF	Cold Start: <32 sec, Warm Start: 1 sec, Hot Start: 1 sec
ADEV	10s: <7E-011 10Ks: <2E-012 (GPS Locked, 25°C, no motion, no airflow)
Warm Up Time / Stabilization Time	<10 min at +25C to 1E-09 Accuracy
Supply Voltage (Vdd)	3.3V Single-Supply, +0.2V/-0.15V
Power Consumption	<0.16W
Operating Temperature	-10°C to +70°C
Storage Temperature	-45C to 85C
Oscillator Specifications (internal)	
Frequency Output of local crystal	20MHz CMOS 3Vpp
Phase Noise	
20MHz Reference	2E-08 After 1 Hour @ +25°C without GPS
RF Output Amplitude	3Vpp CMOS
20MHz Phase Jitter (100Hz to 10MHz)	<0.135ps rms
Frequency Stability Over Temperature (0°C to +60°C)	+0.1ppm (internal TCXO without GPS)
Warm Up Time	< 1 min at +25C
Phase Noise at 20MHz	1Hz -65 dBc/Hz 10Hz -97 dBc/Hz 100Hz -116 dBc/Hz 1kHz -136 dBc/Hz 10kHz <-148 dBc/Hz 100 kHz <-155 dBc/Hz

35.10.3

- Spec Sheet: http://www.jackson-labs.com/assets/uploads/main/LTE-Lite_specsheet_20MHz.pdf
- User Manual: http://www.jackson-labs.com/assets/uploads/main/LTE-Lite.pdf

35.11

35.11.1

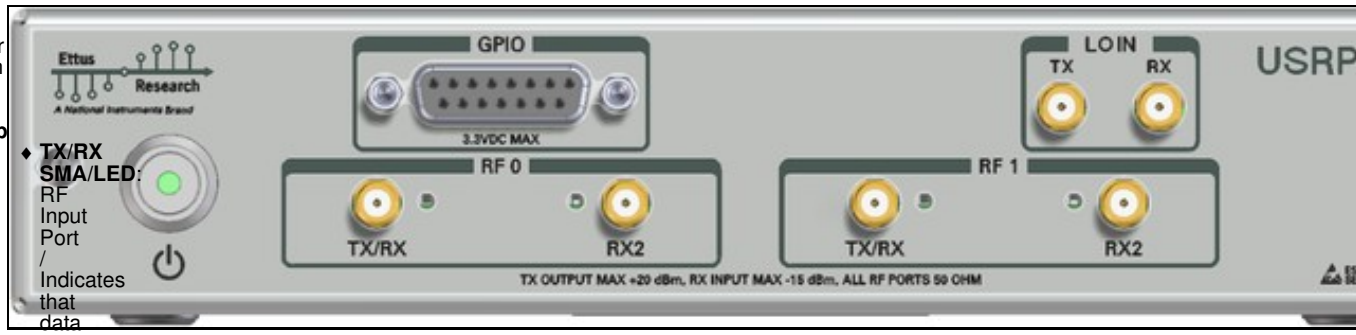
The Verilog code for the FPGA in the USRP N320/N321 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Specifically, changing the I/O interface of the FPGA in any way, or modifying the pin and timing constraint files, could result in physical damage to other components on the motherboard, external to the FPGA, and doing this will void the warranty. The constraint files should not be modified. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

35.12

35.12.1

35.12.1.1

- **PWR**: Power switch
- **RF 0 Group**



  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 0, channel 0
  - **RX2 SMA/LED**: RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 0, channel 0
- **RF 1 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 1, channel 0.
  - **RX2 SMA/LED**: RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 1, channel 0

- **LO IN**
  - **TX**: Input port for TX LO. Supported LO frequency range is from 450 MHz to 6 GHz. External LO inputs below 450 MHz are not supported. The LO input signal level should be +5 dBm, but may be between +3 dBm and +7 dBm.
-
  - **RX**: Input port for RX LO of Daughterboard 0. Supported LO frequency range is from 450 MHz to 6 GHz. External LO inputs below 450 MHz are not supported. The LO input signal level should be +5 dBm, but may be between

+3 dBm and +7 dBm.

- **GPIO**
  - **GPIO**: DB15 GPIO Interface. Additional details below.

35.12.1.2

- **GPS ANT**: Connection for the GPS antenna
- **REF IN**: Reference clock input
- **PPS/TRIG IN**: Input port for the PPS signal
- **TRIG OUT**: Output port for the exported reference clock
- **PWR**: Connector for the USRP N320 Series power supply
- **RESET**: Input button to reset device
- **MicroSD**: MicroSD Card for OE Linux File System
- **Console JTAG**: Micro USB connector for the on-board USB-JTAG programmer as well as TTY login to

the console
- **USB 2.0**: Host USB connector to ARM CPU
- **SFP+**: 1/10Gb SFP+ ports for Ethernet interfaces
- **QSFP+**: QSFP+ port for Ethernet interfaces (2 x 10Gb lanes)
- **10/1000/1000**: 10/100/1000 Mb Ethernet interface to ARM CPU

35.12.1.3

The GPIO port is not meant to drive big loads. You should not try to source more than 5mA per pin.

The +3.3V is for ESD clamping purposes only and not designed to deliver high currents.

35.12.1.3.1

The hardware power on state and UHD initial state for the front-panel GPIOs is high-Z. For the N320, there are no external pull-ups/pull-downs for the GPIO pins, but the FPGAs do have them and they are configured as follows: pull-down.

35.12.1.3.2

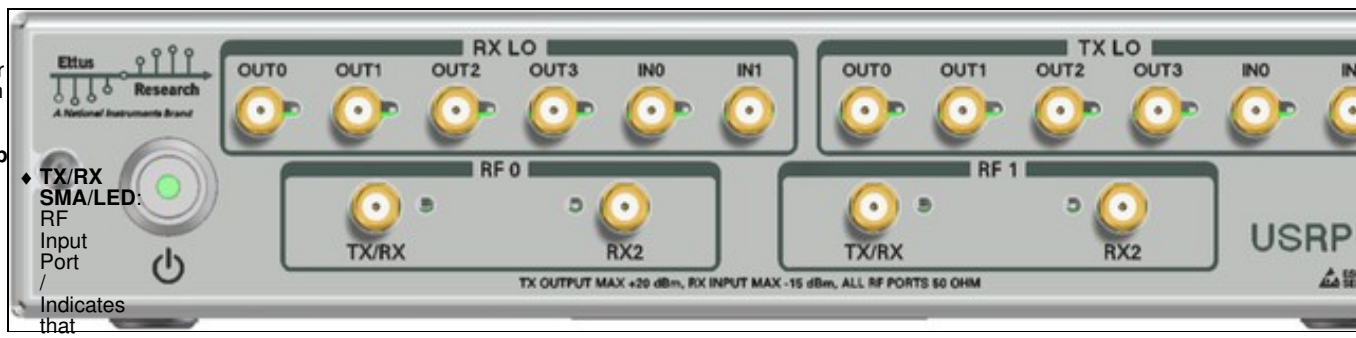
- Pin 1: +3.3V
- Pin 2: Data[0]
- Pin 3: Data[1]
- Pin 4: Data[2]
- Pin 5: Data[3]
- Pin 6: Data[4]
- Pin 7: Data[5]
- Pin 8: Data[6]
- Pin 9: Data[7]
- Pin 10: Data[8]
- Pin 11: Data[9]
- Pin 12: Data[10]
- Pin 13: Data[11]
- Pin 14: 0V
- Pin 15: 0V

Note: Please see the E3x0/X3x0/N3x0 GPIO API for information on configuring and using the GPIO bus.

35.12.2

35.12.2.1

- **PWR**: Power switch
- **RF 0 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on



the TX/RX channel on daughterboard 0, channel 0

- **RX2 SMA/LED**: RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 0, channel 0

- **RF 1 Group**
  - **TX/RX SMA/LED**: RF Input Port / Indicates that data is streaming on the TX/RX channel on daughterboard 1, channel 0.
  - **RX2 SMA/LED**: RF Input Port / Indicates that data is streaming on the RX2 channel on daughterboard 1, channel 0

- **TX LO**
  - **IN0**: Input port for TX LO. Supported LO frequency range is from 450 MHz to

6 GHz. External LO inputs below 450 MHz are not supported. The LO input signal level should be +5 dBm, but may be between +3 dBm and +7 dBm.

- **IN1**: Input port for TX LO. Supported LO frequency range is from 450 MHz to 6 GHz. External LO inputs below 450 MHz are not supported. The LO input signal level should be +5 dBm, but may be between +3 dBm and +7 dBm.
- **OUT0-3**: TX LO Outputs from 1:4 splitter

- **RX LO**
  - **IN0**: Input port for RX LO

of Daughterboard 0. Supported LO frequency range is from 450 MHz to 6 GHz. External LO inputs below 450 MHz are not supported. The LO input signal level should be +5 dBm, but may be between +3 dBm and +7 dBm.

- **IN1**: Input port for RX LO of Daughterboard 0. Supported LO frequency range is from 450 MHz to 6 GHz. External LO inputs below 450 MHz are not supported. The LO input signal level should be +5 dBm, but may be between +3 dBm and +7 dBm.

- **OUT0-3**: RX LO Outputs from 1:4 splitter

- Additional details on the N321 Distribution Board can be found here: https://files.ettus.com/manual/page_usrp_n3xx.html#n3xx_rh_lo_sharing

35.12.2.2

- **GPS ANT**: Connection for the GPS antenna
- **REF IN**: Reference clock input
- **PPS/TRIG IN**: Input port for the PPS signal
- **TRIG OUT**: Output port for the exported reference clock
- **PWR**: Connector for the USRP N320 Series power supply

- **RESET**: Input button to reset device

- **MicroSD**: MicroSD Card for OE Linux File System
- **JTAG**: Micro USB connector for the on-board USB-JTAG programmer
- **USB 2.0**: Host USB connector to ARM CPU
- **SFP+**: 1/10Gb SFP+ ports for

Ethernet interfaces
- **QSFP+**: QSFP+ port for Ethernet interfaces (2 x 10Gb lanes)
- **10/1000/1000**: 10/100/1000 Mb Ethernet interface to ARM CPU

35.12.3

Using an external 10 MHz reference clock, a square wave will offer the best phase noise performance, but a sinusoidal is acceptable. The power level of the reference clock cannot exceed +10 dBm.

35.12.4

Using a PPS signal for timestamp synchronization requires a square wave signal with the following a 5Vpp amplitude.

To test the PPS input, you can use the following tool from the UHD examples:

- `<args>` are device address arguments (optional if only one USRP device is on your machine)

```
cd <install-path>/lib/uhd/examples ./test_pps_input ?args=<args>
```

35.13

35.13.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

35.13.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

35.14

Found on the NI Product Certifications lookup tool here.

35.15

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

35.16

35.16.1

Recommended 10 Gigabit Ethernet Cards

- Intel X520-DA2
  - Intel® Ethernet Converged Network Adapter X520-DA2
- Intel X520-DA1
  - Intel® Ethernet Converged Network Adapter X520-DA1
- Intel X710-DA2
  - Intel® Ethernet Converged Network Adapter X710-DA2
- Intel X710-DA4
  - Intel® Ethernet Converged Network Adapter X710-DA4
- Mellanox MCX4121A-ACAT
  - Mellanox MCX4121A-ACAT

35.17

The power supply provided with the USRP N320 kit is packaged with a power cord that is compatible with power outlets in the US/Japan. If you are not using the USRP N320 in the US/Japan, we recommend purchasing the International USRP N320 Power Cord set.

35.18

The USRP N320/N321 was designed to be used with a 1U Rackmount Assembly for building high-density MIMO systems in a compact and well-organized setup. This mount *requires* two compatible USRPs, and provides rubber standoffs between the USRPs to avoid both direct contact and surface scratching. If the user will be developing in a laboratory environment or building a high-channel count USRP system, then a 1U Rackmount Assembly is highly recommended. This specific mount is compatible with only the USRP N300, N310, N320, and N321, and allows the integration of up to eight bidirectional RF channels per 1U.

35.19

Ettus Research currently offers direct-connect, copper cabling accessories for the USRP N320/N321. However, it is also possible to use multi-mode fiber instead of copper connections for these devices. In this section, we will provide general guidance on the types of fiber adapters and cables that can be used with these products.

The USRP N320/N321 USRP is compatible with most brands of SFP+ fiber adapters. In some cases, other equipment in the systems such as 1/10 Gigabit Ethernet switches are only compatible with specific brands of SFP+ adapters and cables. As a general rule, we recommend checking compatibility with the switches and network cards in your system before purchasing an adapter.

Ettus Research does test the USRP N320/N321 USRP devices with our 10 Gigabit Ethernet Connectivity Kit and a Blade Networks G8124 1/10 GigE switch. Here are is a list of known-good cables and adapters.

Ettus Research has only tested multi-mode fiber accessories.

35.19.1

- Approved Optics BN-CKM-SP-SR-A

35.19.2

- Elpeus 10GbE SFP+ AOC Cable, 3 meters

35.20

Many new motherboards come equipped with an onboard 10Gb RJ45 NIC. It is possible to use a SFP+ to RJ45 adapter and operate at 10Gb speeds using a Cat6/7 Ethernet cables.

Ettus Research has tested the adapters linked below.

35.20.1

- 10Gtek SFP+ to RJ45 Copper Module
- ProLabs 10G-SFPP-T-C

36 X310

36.1

The Ettus Research USRP X310 is a high-performance, scalable software defined radio (SDR) platform for designing and deploying next generation wireless communications systems. The hardware architecture combines two extended-bandwidth daughterboard slots covering DC ? 6 GHz with up to 160 MHz of baseband bandwidth, multiple high-speed interface options (PCIe, dual 10 GigE, dual 1 GigE), and a large user-programmable Kintex-7 FPGA in a convenient desktop or rack-mountable half-wide 1U form factor.

36.2

36.2.1

- Xilinx Kintex-7 XC7K325T FPGA
- 14 bit 200 MS/s ADC
- 16 bit 800 MS/s DAC
- Frequency range: DC - 6 GHz with suitable daughterboard
- Up 160MHz bandwidth per channel
- Two wide-bandwidth RF daughterboard slots
- Optional GPSDO
- Multiple high-speed interfaces (Dual 10G, PCIe Express, ExpressCard, Dual 1G)

36.2.2

- Xilinx Kintex-7 XC7K410T FPGA
- 14 bit 200 MS/s ADC
- 16 bit 800 MS/s DAC
- Frequency range: DC - 6 GHz with suitable daughterboard
- Up 160MHz bandwidth per channel
- Two wide-bandwidth RF daughterboard slots
- Optional GPSDO
- Multiple high-speed interfaces (Dual 10G, PCIe Express, ExpressCard, Dual 1G)

36.3

- WBX-120 / WBX-40
- SBX-120 / SBX-40
- CBX-120 / CBX-40
- UBX-160 / UBX-40
- BasicTX / BasicRX
- LFRX / LFTX
- TwinRX
- DBSRX2 (EOL)
- RFX Series (EOL)
- TVRX2 (EOL)

36.4

36.4.1

- SSB/LO Suppression -35/50 dBc
- Phase Noise 3.5 GHz 1.0 deg RMS
- Phase Noise 6 GHz 1.5 deg RMS
- Power Output >10dBm

- IIP3 (@ typ NF) 0dBm
- Typical Noise Figure 8dB

36.5

- Ettus Research recommends to always use the latest stable version of UHD

36.5.1

- Current Hardware Revision: 8
- Minimum version of UHD required: 3.9.0

36.5.2

- Current Hardware Revision: 8
- Minimum version of UHD required: 3.9.0

36.5.3

There are two master clock rates (MCR) supported on the X300 and X310: 200.0 MHz and 184.32 MHz.

The sampling rate must be an integer decimation rate of the MCR. Ideally, this decimation factor should be an even number. An odd decimation factor will result in additional unwanted attenuation (roll-off from the CIC filter in the DUC and DDC blocks in the FPGA). The valid decimation rates are between 1 and 1024.

For the MCR of 200.0 MHz, the achievable sampling rates using an even decimation factor are 200.0, 100.0, 50.0, 33.33, 25.0, 20.0, 16.67, 14.286 Msps, ... 195.31 Ksps.

For the MCR of 184.32 MHz, the achievable sampling rates using an even decimation factor are 184.32, 92.16, 46.08, 30.72, 23.04, 18.432, 15.36, 13.166 Msps, ... 180.0 Ksps.

If the desired sampling rate is not directly supported by the hardware, then it will be necessary to re-sample in software. This can be done in C++ using libraries such as Liquid DSP [1], or can be done in GNU Radio, in which there are three blocks that perform sampling rate conversion.

36.6

36.6.1

27.7 x 21.8 x 3.9 cm

36.6.2

With 2x SBX-120: 1.7kg

36.6.3

- Enclosure
- Motherboard
- Rackmount kit

36.6.4

36.6.4.1

- Motherboard

36.6.4.2

- Enclosure

36.7

36.7.1

- X300/X310: 25 °C

36.7.2

- 10% to 90% non-condensing

36.8

36.8.1

X300/X310 Schematics

36.9

Part Number	Description	Schematic ID (Page)
XC7K325T / XC7K410T	FPGA	U23 (3,5,8,9,10,18)
AD9146	Dual Channel, 16-Bit, 1230 MSPS DAC	U12, U36 (7)
ADS62P48	Dual Channel, 14-Bit 210 MSPS ADC	U11, U35 (6)
FIN1002	High Speed Differential Receiver	U3, U5, U31, U32 (4)

24LC256T	EEPROM	U530 (11)
LMK04816BISQ/NOPB_1/3	Jitter Cleaner With Dual Loop PLLs	U531 (11)
SY89547LMGTR	Multiplexer	U506 (12)
SN74AUP1T17	Single Schmitt-Trigger Buffer Gate	U6, U519 (12)
TPS54620RGYT	Synchronous Step Down SWIFT? Converter	U515 (21); U516 (26)
LT1764EQ-3.3	Voltage Regulator	U27 (21); U516 (26)
TPS7A47	Voltage Regulator	U28, U532 (21)
LTC3603EUF_TRPBF	Monolithic Synchronous Step-Down Regulator	U517 (23); U500 (25); U514, U513 (27)
TPS77625_SM	Low-Dropout Voltage Regulators	U30 (23)
TPS79318_SM	Low-Dropout Voltage Regulators	U510 (27)
OSC-96MHZ-724821-01	Voltage Controlled Crystal Oscillator	U25 (11)

36.10

- Support GPSDO NMEA Strings
- JacksonLabs LC_XO

36.10.1

You can query the lock status with the `gps_locked` sensor, as well as obtain raw NMEA sentences using the `gps_gprmc`, and `gps_gpgga` sensors. Location information can be parsed out of the `gps_gpgga` sensor by using `gpsd` or another NMEA parser.

36.11

- Utilization statistics are subject to change between UHD releases. This information is current as of UHD 3.9.4 and was taken directly from Xilinx Vivado 2014.4.

36.11.1

```
1. Slice Logic
--------------

+----------------------------+-------+-----------+-------+
|          Site Type         |  Used | Available | Util% |
+----------------------------+-------+-----------+-------+
| Slice LUTs                 | 61622 |    203800 | 30.23 |
|   LUT as Logic             | 52887 |    203800 | 25.95 |
|   LUT as Memory            |  8735 |     64000 | 13.64 |
|     LUT as Distributed RAM |  1878 |           |       |
|     LUT as Shift Register  |  6857 |           |       |
| Slice Registers            | 62961 |    407600 | 15.44 |
|   Register as Flip Flop    | 62961 |    407600 | 15.44 |
|   Register as Latch        |     0 |    407600 |  0.00 |
| F7 Muxes                   |  1209 |    101900 |  1.18 |
| F8 Muxes                   |   150 |     50950 |  0.29 |
+----------------------------+-------+-----------+-------+

3. Memory
---------

+-------------------+------+-----------+-------+
|     Site Type     | Used | Available | Util% |
+-------------------+------+-----------+-------+
| Block RAM Tile    |  409 |       445 | 91.91 |
|   RAMB36/FIFO*    |  398 |       445 | 89.43 |
|     RAMB36E1 only |  398 |           |       |
|   RAMB18          |   22 |       890 |  2.47 |
|     RAMB18E1 only |   22 |           |       |
+-------------------+------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a


4. DSP
------

+----------------+------+-----------+-------+
|    Site Type   | Used | Available | Util% |
+----------------+------+-----------+-------+
| DSPs           |  123 |       840 | 14.64 |
|   DSP48E1 only |  123 |           |       |
+----------------+------+-----------+-------+
```

36.11.2

```
1. Slice Logic
--------------

+----------------------------+-------+-----------+-------+
|          Site Type         |  Used | Available | Util% |
+----------------------------+-------+-----------+-------+
| Slice LUTs                 | 61616 |    254200 | 24.23 |
|   LUT as Logic             | 52885 |    254200 | 20.80 |
|   LUT as Memory            |  8731 |     90600 |  9.63 |
|     LUT as Distributed RAM |  1878 |           |       |
|     LUT as Shift Register  |  6853 |           |       |
| Slice Registers            | 62958 |    508400 | 12.38 |
|   Register as Flip Flop    | 62958 |    508400 | 12.38 |
|   Register as Latch        |     0 |    508400 |  0.00 |
| F7 Muxes                   |  1209 |    127100 |  0.95 |
| F8 Muxes                   |   150 |     63550 |  0.23 |
+----------------------------+-------+-----------+-------+

3. Memory
---------
```

```
+-------------------+------+-----------+-------+
|     Site Type     | Used | Available | Util% |
+-------------------+------+-----------+-------+
| Block RAM Tile    |  409 |       795 | 51.44 |
|   RAMB36/FIFO*    |  398 |       795 | 50.06 |
|     RAMB36E1 only |  398 |           |       |
|   RAMB18          |   22 |      1590 |  1.38 |
|     RAMB18E1 only |   22 |           |       |
+-------------------+------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a


4. DSP
------

+-----------------+------+-----------+-------+
|    Site Type    | Used | Available | Util% |
+-----------------+------+-----------+-------+
| DSPs            |  123 |      1540 |  7.98 |
|   DSP48E1 only  |  123 |           |       |
+-----------------+------+-----------+-------+
```

36.11.3

The Verilog code for the FPGA in the USRP X300 and USRP X310 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Specifically, changing the I/O interface of the FPGA in any way (do not remove any of the I/O for the PCIe interface, such as `x300_pcie_int` and `LvFpga_Chinch_Interface`), or modifying the pin and timing constraint files, could result in physical damage to other components on the motherboard, external to the FPGA, and doing this will void the warranty. Also, even if the PCIe interface is not being used, you cannot remove or reassign these pins in the constraint file. The constraint files should not be modified. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

36.12

The USRP X300 series runs a small amount of software within the FPGA, within a ZPU soft processor. Its main responsibility is to provide access to some registers, handle the networking stacks, and monitor the USRP status.

The source code for the ZPU is stored in `firmware/usrp3/` in the UHD repository. To modify the firmware, you need to download a recent ZPU compiler (e.g. from https://github.com/zylin/zpugcc/tree/master/releases/20150428). Unpack the tarball (e.g., into `/usr/local`) and make sure the `zpu-elf-gcc` binary is in your path. Then, execute the following steps:

- Create and enter a build directory: `mkdir build && cd build`
- Run cmake: `cmake /path/to/firmware/usrp3/`
  - If all is correctly configured, and the ZPU compiler can be found, this will pass without errors.
- Build the firmware: `make`
  - This should yield output similar to this:

```
Scanning dependencies of target x300
[ 79%] Generating x300_main.map
[ 83%] Generating x300_main.bin
[ 87%] Generating x300_main.ihx
[ 91%] Generating x300_main.dump
[ 95%] Generating x300_main.rom
[100%] Generating x300_main.coe
[100%] Built target x300
```

- These files will be copied into the `build/x310` directory.
- To non-persistently load the newly built firmware image into the running FPGA image, simply launch a UHD session with the `fw` parameter. The binary must be within UHD's image dir, e.g. by copying x300_main.bin to the default image directory (e.g., `/usr/local/share/uhd/images`) or by temporarily setting the `UHD_IMAGES_DIR` variable:

```
UHD_IMAGES_DIR=/path/to/build/x300 uhd_usrp_probe --args type=x300,fw=x300_main.bin
```

- To permanently bake the firmware image into the FPGA bitfile, copy the file `x300_main.coe` to `fpga/usrp3/top/x300/ip/bootram/bootram.coe` and rebuild the bitfile.

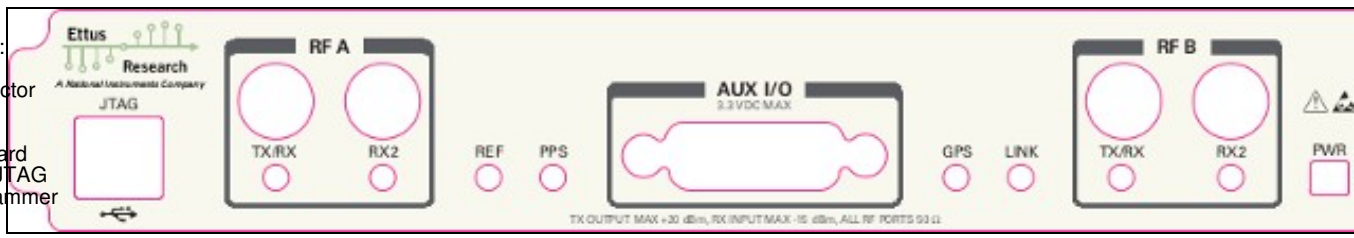
36.13

Follow the links below for additional information on configuring each interface for the USRP X300 or X310 SDRs.

- Dual 10 Gigabit Ethernet - 200 MS/s Full Duplex @ 16-bit
- PCIe Express (Desktop) - 200 MS/s Full Duplex @ 16-bit
- ExpressCard (Laptop) - 50 MS/s Full Duplex @ 16-bit
- Dual 1 Gigabit Ethernet - 25 MS/s Full Duplex @ 16-bit

36.13.1

- **JTAG**: USB connector for the on-board USB-JTAG programmer

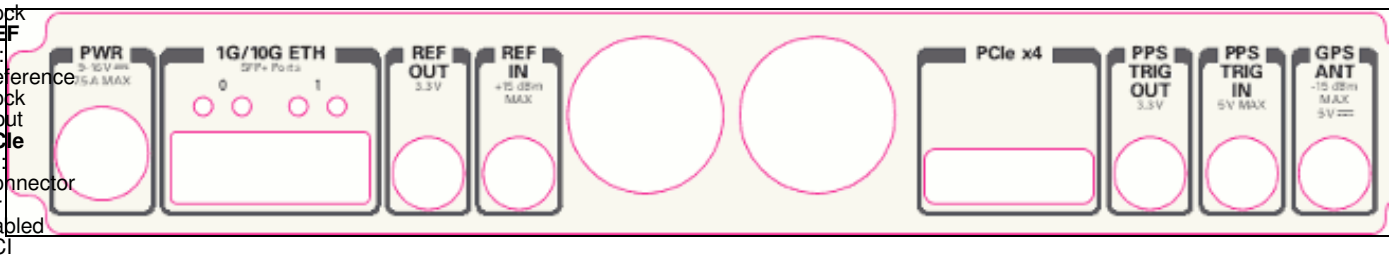


- **RF A Group**
    - **TX/RX LED**: Indicates that data is streaming on the TX/RX channel on daughterboard A
    - **RX2 LED**: Indicates that data is streaming on the RX2 channel on daughterboard A
- **REF**: Indicates that the external Reference Clock is locked
- **PPS**: Indicates a valid PPS signal by pulsing once per second
- **AUX I/O**: Front panel GPIO connector.
- **GPS**: Indicates that GPS reference is locked
- **LINK**: Indicates that the host computer is communicating with the device (Activity)
- **RF B Group**
    - **TX/RX LED**: Indicates that data is streaming on the TX/RX channel on

daughterboard B
    - **RX2 LED**: Indicates that data is streaming on the RX2 channel on daughterboard B
- **PWR**: Power switch

36.13.2

- **PWR**: Connector for the USRP-X Series power supply
- **1G/10G ETH**: SFP+ ports for Ethernet interfaces
- **REF OUT**: Output port for the exported reference clock
- **REF IN**: Reference clock input
- **PCIe x4**: Connector for Cabled PCI Express link
- **PPS/TRIG OUT**: Output port for the PPS signal
- **PPS/TRIG IN**: Input port for the PPS signal
- **GPS**: Connection for the GPS antenna



36.13.3

Using an external 10 MHz reference clock, a square wave will offer the best phase noise performance, but a sinusoid is acceptable. The power level of the reference clock cannot exceed +15 dBm.

36.13.4

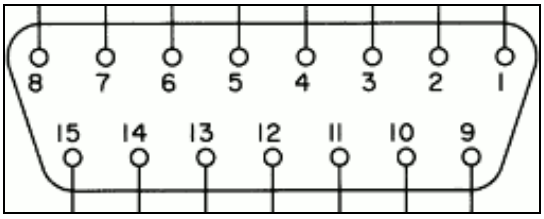
Using a PPS signal for timestamp synchronization requires a square wave signal with the following a 5Vpp amplitude.

To test the PPS input, you can use the following tool from the UHD examples:

- `<args>` are device address arguments (optional if only one USRP device is on your machine)

```
cd <install-path>/lib/uhd/examples ./test_pps_input ?args=<args>
```

36.13.5



The GPIO port is not meant to drive big loads. You should not try to source more than 5mA per pin.

The +3.3V is for ESD clamping purposes only and not designed to deliver high currents.

The switching speed is below 10 MHz.

36.13.5.1

The hardware power on state and UHD initial state for the front-panel GPIOs is high-Z. For the X3xx, there are no external pull-ups/pull-downs for the GPIO pins, but the FPGAs do have them and they are configured as follows: X3xx: pull-down.

36.13.5.2

- Pin 1: +3.3V
- Pin 2: Data[0]
- Pin 3: Data[1]
- Pin 4: Data[2]
- Pin 5: Data[3]
- Pin 6: Data[4]
- Pin 7: Data[5]
- Pin 8: Data[6]
- Pin 9: Data[7]
- Pin 10: Data[8]
- Pin 11: Data[9]
- Pin 12: Data[10]
- Pin 13: Data[11]
- Pin 14: 0V
- Pin 15: 0V

Note: Please see the E3x0/X3x0 GPIO API for information on configuring and using the GPIO bus.

36.13.6

LED	Detail	Description
DS1	1.2V	Power
DS2	TXRX1	Red: TX, Green: RX
DS3	RX1	Green: RX
DS4	REF	Reference Lock
DS5	PPS	Flashes on Edge
DS6	GPS	GPS Lock
DS7	SFP0	Link, Right, Green
DS8	SFP0	Link Activity, Left, Yellow
DS10	TXRX2	Red: TX Green: RX
DS11	RX2	Green: RX
DS12	6V	Daughterboard Power
DS13	3.8V	Power
DS14	3.3V	Management Power
DS15	3.3V	Auxiliary Management Power
DS16	3.3V	FPGA Power
DS19	SFP1	Link Active, Left, Yellow
DS20	SFP1	Link, Right, Green
DS21	LINK	Link Activity

36.13.7

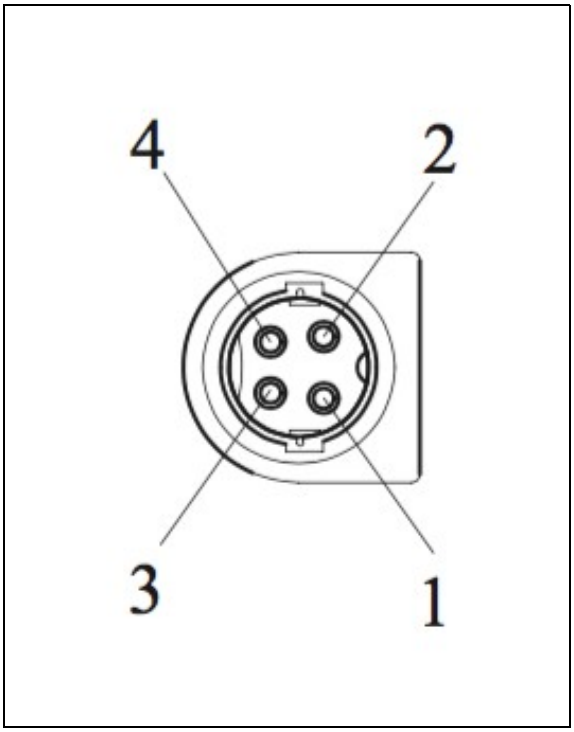
Model: PDP-40 by CUI Inc.

Power plug connectors for custom power harnesses can be purchased here:
https://www.digikey.com/products/en?KeyWords=CP-7340-ND&WT.z_cid=sp_102_buynow

Assembly instructions: Media:pdp-40.pdf

36.13.7.1

- Pins #1 / #2: 12v
- Pins #3 / #4: Ground



36.14

36.14.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

36.14.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

36.15

Found on the NI Product Certifications lookup tool here.

36.16

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

36.17

In terms of host bandwidth, interface options, and all other hardware features the USRP X300 and USRP 310 are identical. However, the USRP X310 provides a larger FPGA, a Xilinx XC7K410T, as opposed to XC7K325T. While both options provide a significant amount of free resources for custom FPGA development, the XC7K410T provides additional design margin, which translates to ease of development and future expandability. Most users choose the USRP X310 for their development.

USRP X300 and X310 FPGA Resource Summary

Resource Type	USRP X300 (XC7K325T) Count	USRP X310 (XC7K410T) Count
DSP48 Blocks	840	1540
Block Rams (18kB)	890	1590
Logic Cells	326,080	406,720
Slices (logic)	50,950	63,550

For up-to-date information on FPGA resource utilization in the stock FPGA design, please see "USRP 300/X310 FPGA Resources" in the Ettus Research knowledge base (https://kb.ettus.com).

36.18

With the increased sample rates used by the USRP X300 and USRP X310, these new device can support extended-bandwidth daughterboards. The WBX-120, SBX-120, and CBX-120 are recommended to take advantage of the full bandwidth capability of the USRP X300 and X310. The WBX-120, SBX-120, and CBX-120 have been upgraded from their predecessors (40 MHz) to use 120 MHz baseband filters. You can select your daughterboard based on the center frequency of your primary application.

Daughterboard	Frequency Range	Bandwidth
WBX-120	50 MHz - 2200 MHz	120 MHz
SBX-120	400 MHz - 4400 MHz	120 MHz
CBX-120	1200 MHz - 6000 MHz	120 MHz
UBX-160	10 MHz - 6000 MHz	160 MHz
TwinRX	10 MHz - 6000 MHz	80 MHz per channel, 160 MHz total

If your application is in the HF frequency range, the LFRX and LFTX are recommended for up to 30 MHz of bandwidth per channel. The BasicRX and BasicTX are ideal for configurations that use an external frontend for tuning and filtering with either an IF or baseband interface.

The USRP X300 and X310 are backward compatible with legacy daughterboards except for the RFX Series and XCVR2450. Please note, while there are two daughterboard slots, the USRP X300/X310 can only support a single TVRX2.

If you plan to transmit or receive over the air, you should also purchase an antenna.

36.19

The USRP X300/X310 provide three interface options ? 1 Gigabit Ethernet (1 GigE), 10 Gigabit Ethernet (10 GigE), and PCI-Express (PCIe). The PCIe interface is always available regardless of what FPGA image is loaded. Ettus ships two FPGA image variants, the HG or HGS image which has one 1 GigE interfaces and one 10 GigE interfaces, and the XG image which has two 10 GigE interfaces. Generally, Ettus Research recommends using 10 GigE to achieve the maximum throughput available from the USRP X300/X310. PCIe is recommended for applications that require the lowest possible latency, which is a desirable characteristic for PHY/MAC research. If your application does not require the full bandwidth of the USRP ? X300 and X310, the 1 GigE interface serves as a cost-effective fall-back option. Ettus Research provides a complete interface kit for each of these options, which is also shown in Table 3.

Table 3 - Interface Performance Summary

Interface	Throughput (MS/s @ 16-bit)	Target	Recommended Kit
1 Gigabit	25 MS/s	Desktop/Laptop	Components provided with USRP X300/X310 kit. For additional connections, purchase the following: SFP Adapter + GigE Cable
10 Gigabit	200 MS/s	Desktop	10 GigE Interface Kit
PCI-Express (PCIe, 4 lane)	200 MS/S	Desktop	PCI-Express Desktop Kit
Express Card (PCIe, 1 lane)	50 MS/s	Laptop	ExpressCard Kit

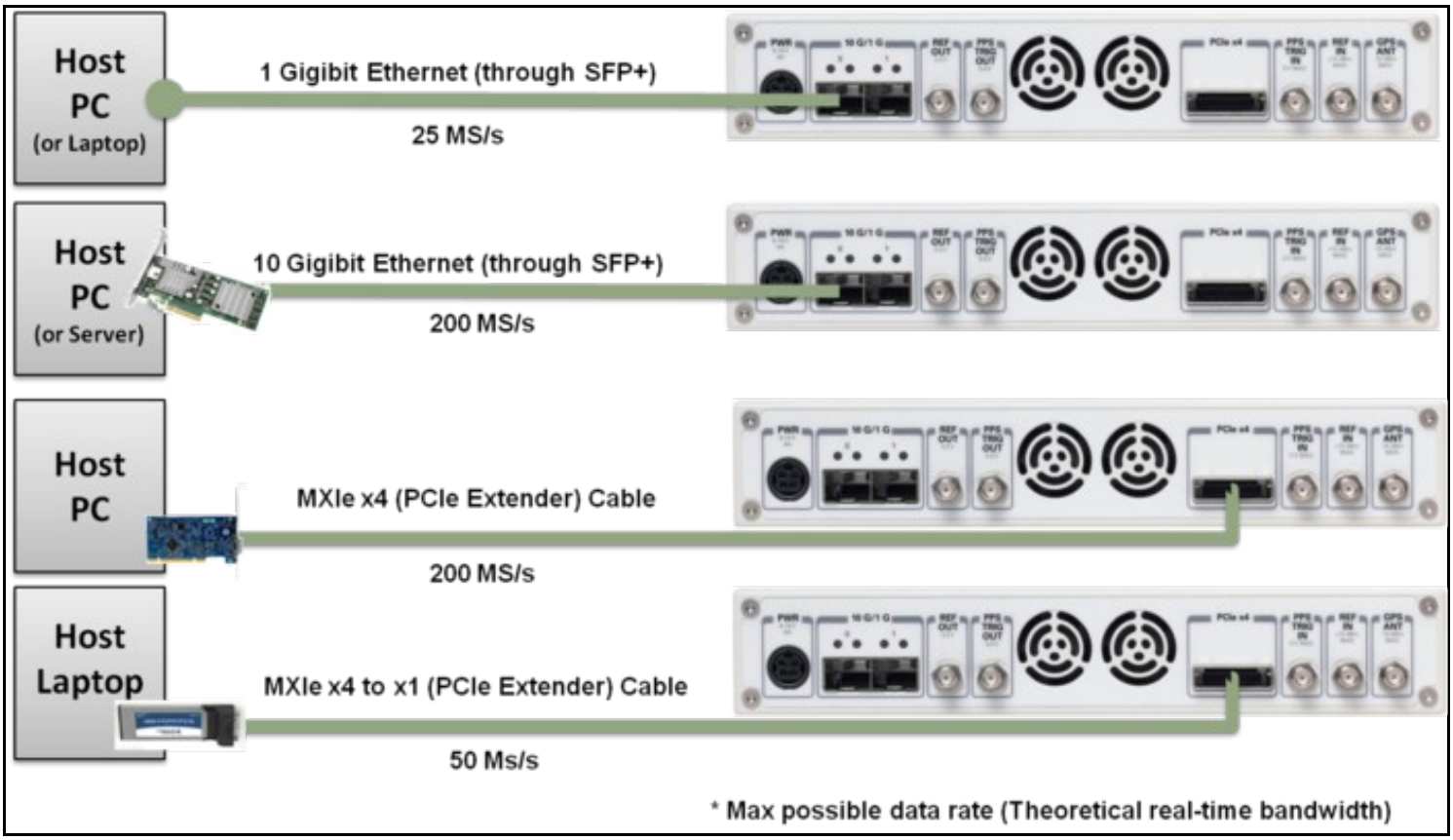


Figure 2 - Host Interface Options

36.19.1

See this app note for how to use the X3x0 with dual 10 GbE links.

Recommended 10 Gigabit Ethernet Cards

- Intel X520-DA2
  - Intel® Ethernet Converged Network Adapter X520-DA2
- Intel X520-DA1
  - Intel® Ethernet Converged Network Adapter X520-DA1
- Intel X710-DA2
  - Intel® Ethernet Converged Network Adapter X710-DA2
- Intel X710-DA4
  - Intel® Ethernet Converged Network Adapter X710-DA4
- Mellanox MCX4121A-ACAT
  - Mellanox MCX4121A-ACAT

36.20

The power supply provided with the USRP X300/X310 kit is packaged with a power cord that is compatible with power outlets in the US/Japan. If you are not using the USRP X300/X310 in the US/Japan, we recommend purchasing the International USRP X300/X310 Power Cord set.

36.21

The USRP X300 and USRP X310 provide the option to integrate a high-accuracy GPS-disciplined oscillator (GPSDO). The GPSDO improves the accuracy of the internal frequency reference to 20 ppb, or 0.1 ppb if the GPS is synchronized to the GPS constellation. When synchronized to the GPS constellation, all USRP ? devices will also be synchronized in time within 50 ns.

	Internal TCXO	GPS-Disciplined Clock
Frequency Reference	TCXO	OCXO
Frequency Accuracy (unlocked)	± 2.5ppm ± 2,500 Hz @ 1 GHz	± 25 ppb ± 25 Hz @ 1 GHz
Frequency Accuracy (GPS-Disciplined)		± 0.01ppb ~ ± 0.01 Hz @ 1 GHz
GPS Time Sync Accuracy		±50ns to UTC Time**
10 MHz Reference Phase Drift with GPS Sync		<±20ns After 1 Hour**

36.22

The GPSDO Mini Kit will improve the accuracy of the USRP reference clock, even if it does not receive signals from the GPS Constellation. However, to achieve the best accuracy possible, and to achieve global timing alignment across multiple USRPs, Ettus Research recommends the GPSDO Mini Antenna Kit.

36.23

The USRP X300 and X310 include a DB15 connector on the front panel that provides convenient access to GPIO signals. Each pin can be configured as an input or output, uses 3.3V-level logic, and is protected with basic anti-static circuitry. These pins can be used to control external devices like RF switches and amplifiers, trigger software events on the host, or even provide basic debugging functionality. The USRP GPIO Kit is an affordable option that provides access to these signals with a DB15 cable and a breakout board. The breakout board allows the user to connect external devices through a terminal block. The user can also solder wires and components into the dedicated prototyping area.

36.24

Multiple USRP X300/X310s can be synchronized for coherent operation by sharing a common 10 MHz and 1 PPS signal. We recommend using a star-distribution topology with an OctoClock or OctoClock-G, as seen in Figure 4. This requires matched length cables to be used for both 10 MHz and 1 PPS.

For more information about MIMO operation, please see the MIMO and Synchronization Application Note.

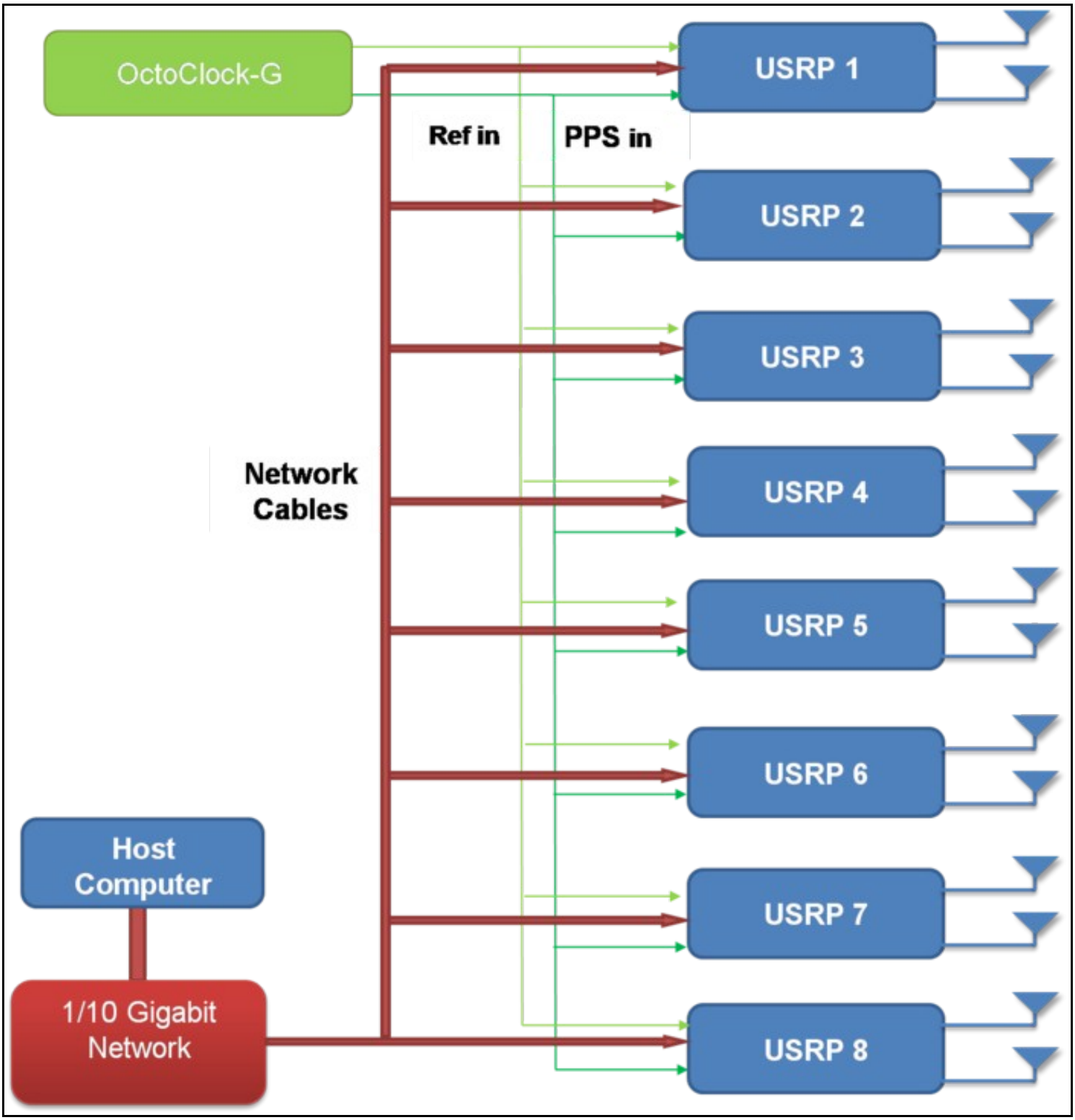


Figure 4 - Star-Distribution of 10 MHz/PPS Signals with OctoClock

36.25

The USRP X300 and X310 were designed to be used with a 1U Rackmount Assembly for building high-density MIMO systems in a compact and well-organized setup. This mount supports one or two compatible USRPs, and if two are mounted then there are rubber standoffs between the USRPs to avoid both direct contact and surface scratching. If the user will be developing in a laboratory environment or building a high-channel count USRP system, then a 1U Rackmount Assembly is highly recommended. This specific mount is compatible with only the USRP X300 and X310, and allows the integration of up to four bidirectional -- or eight receive-only -- RF channels per 1U.

36.26

Ettus Research currently offers direct-connect, copper cabling accessories for the USRP X300 and USRP X310. However, it is also possible to use multi-mode fiber instead of copper connections for these devices.

The USRP X Series is compatible with most brands of SFP+ fiber adapters. In some cases, other equipment in the systems such as 1/10 Gigabith Ethernet switches are only compatible with specific brands of SFP+ adapters and cables. As a general rule, we recommend checking compatibility with the switches and network cards in your system before purchasing an adapter.

Ettus Research does test the USRP X Series devices with our 10 Gigabit Ethernet Connectivity Kit and a Blade Networks G8124 1/10 GigE switch. Here are is a list of known-good cables and adapters.

Ettus Research has only tested multi-mode fiber accessories.

36.26.1

- Approved Optics Blade Networks BN-CKM-SP-SR-A

36.26.2

- Myricom Fiber Cables for 10GBase-SR, 3 Meters 10G-SR-3M

36.27

Many new motherboards come equipped with an onboard 10Gb RJ45 NIC. It is possible to use a SFP+ to RJ45 adapter and operate at 10Gb speeds using a Cat6/7 Ethernet cables.

Ettus Research has tested the adapters linked below.

36.27.1

- 10Gtek SFP+ to RJ45 Copper Module
- ProLabs 10G-SFPP-T-C

36.28

USRP? X300 and USRP? X310 SDRs Frequently Asked Questions

- **What is the bandwidth of the USRP X300/X310**

The ADC rate on each analog RX channel is 200 MS/s quadrature, which provides a theoretical analog bandwidth of approximately 80% of the Nyquist bandwidth of +/- 100 MHz (+/- 80 MHz around the center frequency). The resulting maximum theoretical analog bandwidth is 160 MHz. The actual analog bandwidth may be reduced due the RF daughterboard selected.

RF Daughterboard Bandwidths: See the daughterboard specifications [link]

FPGA Processing Bandwidth: Up to 200 MS/s quadrature.

Host Bandwidth: Up to 200 MS/s quadrature, dependent on selected interface

For more information about achieving the maximum bandwidth with a USRP X300/X310, please see the "USRP X300/X310 Configuration Guide" or the "USRP System Bandwidth" application note.

- **How can I program the USRP X300/X310**

Like all other USRP models, the USRP X300 and X310 are compatible with the USRP Hardware Driver? (UHD) architecture. The UHD architecture is a common driver that allows users to develop and execute applications on a host-PC. UHD provides a direct C++ API to control and stream to/from the USRP X300/X310. It also provides compatibility with a variety of third-party software frameworks including GNU Radio, LabVIEW, and Matlab. You may also customize the FPGA image provided with UHD to integrate your own signal processing. For more information about UHD, and supported software frameworks, please see:

http://files.ettus.com/manual/

- **How do I update the FPGA images and firmware with the latest from UHD**

You can find more information about updating the FPGA image in the UHD manual:
https://files.ettus.com/manual/page_usrp_x3x0.html#x3x0_getting_started_fpga_update

- **How can I modify the FPGA of the USRP X300/X310**

The source code (Verilog) for the USRP X300/X310 is available in the UHD repository. The build process leverages the existing CMAKE build system used to compile the host-side driver. A Linux-based setup will provide the best results.

Which FPGA toolchain required to build the FPGA images will depend upon your version of UHD. For more details please see the UHD Software Resource page.

- **How much free space is available in the USRP X300/X310 FPGA**

Please see the USRP X300/X310 FPGA resources page for more information.

- **What type of PC setup is recommended for use with the USRP X300/X310**

The type of PC required depends heavily on the complexity and bandwidth of the application. To demonstrate the USRP X300/X310, we typically use a desktop computer with a quadcore i7, 8+ GB of DDR3, and install the PCIe interface card that is also provide with the 10 GigE, PCIe, and ExpressCard interface kits.

- **What frequency range does the USRP X300/X310 cover**

The frequency range depends on the daughterboard select by the users. For more information, please see the USRP X300/X310 Configuration Guide.

- **What components do I need to purchase for a complete USRP X300/X310 system**

For a more comprehensive guide, please see the USRP X300/X310 Configuration Guide.

- **What is the difference between the USRP X300/X310**

The USRP X310 includes a larger Kintex-7 series FPGA (XC7K410T) with additional development resources for more complex designs. The USRP X300 includes the smaller XC7K325T FPGA.

- **What is the part number of the X300/X310 power connector**

Model: PDP-40 by CUI Inc. Power plug connectors for custom power harnesses can be purchased here:
https://www.digikey.com/products/en?KeyWords=CP-7340-ND&WT.z_cid=sp_102_buynow

37 USRP-2974

37.1

The NI USRP-2974 is a high-performance, USRP software defined radio (SDR) stand-alone device for designing and deploying next generation wireless communications systems. The hardware architecture combines two extended-bandwidth daughterboard slots covering 10 MHz ? 6 GHz with up to 160 MHz of baseband bandwidth, multiple high-speed interface options (PCIe, dual 10 GigE), an onboard Intel Core i7 processor, and a large user-programmable Kintex-7 FPGA in a convenient desktop or rack-mountable half-wide 2U form factor.

The USRP-2974 is the equivalent to a USRP X310 with two UBX-160 boards, a GPSDO and an onboard Intel i7 computer. The USRP-2974 comes with NI Linux RTOS pre-installed, but in order to use it with open-source tool-chain, a user will need to install Linux (preferably Fedora or Ubuntu) and then the USRP Hardware driver (UHD). After these have been installed, any other open-source tools can be installed, such as GNU Radio.

37.2

- Intel Core i7 6822EQ 2GHz Quad CoreProcessor
- 16GB DDR4 Memory
- 512GB SSD
- USB-to-UART to the CPU
- Xilinx Kintex-7 XC7K410T FPGA
- 14 bit 200 MS/s ADC
- 16 bit 800 MS/s DAC
- Frequency range: 10 MHz - 6 GHz
- Up 160MHz* bandwidth per channel
- 2 Transmit ports
- 2 Receive ports
- GPSDO
- Multiple high-speed interfaces (Dual 10G, PCIe Express, 1G)

37.3

System on module (SoM)	Congatec COM Express conga-TS170
CPU	Intel Core i7 6822EQ (2 GHz Quad Core)
Memory	SO-DIMM DDR4 16 GB
SFP+[1]	10G ETH connection to the SoM
Cabled PCIe	PCIe Gen 2 x4
MicroUSB[2]	USB-to-UART to the SoM
RJ45	1G ETH host connection

[1] Can be bypassed to the FPGA.

[2] Device port for external host.

37.4

Transmitter	
Number of channels	2
Frequency range	10 MHz to 6 GHz
Frequency step	<1kHz
Maximum output power	5 mW to 100 mW (7 dBm to 20 dBm)
Gain range[1]	0 dB to 31.5 dB
Gain step	0.5 dB
Maximum instantaneous real-time bandwidth	160 MHz
Receiver	
Number of channels	2
Frequency range	10 MHz to 6 GHz
Frequency step	<1 kHz
Gain range[2]	0 dB to 37.5 dB
Gain step	0.5 dB
Maximum input power	-15 dBm
Noise Figure	5 dB to 7 dB

Maximum instantaneous real-time bandwidth[3]	160MHz

[1] The output power resulting from the gain setting varies over the frequency band and among devices.

[2]The received signal amplitude resulting from the gain setting varies over the frequency band and among devices.

[3]The USRP-2974 receiver path has 84 MHz of bandwidth for center frequencies from 10 MHz to 500 MHz

NOTE: As mentioned earlier, the USRP-2974 incorporates 2 UBX-160 daughterboards. Therefore, for more information on RF performance, please see the UBX hardware resource page

37.5

37.5.1

- Minimum version of UHD required: **3.14.1.0**

37.5.2

There are two master clock rates (MCR) supported on the USRP-2974 like on the X310: 200.0 MHz and 184.32 MHz.

The sampling rate must be an integer decimation rate of the MCR. Ideally, this decimation factor should be an even number. An odd decimation factor will result in additional unwanted attenuation (roll-off from the CIC filter in the DUC and DDC blocks in the FPGA). The valid decimation rates are between 1 and 1024.

For the MCR of 200.0 MHz, the achievable sampling rates using an even decimation factor are 200.0, 100.0, 50.0, 33.33, 25.0, 20.0, 16.67, 14.286 Msps, ... 195.31 Ksps.

For the MCR of 184.32 MHz, the achievable sampling rates using an even decimation factor are 184.32, 92.16, 46.08, 30.72, 23.04, 18.432, 15.36, 13.166 Msps, ... 180.0 Ksps.

If the desired sampling rate is not directly supported by the hardware, then it will be necessary to re-sample in software. This can be done in C++ using libraries such as Liquid DSP [1], or can be done in GNU Radio, in which there are three blocks that perform sampling rate conversion.

37.6

37.6.1

(L × W × H) 29.08 cm × 21.84 cm × 7.98 cm (11.45 in. × 8.60 in. × 3.14 in.)

37.6.2

3.34 kg (7.35 lb)

37.7

Voltage range	14.25 V to 15.75 V DC
Current	10 A, maximum
Power	150 W, maximum

37.8

NOTE: Indoor use only

37.8.1

- 0 °C to 50 °C

37.8.2

- 2,000 m (800 mbar) (at 25 °C ambient temperature)

37.8.3

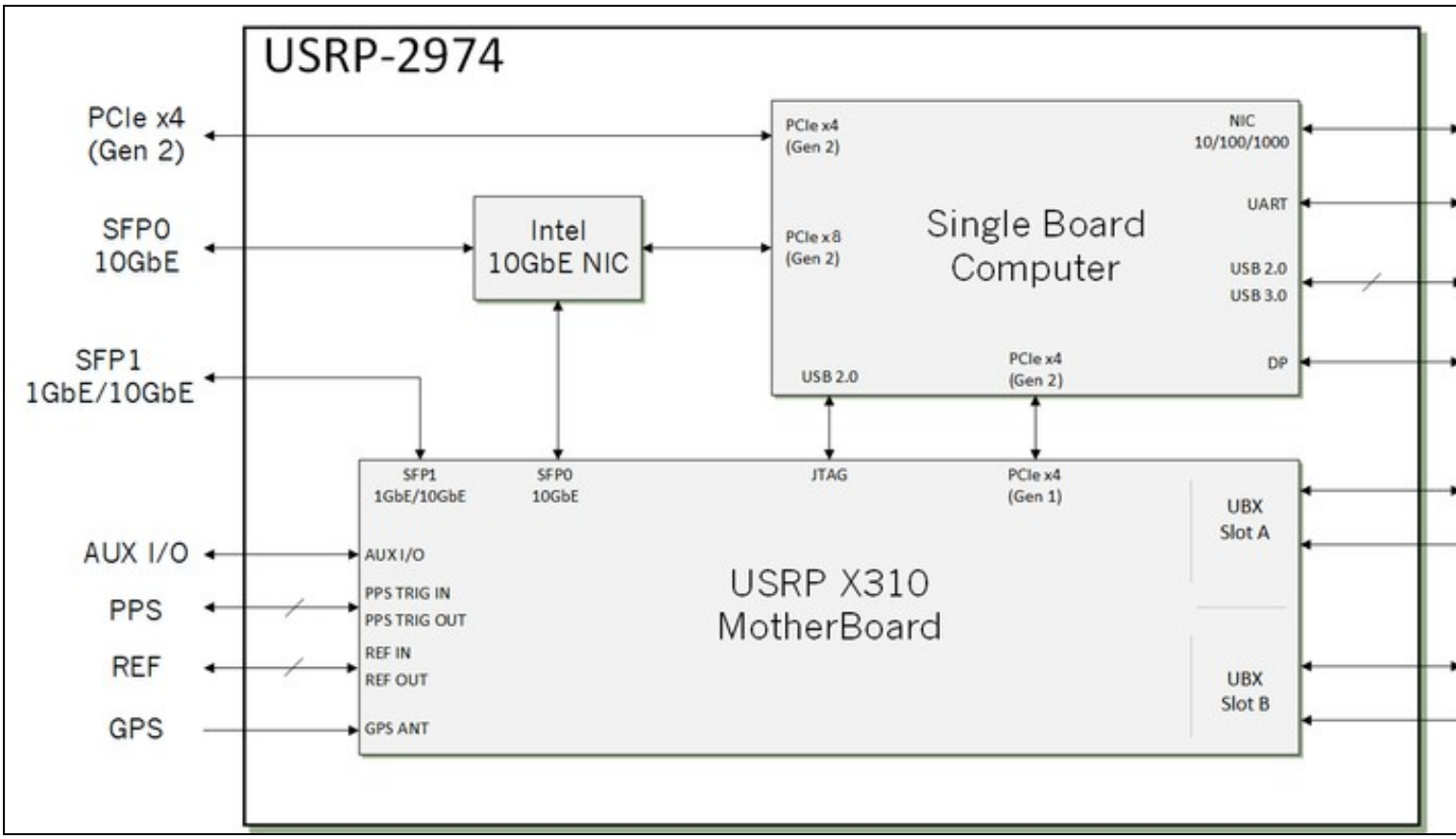
- 10% to 90% non-condensing

37.8.4

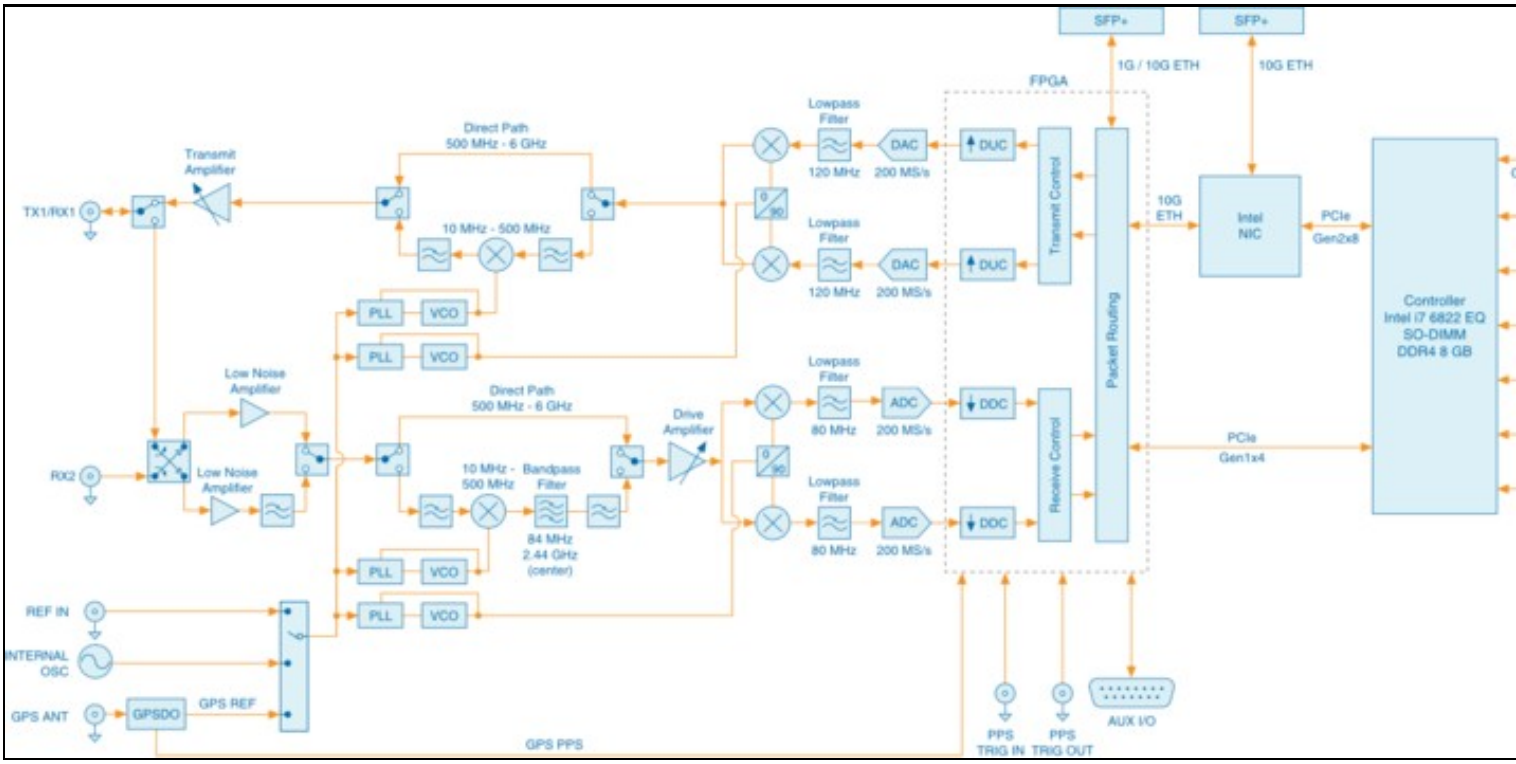
- 2

37.9

37.9.1



High Level Block Diagram of the USRP 2974



Detailed System Block Diagram

37.9.2

Because the USRP-2974 is a combination of an Intel i7 SOM and an X310 USRP, a user can reference the X310 Schematics.

X310 Schematics

37.10

Part Number	Description	Schematic ID (Page)
conga-TS170	System on Module (SoM)	
XC7K410T	FPGA	U23 (3,5,8,9,10,18)
AD9146	Dual Channel, 16-Bit, 1230 MSPS DAC	U12, U36 (7)
ADS62P48	Dual Channel, 14-Bit 210 MSPS ADC	U11, U35 (6)
FIN1002	High Speed Differential Receiver	U3, U5, U31, U32 (4)
24LC256T	EEPROM	U530 (11)
LMK04816BISQ/NOPB_1/3	Jitter Cleaner With Dual Loop PLLs	U531 (11)
SY89547LMGTR	Multiplexer	U506 (12)
SN74AUP1T17	Single Schmitt-Trigger Buffer Gate	U6, U519 (12)
TPS54620RGYT	Synchronous Step Down SWIFT? Converter	U515 (21); U516 (26)
LT1764EQ-3.3	Voltage Regulator	U27 (21); U516 (26)
TPS7A47	Voltage Regulator	U28, U532 (21)
LTC3603EUF_TRPBF	Monolithic Synchronous Step-Down Regulator	U517 (23); U500 (25); U514, U513 (27)
TPS77625	Low-Dropout Voltage Regulators	U30 (23)
TPS79318_SM	Low-Dropout Voltage Regulators	U510 (27)
OSC-96MHZ-724821-01	Voltage Controlled Crystal Oscillator	U25 (11)

37.11

FPGA	Kintex-7 XC7K410T
DRAM	1 GB
Baseband analog-to-digital converter (ADC) resolution	14 bit
Baseband digital-to-analog converter (DAC) resolution	16 bit
ADC spurious-free dynamic range (sFDR)	88 dB
DAC sFDR	80 dB
Maximum I/Q sample rate	200 MS/s
SFP+[1]	High speed serial link to one of the FPGA GTX transceivers

[1]Can be bypassed to the SoM if using the 10 GbE as protocol.

37.11.1

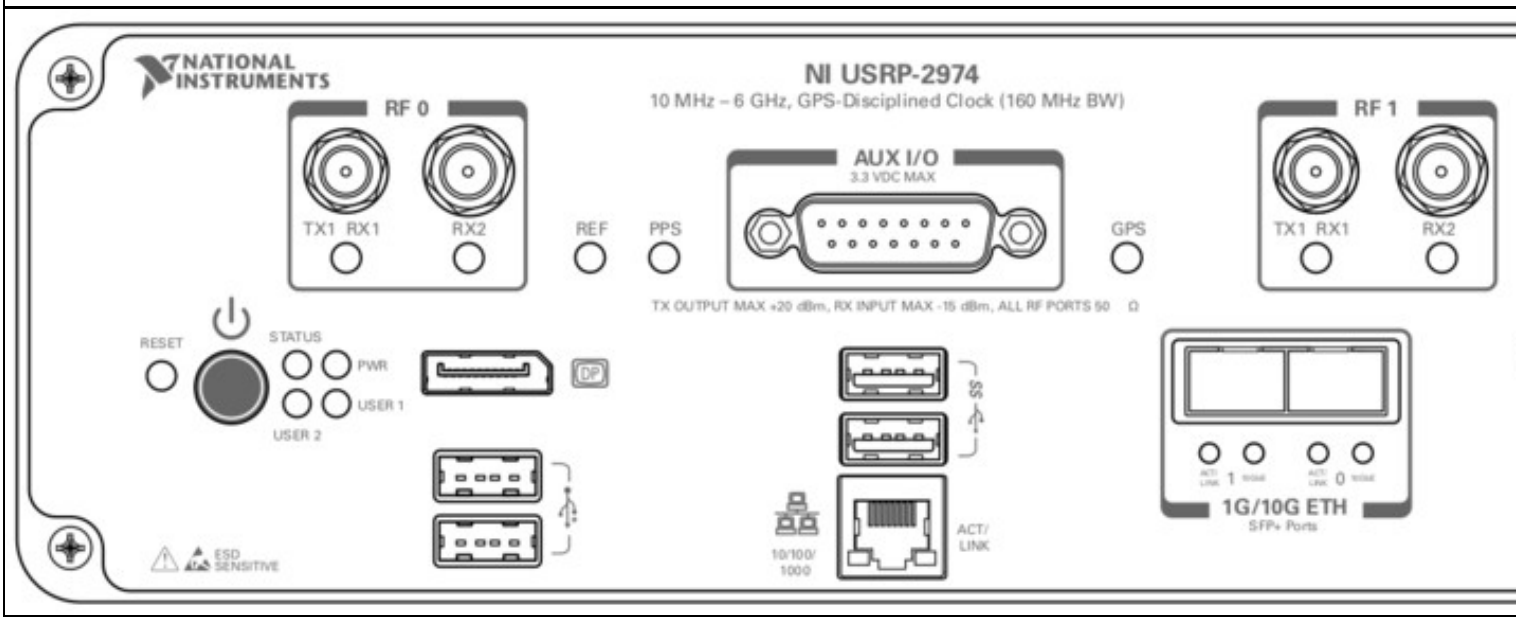
The Verilog code for the FPGA in the NI USRP-2974 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Specifically, changing the I/O interface of the FPGA in any way (do not remove any of the I/O for the PCIe interface, such as `x300_pcie_int` and `LvFpga_Chinch_Interface`), or modifying the pin and timing constraint files, could result in physical damage to other components on the motherboard, external to the FPGA, and doing this will void the warranty. Also, even if the PCIe interface is not being used, you cannot remove or reassign these pins in the constraint file. The constraint files should not be modified. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

37.12

Follow the links below for additional information on configuring each interface for the USRP-2974.

- Dual 10 Gigabit Ethernet - 200 MS/s Full Duplex @ 16-bit
- PCIe Express (Desktop) - 200 MS/s Full Duplex @ 16-bit
- 1 Gigabit Ethernet - 25 MS/s Full Duplex @ 16-bit

37.12.1

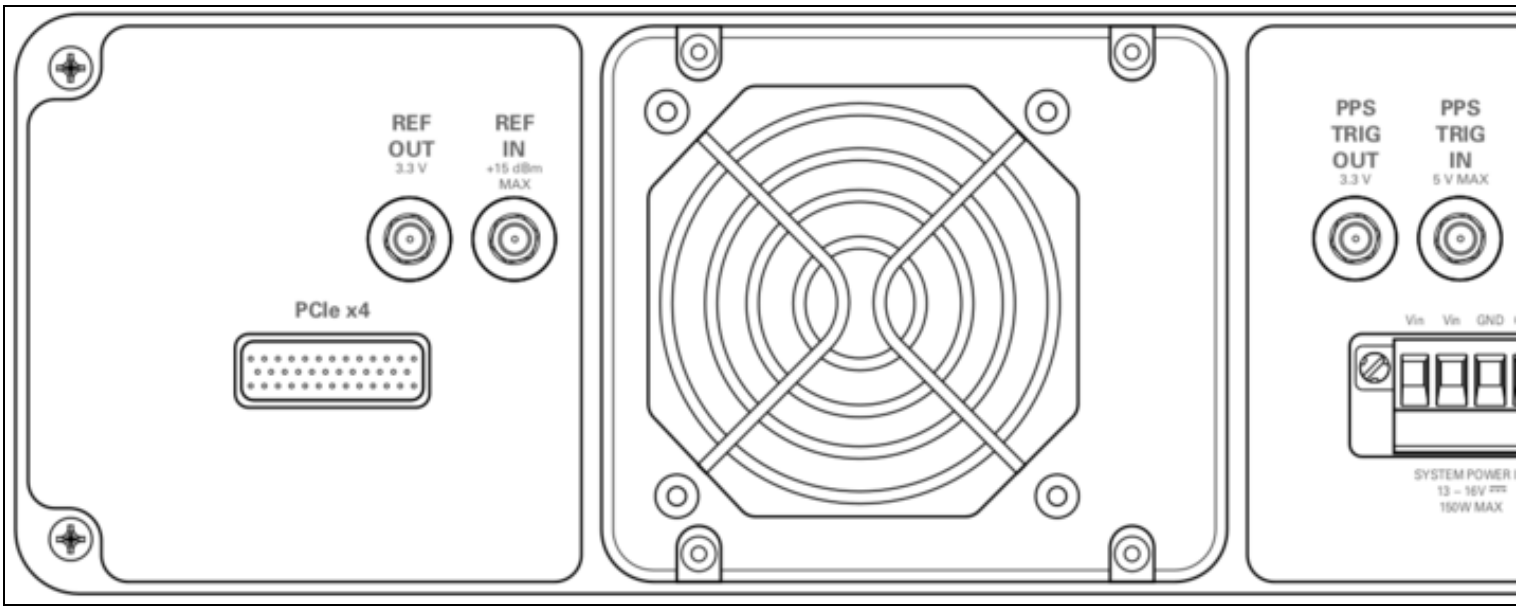


Connector		Use
RF 0	TX1 RX1	Input and output terminal for the RF signal. TX1 RX1 is an SMA (f) connector with an impedance of 50 ? and is a single-ended input or output channel.
	RX2	Input terminal for the RF signal. RX2 is an SMA (f) connector with an impedance of 50 ? and is a single-ended input channel.
AUX I/O		General-purpose I/O (GPIO) port. AUX I/O is controlled by the FPGA.
RF 1	TX1 RX1	Input and output terminal for the RF signal. TX1 RX1 is an SMA (f) connector with an impedance of 50 ? and is a single-ended input or output channel.
	RX2	Input terminal for the RF signal. RX2 is an SMA (f) connector with an impedance of 50 ? and is a single-ended input channel.
DP		DisplayPort connector to connect one monitor for your controller.
USB2.0		USB ports that support common USB peripheral devices such as flash drives, hard drives, keyboards, and mice.
USB3.0		USB ports that support common USB peripheral devices such as flash drives, hard drives, keyboards, and mice.
1G ETH		RJ45 port used for 1G ETH connectivity to other ethernet devices.
?USB		USB port used for UART connectivity to the controller.
1G/10G ETH 0		SFP+ port used for 10G ETH connectivity to other ethernet devices. Connects to the embedded Linux computer for communication with LabVIEW RT.
1G/10G ETH 1		SFP+ port used for 1G/10G ETH connectivity to other ethernet devices. Connects to the FPGA. Not currently supported in LabVIEW Communications System Design Suite.

	LED	Description	Color	State	Indication
RF 0	TX1 RX1	Indicates thetransmit status of the device	OFF	?	The device is not active.
			Red	Solid	The device is transmitting data.
			Green	Solid	The device is receiving data.
	RX2	Indicates the receive status of the device.	OFF	?	The device is not receiving data.

			Green	Solid	The device is receiving data.
REF		Indicates the status of the reference signal.	OFF	?	There is no reference signal, or the device is not locked to the reference signal.
			Green	Blinking	The device is not locked to the reference signal.
				Solid	The device is locked to the reference signal.
PPS		Indicates the pulse per second (PPS).	OFF	?	There is no PPS timing reference signal, or the device is not locked to the reference signal.
			Green	Blinking	The device is locked to the PPS timing reference signal.
GPS		Indicates whether the GPSDO is locked.	OFF	?	There is no GPSDO or the GPSDO is not locked.
			Green	Solid	The GPSDO is locked.
	TX1	Indicates thetransmit status of the device	OFF	?	The device is not active.
RF 1	RX1		Red	Solid	The device is transmitting data.
			Green	Solid	The device is receiving data.
	RX2	Indicates the receive status of the device.	OFF	?	The device is not receiving data.
			Green	Solid	The device is receiving data.
			OFF	?	The device initialized successfully and is ready for use.
Status		Indicates the status of the device	Red	Blinking	Hardware error. An internal power supply has failed. Check front-panel I/O connections for shorts. Remove any shorts and cycle power to the USRP-2974. Contact NI if the problem persists.
PWR		Indicates the power status of the device	OFF	?	The device is powered off.
			Green	Solid	The devices is powered on.
10/100/1000		Indicates the speed of the Gigabit Ethernet link.	OFF	?	No link, or 10 Mbps link.
			Green	Solid	100 Mbps link.
			Amber	Solid	1,000 Mbps link.
ACT/LINK		Indicates the Gigabit Ethernet link activity or status.	OFF	?	No link has been established.
			Green	Solid	A link has been negotiated.
				Blinking	Activity on the link.
1G/10G ETH 0	ACT/LINK	Indicates the status of the SFP+ port.	OFF	?	The link is down.
			Green	Solid	The link is up.
				Blinking	The link is active (transmitting and receiving).
	10GbE	Indicates the status of the 10G ETH link.	OFF	?	The 10G ETH link is down.
			Green	Solid	The 10G ETH link is up.
1G/10G ETH 1	10GbE	Indicates the status of the 10G ETH link.	OFF	?	The 10G ETH link is down.
			Green	Solid	The 10G ETH link is up.

37.12.2



Connector	Use
REF OUT	Output terminal for an external reference signal for the LO on the device. REF OUT is an SMA (f) connector with an impedance of 50 ?, and it is a single-ended reference output. The output signal at this connector is 10 MHz at 3.3 V.
REF IN	Input terminal for an external reference signal for the LO on the device. REF IN is an SMA (f) connector with an impedance of 50 ?, and it is a single-ended reference input. REF IN accepts a 10 MHz signal with a minimum input power of 0 dBm (0.632 Vpk-pk) and a maximum input power of 15 dBm (3.56 Vpk-pk) for a square wave or sine wave.
PPS TRIG OUT	Output terminal for the PPS timing reference. PPS TRIG OUT is an SMA (f) connector with an impedance of 50 ? and is a single-ended input. The output signal is 0 V to 3.3 V TTL. You can also use this port as a triggered output (TRIG OUT) that you program with the PPS Trig Out I/O signal.
PPS TRIG IN	Input terminal for PPS timing reference. PPS TRIG IN is an SMA (f) connector with an impedance of 50 ? and is a single-ended input channel. PPS TRIG IN accepts 0 V to 3.3 V TTL and 0 V to 5 V TTL signals. You can also use this port as a triggered input (TRIG IN) that you control using NI-USRP software.
GPS ANT	Input terminal for the GPS antenna signal. GPS ANT is an SMA (f) connector with a maximum input power of -15 dBm and an output of DC 5 V to power an active antenna. **Notice:** Do not terminate the GPS ANT port if you do not use it.
PCIe x4	Port for a PCI Express Generation 2, x4 bus connection through an MXI Express four-lane cable. Can be used to connect an external USRP device or external chassis.
SYSTEM POWER IN	Input that accepts a 15 V ± 5%, 10 A external DC power connector.

37.12.3

Using an external 10 MHz reference clock, a square wave will offer the best phase noise performance, but a sinusoid is acceptable. The power level of the reference clock cannot exceed +15 dBm.

37.12.4

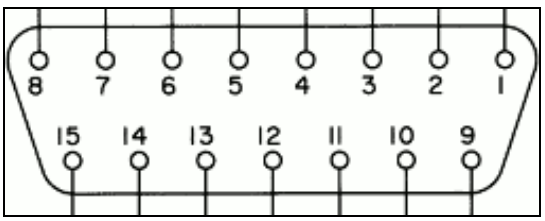
Using a PPS signal for timestamp synchronization requires a square wave signal with the following a 5Vpp amplitude.

To test the PPS input, you can use the following tool from the UHD examples:

- `<args>` are device address arguments (optional if only one USRP device is on your machine)

```
cd <install-path>/lib/uhd/examples ./test_pps_input ?args=<args>
```

37.12.5



The GPIO port is not meant to drive big loads. You should not try to source more than 5mA per pin.

The +3.3V is for ESD clamping purposes only and not designed to deliver high currents.

37.12.5.1

The hardware power on state and UHD initial state for the front-panel GPIOs is high-Z. For the X3xx, there are no external pull-ups/pull-downs for the GPIO pins, but the FPGAs do have them and they are configured as follows: X3xx: pull-down.

37.12.5.2

- Pin 1: +3.3V
- Pin 2: Data[0]
- Pin 3: Data[1]
- Pin 4: Data[2]
- Pin 5: Data[3]
- Pin 6: Data[4]
- Pin 7: Data[5]
- Pin 8: Data[6]
- Pin 9: Data[7]
- Pin 10: Data[8]
- Pin 11: Data[9]
- Pin 12: Data[10]
- Pin 13: Data[11]
- Pin 14: 0V
- Pin 15: 0V

Note: Please see the E3x0/X3x0 GPIO API for information on configuring and using the GPIO bus.

37.13

37.13.1

As of December 1st, 2010 all NI/Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

37.13.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

37.14

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

37.15

The USRP-2974 provides three interface options ? 1 Gigabit Ethernet (1 GigE), 10 Gigabit Ethernet (10 GigE), and PCI-Express (PCIe). The PCIe interface is always available regardless of what FPGA image is loaded. Ettus ships two FPGA image variants, the HG or HGS image which has one 1 GigE interfaces and one 10 GigE interfaces, and the XG image which has two 10 GigE interfaces. Generally, Ettus Research recommends using 10 GigE to achieve the maximum throughput available from the USRP-2974. PCIe is recommended for applications that require the lowest possible latency, which is a desirable characteristic for PHY/MAC research. If your application does not require the full bandwidth of the USRP-2974, the 1 GigE interface serves as a cost-effective fall-back option. Ettus Research provides a complete interface kit for each of these options, which is also shown in the following table.

Interface Performance Summary

Interface	Throughput (MS/s @ 16-bit)	Target	Recommended Kit
1 Gigabit	25 MS/s	Desktop/Laptop	SFP Adapter + GigE Cable
10 Gigabit	200 MS/s	Desktop	10 GigE Interface Kit
PCI-Express (PCIe, 4 lane)	200 MS/S	Desktop	PCI-Express Desktop Kit

37.15.1

In order to utilize the dual 10 Gigabit Ethernet interfaces, ensure the XG image is installed (see FPGA Image Flavors). In addition to burning the prerequisite FPGA image, it may also be necessary to tune the network interface card (NIC) to eliminate drops (Ds) and reduce overflows (Os). This is done by increasing the number of RX descriptors (see Linux specific notes).

The `benchmark_rate` tool can be used to test this capability. Run the following commands to test the X-series USRP over both 10 Gigabit Ethernet interfaces with the maximum rate of 200 Msps per channel:

```
cd <install-path>/lib/uhd/examples
./benchmark_rate --args="type=x300,addr=<Primary IP>,second_addr=<secondary IP>" --channels="0,1" --rx_rate 200e6
```

The second interface is specified by the extra argument **second_addr**.

Recommended 10 Gigabit Ethernet Cards

- Intel X520-DA2
  - Intel® Ethernet Converged Network Adapter X520-DA2
- Intel X520-DA1
  - Intel® Ethernet Converged Network Adapter X520-DA1
- Intel X710-DA2
  - Intel® Ethernet Converged Network Adapter X710-DA2
- Intel X710-DA4

- Intel® Ethernet Converged Network Adapter X710-DA4
- Mellanox MCX4121A-ACAT
  - Mellanox MCX4121A-ACAT

37.16

The USRP-2794 has a high-accuracy GPS-disciplined oscillator (GPSDO). The GPSDO improves the accuracy of the internal frequency reference to 20 ppb, or 0.1 ppb if the GPS is synchronized to the GPS constellation. When synchronized to the GPS constellation, all USRP? devices will also be synchronized in time within 50 ns.

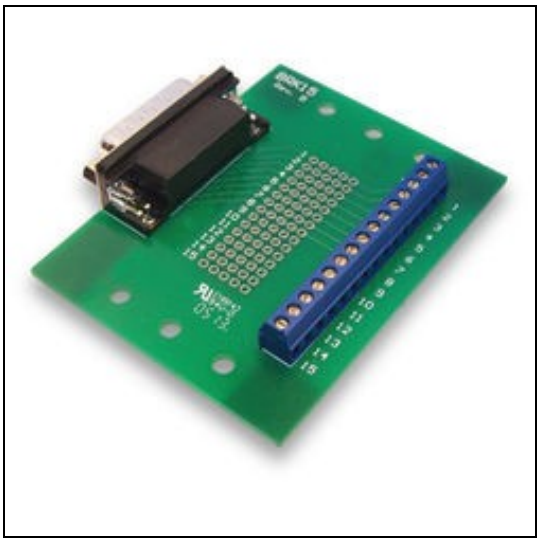
- Support GPSDO NMEA Strings
- JacksonLabs LC_XO

	Internal TCXO	GPS-Disciplined Clock
Frequency Reference	TCXO	OCXO
Frequency Accuracy	± 2.5ppm ± 2,500 Hz @ 1 GHz	± 25 ppb ± 25 Hz @ 1 GHz
Frequency Accuracy (GPS-Disciplined)		± 0.01ppb ~ ± 0.01 Hz @ 1 GHz
GPS Time Sync Accuracy		±50ns to UTC Time**
10 MHz Reference Phase Drift with GPS Sync		<±20ns After 1 Hour**

37.16.1

You can query the lock status with the `gps_locked` sensor, as well as obtain raw NMEA sentences using the `gps_gprmc`, and `gps_gpgga` sensors. Location information can be parsed out of the `gps_gpgga` sensor by using `gpsd` or another NMEA parser.

37.17



This General Purpose Input/output (GPIO) breakout kit provides access to general purpose digital I/O signals with simple terminal blocks, and a prototyping area where wires and components can be soldered. Each GPIO pin is connected to an FPGA digital line allowing it to be configured as an input, or an output, using the various software frameworks that support the USRP? GPIO.

These GPIO signals can serve the following functions:

- Control of external devices, such as power amplifiers and RF switches
- Provide output signals that can help with debugging
- Provide observables to be analyzed by oscilloscopes or other external equipment
- Accept input from external devices for local, software-based triggering
- Implement a protocol line such as SPI or I2C

37.17.1

- 1 GPIO Breakout Board
- 1 DB-15, 1-meter cable
- GPIO Quick Reference

37.17.2

The GPIO signals exposed with this breakout kit are routed directly to the USRP device's FPGA with limited protection circuitry. However, the user must take precautionary measures to ensure input/output signals meet the specifications shown in this document. Over voltage, excess current draw, and other conditions can damage the USRP device and void the warranty. Special care should be taken when the USRP is powered off.

37.17.3

The GPIO breakout board can be mounted directly to the DB15 connector of a USRP ? device, or mounted remotely with the cable provided in this kit. The screws on the DB15 connector of the breakout board must be removed to mount the board directly. For remote mounting, the breakout board is supplied with rubber standoffs to avoid scratching surfaces, and several through-holes for hard mounting with screws or other hardware (not provided).

37.17.4

When used with UHD, or other third party frameworks that leverage UHD, the GPIO expansion can be controlled with simple API calls. For more information, on the C++ API, and examples of how to use the GPIO in frameworks such as GNU Radio, please see the Application Notes section of the

37.17.5

Parameter	Typical
Configured as Input	
Default Voltage Standard	3.3V LVCMOS
Voltage High Threshold	2.0V
Voltage Low Threshold	0.8V
Voltage Input Limits (no damage)	-0.3V/3.45V
Configured as Output	
Voltage Standard	3.3V LVCMOS
Voltage High Output	2.8V
Voltage Low Output	0.4V
Current Source Capability	12 mA
Output Source Impedance	>33 ohms typical

37.18

The GPSDO Mini Kit will improve the accuracy of the USRP reference clock, even if it does not receive signals from the GPS Constellation. However, to achieve the best accuracy possible, and to achieve global timing alignment across multiple USRPs, Ettus Research recommends the GPSDO Mini Antenna Kit.

37.19

Multiple USRP-2974s can be synchronized for coherent operation by sharing a common 10 MHz and 1 PPS signal. We recommend using a star-distribution topology with an OctoClock or OctoClock-G, as seen in Figure 4. This requires matched length cables to be used for both 10 MHz and 1 PPS.

For more information about MIMO operation, please see the MIMO and Synchronization Application Note.

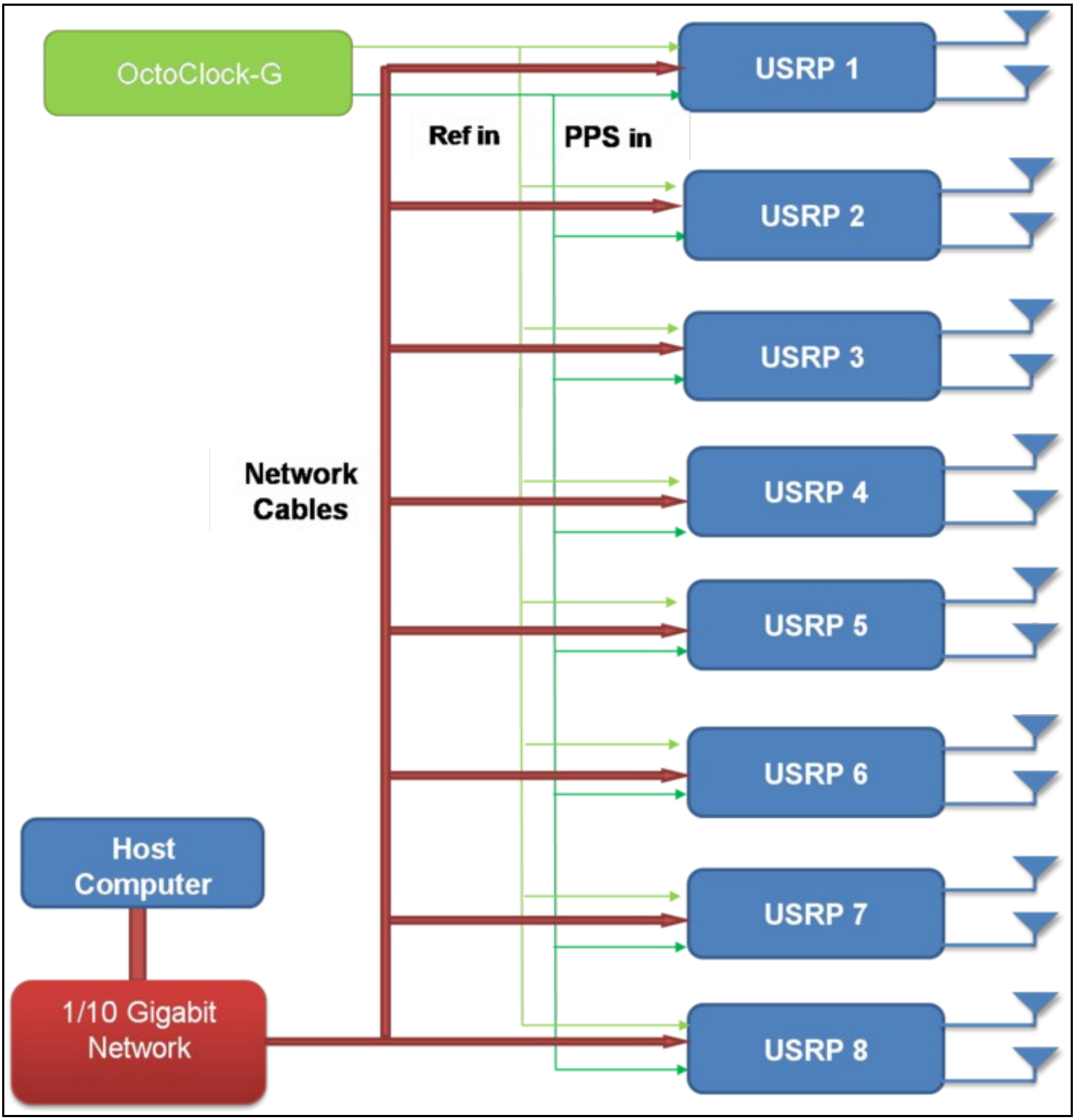


Figure 4 - Star-Distribution of 10 MHz/PPS Signals with OctoClock

37.20

- **What is the bandwidth of the USRP-2974**

The ADC rate on each analog RX channel is 200 MS/s quadrature, which provides a theoretical analog bandwidth of approximately 80% of the Nyquist bandwidth of +/- 100 MHz (+/- 80 MHz around the center frequency). The resulting maximum theoretical analog bandwidth is 160 MHz.

FPGA Processing Bandwidth: Up to 200 MS/s quadrature.

Host Bandwidth: Up to 200 MS/s quadrature, dependent on selected interface

For more information about achieving the maximum bandwidth with a USRP-2974, please see the "USRP X300/X310 Configuration Guide" or the "USRP System Bandwidth" application note.

- **How can I program the USRP-2974**

Like all other USRP models, the USRP-2974 is compatible with the USRP Hardware Driver? (UHD) architecture. The UHD architecture is a common driver that allows users to develop and execute applications on the onboard or host computer. UHD provides a direct C++ API to control and stream to/from the USRP-2974. It also provides compatibility with a variety of third-party software frameworks including GNU Radio, LabVIEW, and MATLAB. You may also customize the FPGA image provided with UHD to integrate your own signal processing. For more information about UHD, and supported software frameworks, please see:

http://files.ettus.com/manual/

- **How do I update the FPGA images and firmware with the latest from UHD**

You can find more information about updating the FPGA image through PCIe, 1/10 GigE, and JTAG here.

- **How can I modify the FPGA of the USRP-2974**

The source code (Verilog) for the USRP-2794 is available in the UHD repository. The build process leverages the existing CMAKE build system used to compile the host-side driver. A Linux-based setup will provide the best results.

Which FPGA toolchain required to build the FPGA images will depend upon your version of UHD. For more details please see the UHD Software Resource page.

- **How much free space is available in the USRP-2974 FPGA**

Please see the #Utilization statistics section of this resources page for more information.

- **What frequency range does the USRP-2974 cover**

10MHz to 6GHz.

- **What components do I need to purchase for a complete USRP-2974 system**

The USRP-2974 is a complete stand alone SDR. Additional components might include RF filters, antennas, RF power amplifiers or other RF components needed for a specific application.

38 X410

38.1

When you receive a brand-new device, it is strongly recommended that you download the most recent filesystem image from the Ettus Research website and write it to the NI Ettus USRP X410. Instructions on downloading the latest filesystem image and writing it to the X410 is described in the USRP X410 Getting Started Guide.

Note that if you are operating the device in Network Mode, then the versions of UHD running on the host computer and on the NI Ettus USRP X410 device must match.

38.2

The NI Ettus USRP X410 is a high-performance, multi-channel software defined radio. The SDR is designed for frequencies from 1 MHz to 7.2 GHz, tunable up to 8 GHz and features a two-stage superheterodyne architecture with 4 independent TX and RX channels capable of 400 MHz of instantaneous bandwidth each. Digital interfaces for data offload and control include two QSFP28 interfaces capable of 100 GbE[1], a PCIe Gen3 x8 [3] interface, as well standard command, control, and debug interfaces: USB-C JTAG, USB-C console, Ethernet 10/100/1000. The USRP X410 is an all-in-one device built on the Xilinx Zynq Ultrascale+ ZU28DR RF System on Chip (RFSoC) with built-in digital up and down conversion and onboard Soft-Decision Forward Error Correction (SD-FEC) IP.

38.3

38.3.1

- High channel density
- Reliable and fault-tolerant deployment
- Stand-alone (embedded) or host-based (network streaming) operation
- Fully integrated and assembled (the USRP X410 does not support swappable daughtercards)
- 1 MHz to 7.2 GHz frequency range (tunable up to 8GHz)
- Up to 400 MHz of instantaneous bandwidth per channel
- 4 RX, 4 TX in half-wide RU form factor
- Xilinx Zynq-Ultrascale+ ZU28DR RFSoC
- 12 bit ADC, 14 bit DAC
- IQ Sample Clock rates up to 500 MS/s
- Onboard SD-FEC, DDC, DUC
- Quad-core ARM Cortex-A53 up to 1.2 GHz CPU
- Dual-core ARM Cortex-A5 MPCore up to 500 MHz
- Two QSFP28 ports (10 Gigabit Ethernet, 100 Gigabit Ethernet, Aurora)
- Two iPass+? zHD® Interfaces (PCIe Gen3 x 8)
- RJ45 (1 GbE) [1]
- 10 MHz Clock reference
- PPS time reference
- Trig In/Out Interface
- Built-in GPSDO
- Two FPGA Programmable GPIO Interfaces (HDMI)
- 1 Type C USB host port
- 1 Type C USB port (serial console, JTAG)
- Watchdog timer
- OpenEmbedded Linux
- USRP Hardware Driver? (UHD) open-source software API version 4.1.0 or later
- RF Network on Chip (RFNoC?) FPGA development framework
- Xilinx Vivado® 2019.1 Design Suite (license not included)
- GNU Radio support maintained by Ettus Research? through GR-UHD, an interface to UHD distributed by GNU

Radio
- [1] The RJ45 port is used for remote management of the device and does not support IQ streaming.

38.4

https://www.ni.com/pdf/manuals/378493a.pdf

38.4.1

- Current Hardware Revision: Module revision E and Motherboard revision G
- Minimum version of UHD required: 4.1.0.5

https://github.com/EttusResearch/uhd/releases/tag/v4.1.0.5

38.4.2

If you want any CAD / STP models beyond those found here, please send an email to Ettus Support at support@ettus.com noting your request and your use case for any such model. We will determine on a case-by-case basis whether we have any such requested model and, if so, whether to release it -- possibly requiring an NDA for any such release. Note that we do not have models on all USRPs and daughterboards, and requesting any model does not guarantee that either Ettus Research or NI will honor any such request.

38.5

38.5.1

The Verilog code for the FPGA in the NI Ettus USRP X410 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Specifically, changing the I/O interface of the FPGA in any way, or modifying the pin and timing constraint files, could result in physical damage to other components on the motherboard, external to the FPGA, and doing this will void the warranty. Also, even if the PCIe interface is not being used, you cannot remove or reassign these pins in the constraint file. The constraint files should not be modified. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

38.6

38.6.1

38.6.2

38.6.3

Using an external 10 MHz reference clock, a square wave will offer the best phase noise performance, but a sinusoid is acceptable.

38.6.4

Using a PPS signal for timestamp synchronization requires a square wave signal (a typical PPS signal has a 20%-25% duty cycle) with a 5Vpp amplitude.

To test the PPS input, you can use the following tool from the UHD examples:

- `<args>` are device address arguments (optional if only one USRP device is on your machine)

```
cd <install-path>/lib/uhd/examples ./test_pps_input ?args=<args>
```

38.6.5

- https://files.ettus.com/manual/page_x400_gpio_api.html#x4x0gpio_fpanel

38.6.5.1

- https://kb.ettus.com/USRP_X410_Getting_Started_Guide#Autoboot
- https://files.ettus.com/manual/page_usrp_x4xx.html#x4xx_usage_rearpanelleds_power

38.7

For datasheet, drawings, pricing, and purchasing please search for the Part Number listed below via https://www.ni.com/en-us/search.html

- Dual 100 Gigabit Ethernet PCIe Interface Kit For Ettus USRP X4xx
  - Part Number: 788216-01
- USRP X4xx Power Supply, 100-240VAC 50/60HZ 12VDC 24AMP
  - Part Number: 788204-01
- QSFP28 To 4xSFP28 Breakout Cable, 1M
  - Part Number: 788214-01
- QSFP28 Twinaxial Cable, 3M
  - Part Number: 788215-03
- USRP X4xx 19" Rack Mount Accessory, 1U, 2 USRP X4xx Devices, Shoulder to Shoulder
  - Part Number: 788147-01
- USRP X4xx Desktop Stack Accessory, Single USRP X4xx Device Fastened Buildup

        - Part Number: 788148-01

- USRP X4xx 19" Rack Mount Accessory, 1U, 1 USRP X4xx Device, w/Surrogate Extension
    - Part Number: 788149-01

- GPIO Communication Cable
    - SHH19-H19-AUX Shielded Single-Ended Cable, 1M
        - Part Number: 152629-01
    - SHH19-H19-AUX Shielded Single-Ended Cable, 2M
        - Part Number: 152629-02

- SCB-19 Noise Rejecting, Shielded Aux I/O Connector
    - Part Number: 782444-01

- Fan Replacement Cartridges
    - USRP X4xx Fan Cartridge Accessory, Exhaust
        - Part Number: 788164-01
    - USRP X4xx Fan Cartridge Accessory, Intake
        - Part Number: 788165-01

38.8

Recommended 10 Gigabit Ethernet Cards

- Intel X710-DA2
    - Intel® Ethernet Converged Network Adapter X710-DA2
- Intel X710-DA4
    - Intel® Ethernet Converged Network Adapter X710-DA4

38.9

- Requires UHD 4.2 or later: https://github.com/EttusResearch/uhd/releases/tag/v4.2.0.0
- 100GbE Streaming only supports Linux Hosts

Recommended 100 Gigabit Ethernet Bundles

- Dual 100 Gigabit Ethernet PCIe Interface Kit, NIC and Cable
    - ni.com part number 788216-01

Recommended 100 Gigabit Ethernet Cards

- Mellanox/NVIDIA ConnectX-5 EX 100 GbE NIC (MCX516A-CCAT (PCIe Gen3 x16))
- Mellanox/NVIDIA ConnectX-5 EX 100 GbE NIC (MCX516A-CDAT (PCIe Gen4 x16))

Recommended 100 Gigabit Ethernet Cables

- Mellanox/NVIDIA 3m QSFP28 MCP1600-C003E26N
    - Shorter length variants also recommended

Recommended Host PC

- At least 15 CPU Cores
- At least 32 GB RAM
- Ubuntu 20.04 (5.4.0-89-generic kernel)

Validated Hardware and Software Configuration Examples

- Ubuntu 20.04 (5.4.0-89-generic kernel), DPDK 19.11, with Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz - 24 CPU - 4.8 GHz Max CPU freq - 64 GB RAM. Mellanox/NVIDIA ConnectX-5 EX 100 GbE NIC (MCX516A-CCAT (PCIe Gen3 x16)). Mellanox/NVIDIA 3m QSFP28 MCP1600-C003E26N cables.

- Ubuntu 20.04 (5.4.0-89-generic kernel), DPDK 20.11, with AMD Ryzen Threadripper 3960X 24-Core Processor - 48 CPU - 3.6 GHz CPU freq - 64 GB RAM. Mellanox/NVIDIA ConnectX-5 EX 100 GbE NIC (MCX516A-CDAT (PCIe Gen4 x16)). Mellanox/NVIDIA 3m QSFP28 MCP1600-C003E26N cables.

Data Throughput Rates

Testing was completed with the following conditions

- Hardware and Software Configurations listed above
- CPU configured for performance mode: https://kb.ettus.com/USRP_Host_Performance_Tuning_Tips_and_Tricks#CPU_Governor
- DPDK Setup: https://files.ettus.com/manual/page_dpdk.html and https://kb.ettus.com/Getting_Started_with_DPDK_and_UHD#UHD_4.0
- Enabling Tx pause Frames on X410 for the SFP port(s) utilized for streaming: https://files.ettus.com/manual/page_transport.html#transport_udp_linux
    - `ethtool -A sfp0 tx on`
    - `ethtool -A sfp1 tx on`
- uhd.conf: See https://files.ettus.com/manual/page_dpdk.html#dpdk_nic_config

Executing benchmark_rate over multiple iterations as well as over an extended continuous time period (>12 Hours) without data loss resulted in the following maximum rates and channel counts

- Usage of the `--priority` argument set to `high` in benchmark_rate which requires benchmark rate to be executed with root privileges via `sudo`
- Usage of single versus multiple threads in the benchmark_rate utility - controlled by using the `--multi_streamer` argument. Specifying this argument assigns one thread per channel being streamed.
- Utilizing the CG_400 bitfile which does not include DUC/DDC resampling thus supports only rates of 491.52 MS/s and 500 MS/s

- Dual Port DPDK
    - 4 Rx + 4 Tx @ 491.52 MS/s

        - 4 Rx @ 491.52 MS/s
        - 4 Tx @ 491.52 MS/s
    - Single Port DPDK
        - 2 Rx + 2 Tx @ 491.52 MS/s
        - 4 Rx @ 491.52 MS/s
        - 3 Tx @ 491.52 MS/s
    - Dual Port Non-DPDK
        - 1 Rx + 1 Tx @ 491.52 MS/s
        - 4 Rx @ 491.52 MS/s
        - 2 Tx @ 491.52 MS/s
    - Single Port Non-DPDK
        - 1 Rx + 1 Tx @ 491.52 MS/s
        - 3 Rx @ 491.52 MS/s
        - 2 Tx @ 491.52 MS/s

38.10

Ettus Research currently offers direct-connect, copper cabling accessories for the NI Ettus USRP X410. However, it is also possible to use multi-mode fiber instead of copper connections for these devices. In this section, we will provide general guidance on the types of fiber adapters and cables that can be used with these products.

The NI Ettus USRP X410 is compatible with most brands of SFP+ fiber adapters. In some cases, other equipment in the systems such as 1/10/100 Gigabit Ethernet switches are only compatible with specific brands of SFP+ adapters and cables. As a general rule, we recommend checking compatibility with the switches and network cards in your system before purchasing an adapter.

Ettus Research does test the NI Ettus USRP X410 devices with the listed hardware as noted in the above section https://kb.ettus.com/X410#100_Gigabit_Ethernet

38.11

38.11.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

38.11.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

38.12

Found on the NI Product Certifications lookup tool here.

38.13

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

39 X440

39.1

When you receive a brand-new device, it is strongly recommended that you download the most recent filesystem image from the Ettus Research website and write it to the NI Ettus USRP X4x0. Instructions on downloading the latest filesystem image and writing it to the X4x0 are described in the USRP X4x0 Getting Started Guide.

Note that if you are operating the device in Network Mode, then the versions of UHD running on the host computer and on the NI Ettus USRP X4x0 device must match.

39.2

39.2.1

The NI Ettus USRP X440 is the widest bandwidth USRP software defined radio device. It differs architecturally from most USRPs, utilizing a direct sampling architecture that provides balun-coupled access to the ADCs and DACs on the onboard Xilinx Zynq RFSoC. This makes it well suited to use as an intermediate frequency transceiver, connecting to external front ends for applications like satellite communications (SATCOM) prototyping, SATCOM ground station deployment, and mmWave or sub-THz 6G research. USRP X440 features high channel density ? 8 Tx and 8 Rx channels per device ? with phase coherency across channels by sharing sample clocks. Therefore, it is ideal for applications like direction finding and radar research and prototyping.

39.3

39.3.1

- High channel density
- Reliable and fault-tolerant deployment
- Stand-alone (embedded) or host-based (network streaming) operation
- Fully integrated and assembled (the USRP X440 does not support swappable daughtercards)
- 30 MHz to 4 GHz frequency range (tunable down to 1MHz)
- Up to 1600 MHz of instantaneous bandwidth per channel
- 8 RX, 8 TX in half-wide RU form factor
- Xilinx Zynq-Ultrascale+ ZU28DR RFSoC
- 12 bit ADC, 14 bit DAC
- IQ Sample Clock rates up to 2000 MS/s
- Onboard SD-FEC, DDC, DUC
- Quad-core ARM Cortex-A53 up to 1.2 GHz CPU
- Dual-core ARM Cortex-A5 MPCore up to 500 MHz
- Two QSFP28 ports (10 Gigabit Ethernet, 100 Gigabit Ethernet)
- RJ45 (1 GbE) [1]
- 10 MHz Clock reference
- PPS time reference
- Trig In/Out Interface
- Built-in GPSDO
- Two FPGA Programmable GPIO Interfaces (HDMI)
- 1 Type C USB host port
- 1 Type C USB port (serial console, JTAG)
- Watchdog timer
- OpenEmbedded Linux
- USRP Hardware Driver? (UHD) open-source software API version 4.5.0 or later
- RF Network on Chip (RFNoC?) FPGA development framework
- Xilinx Vivado® 2021.1 Design Suite (license not included)
- GNU Radio support maintained by Ettus Research? through GR-UHD, an interface to UHD distributed by GNU

Radio
- [1] The RJ45 port is used for remote management of the device and does not support IQ streaming.

39.4

39.4.1

- Current Hardware Revision: Module revision D and Motherboard revision F
- Minimum version of UHD required: 4.5.0
- USRP X440 is covered by Export Administration Regulations (EAR) NS2. Refer to US Department of Commerce for details and country chart.

https://www.ni.com/docs/en-US/bundle/ettus-usrp-x440-specs/page/specs.html https://github.com/EttusResearch/uhd/releases/tag/v4.5.0.0

39.4.2

If you want any CAD / STP models beyond those found here, please send an email to Ettus Support at support@ettus.com noting your request and your use case for any such model. We will determine on a case-by-case basis whether we have any such requested model and, if so, whether to release it -- possibly requiring an NDA for any such release. Note that we do not have models on all USRPs and daughterboards, and requesting any model does not guarantee that either Ettus Research or NI will honor any such request.

39.5

39.5.1

The Verilog code for the FPGA in the NI Ettus USRP X4x0 is open-source, and users are free to modify and customize it for their needs. However, certain modifications may result in either bricking the device, or even in physical damage to the unit. Specifically, changing the I/O interface of the FPGA in any way, or modifying the pin and timing constraint files, could result in physical damage to other components on the motherboard, external to the FPGA, and doing this will void the warranty. Also, even if the PCIe interface is not being used, you cannot remove or reassign these pins in the constraint file. The constraint files should not be modified. Please note that modifications to the FPGA are made at the risk of the user, and may not be covered by the warranty of the device.

39.6

39.6.1

39.6.1.1

39.6.2

39.6.3

Using an external 10 MHz reference clock, a square wave will offer the best phase noise performance, but a sinusoid is acceptable.

39.6.4

Using a PPS signal for timestamp synchronization requires a square wave signal (a typical PPS signal has a 20%-25% duty cycle) with a 5 Vpp amplitude.

To test the PPS input, you can use the following tool from the UHD examples:

- `<args>` are device address arguments (optional if only one USRP device is on your machine)

```
cd <install-path>/lib/uhd/examples ./test_pps_input ?args=<args>
```

39.6.5

- https://files.ettus.com/manual/page_x400_gpio_api.html#x4x0gpio_fpanel

39.6.5.1

- https://kb.ettus.com/USRP_X410/X440_Getting_Started_Guide#Autoboot
- https://files.ettus.com/manual/page_usrp_x4xx.html#x4xx_usage_rearpanelleds_power

39.7

For datasheet, drawings, pricing, and purchasing please search for the Part Number listed below via https://www.ni.com/en-us/search.html

- Dual 100 Gigabit Ethernet PCIe Interface Kit For Ettus USRP X4xx
    - Part Number: 788216-01
- USRP X4xx Power Supply, 100-240VAC 50/60HZ 12VDC 24AMP
    - Part Number: 788204-01
- QSFP28 To 4xSFP28 Breakout Cable, 1M
    - Part Number: 788214-01
- QSFP28 Twinaxial Cable, 3M
    - Part Number: 788215-03
- USRP X4xx 19" Rack Mount Accessory, 1U, 2 USRP X4xx Devices, Shoulder to Shoulder
    - Part Number: 788147-01
- USRP X4xx Desktop Stack Accessory, Single USRP X4xx Device Fastened Buildup
    - Part Number: 788148-01
- USRP X4xx 19" Rack Mount Accessory, 1U, 1 USRP X4xx Device, w/Surrogate Extension
    - Part Number: 788149-01
- GPIO Communication Cable
    - SHH19-H19-AUX Shielded Single-Ended Cable, 1M
        - Part Number: 152629-01
    - SHH19-H19-AUX Shielded Single-Ended Cable, 2M
        - Part Number: 152629-02
- SCB-19 Noise Rejecting, Shielded Aux I/O Connector
    - Part Number: 782444-01
- Fan Replacement Cartridges
    - USRP X4xx Fan Cartridge Accessory, Exhaust
        - Part Number: 788164-01
    - USRP X4xx Fan Cartridge Accessory, Intake
        - Part Number: 788165-01

39.8

Recommended 10 Gigabit Ethernet Cards

- Dual 10 Gigabit Ethernet Interface for Ettus USRP
    - ni.com part number 788600-01
- Intel X710-DA2
    - Intel® Ethernet Converged Network Adapter X710-DA2
- Intel X710-DA4
    - Intel® Ethernet Converged Network Adapter X710-DA4

39.9

39.9.1

- Requires UHD 4.5 or later:?https://github.com/EttusResearch/uhd/releases/tag/v4.5.0.0
- 100GbE Streaming only supports Linux Hosts

Recommended 100 Gigabit Ethernet Cards

- Mellanox/NVIDIA ConnectX-5 EX 100 GbE NIC (MCX516A-CDAT (PCIe Gen4 x16))

Recommended 100 Gigabit Ethernet Cables

- Mellanox/NVIDIA 3m QSFP28 MCP1600-C003E26N
    - Shorter length variants also recommended

Recommended Host PC

- At least 32 CPU Cores
- At least 64 GB RAM
- Ubuntu 20.04 (5.13.0-44-generic)

Validated Hardware and Software Configuration Examples

- Ubuntu 20.04 (5.13.0-44-generic kernel), DPDK 20.11, with AMD Ryzen Threadripper 3960X 24-Core Processor - 48 CPU - 3.6 GHz CPU freq - 64 GB RAM. Mellanox/NVIDIA ConnectX-5 EX 100 GbE NIC (MCX516A-CDAT (PCIe Gen4 x16)). Mellanox/NVIDIA 3m QSFP28 MCP1600-C003E26N cables.

Data Throughput Rates

Testing was completed with the following conditions

- Hardware and Software Configurations listed above
- CPU configured for performance mode: https://kb.ettus.com/USRP_Host_Performance_Tuning_Tips_and_Tricks#CPU_Governor
- DPDK Setup: https://files.ettus.com/manual/page_dpdk.html and https://kb.ettus.com/Getting_Started_with_DPDK_and_UHD#UHD_4.0
- Enabling Tx pause Frames on X4x0 for the SFP port(s) utilized for streaming: https://files.ettus.com/manual/page_transport.html#transport_udp_linux

        - `ethtool -A sfp0 tx on`
        - `ethtool -A sfp1 tx on`
    - uhd.conf: See https://files.ettus.com/manual/page_dpdk.html#dpdk_nic_config

Executing benchmark_rate over multiple iterations as well as over an extended continuous time period (>12 Hours) without data loss resulted in the following maximum rates and channel counts

- Usage of the `--priority` argument set to `high` in benchmark_rate which requires benchmark rate to be executed with root privileges via `sudo`
- Usage of single versus multiple threads in the benchmark_rate utility - controlled by using the `--multi_streamer` argument. Specifying this argument assigns one thread per channel being streamed.
- Utilizing the CG_400 and CG_1600 bitfile.

- CG_400
    - Dual Port DPDK
        - 6 Rx @ 500 MS/s
        - 8 Rx @ 400 MS/s
        - 6 Tx @ 500 MS/s
        - 8 Tx @ 450 MS/s
        - 4 Rx + 4 Tx @ 500 MS/s
        - 8 Rx + 8 Tx @ 250 MS/s
- CG_1600
    - Dual Port DPDK
        - 2 Rx @ 1000 MS/s
        - 2 Tx @ 1800 MS/s
        - 2 Rx + 2 Tx @ 1000 MS/s

All testing was done using dual 100GbE ports. For a single port, expect around half the number of channels as compared to the dual port equivalent configuration at the same streaming rate.

39.10

Ettus Research currently offers direct-connect, copper cabling accessories for the NI Ettus USRP X4x0. However, it is also possible to use multi-mode fiber instead of copper connections for these devices. In this section, we will provide general guidance on the types of fiber adapters and cables that can be used with these products.

The NI Ettus USRP X4x0 is compatible with most brands of SFP+ fiber adapters. In some cases, other equipment in the systems such as 1/10/100 Gigabit Ethernet switches are only compatible with specific brands of SFP+ adapters and cables. As a general rule, we recommend checking compatibility with the switches and network cards in your system before purchasing an adapter.

Ettus Research does test the NI Ettus USRP X4x0 devices with the listed hardware as noted in the above section https://kb.ettus.com/X440#100_Gigabit_Ethernet

39.11

39.11.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

39.11.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

39.12

Certifications will soon be findable on the NI Product Certifications lookup tool.

The Letter of Volatility Link.

39.13

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

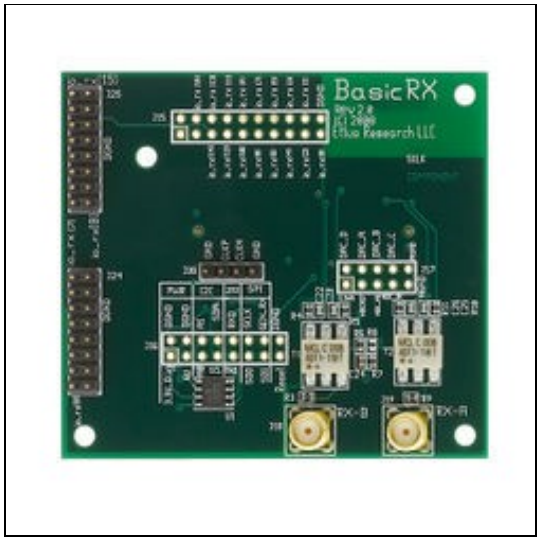
40 BasicRX

40.1

The BasicRX/BasicTX daughterboards are low-cost daughterboards that provides direct access to the ADC inputs. The boards accept real-mode only signals from 1 to 250 MHz. The BasicRX/BasicTX is ideal for applications using an external front end providing relatively clean signals within operable bandwidth. Wideband transformers couple each RF input to a single channel of the USRP device's ADC. The signals sampled by the ADC are manipulated in the FPGA, and the outputs can be processed independently, or as a single I/Q pair. The BasicTX/BasicRX daughterboards are supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.

40.2

- 1-250 MHz coverage
- Real or Complex sampling



40.3

40.3.1

BasicRX

The BasicRX has 4 frontends:

- **Frontend A**: real signal on antenna RXA
- **Frontend B**: real signal on antenna RXB
- **Frontend AB**: quadrature frontend using both antennas (IQ)
- **Frontend BA**: quadrature frontend using both antennas (QI)

BasicTX

The BasicTX has 4 frontends:

- **Frontend A**: real signal on antenna TXA
- **Frontend B**: real signal on antenna TXB
- **Frontend AB**: quadrature frontend using both antennas (IQ)
- **Frontend BA**: quadrature frontend using both antennas (QI)

40.3.2

BasicRX

- The BasicRX has no tunable elements or programmable gains. Through the magic of aliasing, you can down-convert signals greater than the Nyquist rate of the ADC.

BasicTX

- The BasicTX has no tunable elements or programmable gains. Through the magic of aliasing, you can up-convert signals greater than the Nyquist rate of the DAC.

40.3.3

BasicRX

- **For Real-Mode (A or B frontend)**: 250 MHz
- **For Complex (AB or BA frontend)**: 500 MHz

BasicTX

- **For Real-Mode (A or B frontend)**: 250 MHz
- **For Complex (AB or BA frontend)**: 500 MHz

40.3.4

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

40.3.5

- The maximum input power for the BasicRX is +10 dBm.

40.4

- Ettus Research recommends to always use the latest stable version of UHD

40.4.1

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

40.4.2

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

40.5

40.5.1

- 0-40 °C

40.5.2

- 10% to 90% non-condensing

40.6

40.6.1

- N or X Series

40.6.2

- N or X Series

40.7

40.7.1

Basic RX Schematics

40.7.2

Basic TX Schematics

40.8

Part Number	Description	Schematic ID (Page)
24LC025B	EEPROM	U1 (1); U52 (1)
ADT1?1WT	RF Transformer	T1 (1); T2 (1); T50 (1); T51 (1)

40.9

40.9.1

- BasicRX - File:cu ettus-cca-basic-rx.pdf

40.10

- The BasicTX/RX daughterboard features female SMA connectors for both the RX and TX connectors.

40.11

40.11.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

40.11.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

40.12

BasicRX / BasicTX Letter of Volatility

40.13

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

41 CBX

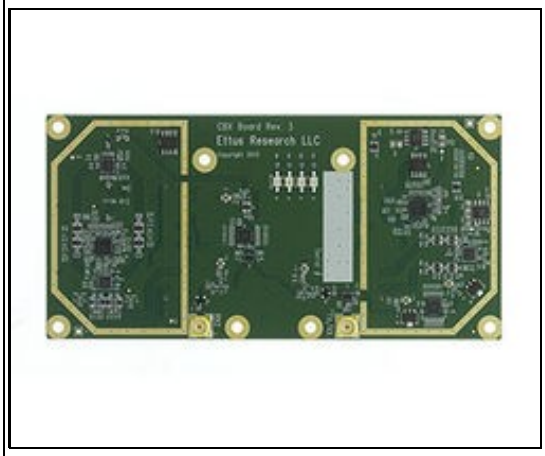
41.1

The CBX is a full-duplex, wideband transceiver that covers a frequency band from 1.2 GHz to 6 GHz with a instantaneous bandwidth of 40 MHz or 120 MHz. The CBX can serve a wide variety of application areas, including WiFi research, cellular base stations, cognitive radio research, and RADAR. The CBX daughterboard is supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.

The CBX does not provided phase coherent operation, and therefore is not recommended for MIMO and Phased Array applications.

41.2

- Frequency Range: 1.2GHz - 6GHz
- Versions: 40MHz / 120MHz



41.3

41.3.1

- 2 quadrature frontends (1 transmit, 1 receive)
  - Defaults to direct conversion
  - Can be used in low IF mode through lo_offset with uhd::tune_request_t
- Independent receive and transmit LO's and synthesizers
  - Allows for full-duplex operation on different transmit and receive frequencies
  - Can be set to use Integer-N tuning for better spur performance with uhd::tune_request_t

41.3.2

Transmit: **TX**/**RX**

Receive: **TX**/**RX** or **RX2**

- **Frontend 0:** Complex baseband signal for selected antenna
- **Note:** The user may set the receive antenna to be TX/RX or RX2. However, when using a CBX board in full-duplex mode, the receive antenna will always be set to RX2, regardless of the settings.

41.3.3

- Transmit Gains: **PGA0**, Range: 0-31.5dB
- Receive Gains: **PGA0**, Range: 0-31.5dB

41.3.4

- CBX: 40 MHz, RX & TX
- CBX-120: 120 MHz, RX & TX

41.3.5

- **lo_locked:** boolean for LO lock state

41.3.6

- All LEDs flash when daughterboard control is initialized
- **TX LD**: Transmit Synthesizer Lock Detect
- **TX**/**RX**: Receiver on TX/RX antenna port (No TX)
- **RX LD**: Receive Synthesizer Lock Detect
- **RX1**/**RX2**: Receiver on RX2 antenna port

41.4

41.4.1

- 1.2GHz - 6GHz

41.4.2

- 5 - 7.5 dB @ (1.2GHz ~ 5GHz)
- 7.5dB - 10 dB (5GHz ~ 6GHz)

41.4.3

- 8 - 10 dBm

41.4.4

- -20 dBc

41.4.5

- 22 dBm @ (1.2GHz ~ 3GHz)
- 12 ~ 22 dBm @ (3GHz ~ 6GHz)

41.4.6

- 30 - 32 dBm @ (1.2GHz ~ 5GHz)
- 26 ~ 30 dBm @ (5GHz ~ 6GHz)

41.4.7

- -20 dBc

41.4.8

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

41.4.9

- The maximum input power for the CBX is -15 dBm.

41.5

- CBX without UHD Corrections

41.6

- Ettus Research recommends to always use the latest stable version of UHD

41.6.1

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

41.7

41.7.1

- 0-40 °C

41.7.2

- 10% to 90% non-condensing

41.8

41.8.1

- N or X Series

41.8.2

- X Series only

41.9

The CBX daughterboard is not capable of phase-synchronous operation. The SBX, UBX, TwinRX daughterboards are recommended for phase-coherent applications.

41.10

41.10.1

CBX Schematics

41.11

Part Number	Description	Schematic ID (Page)
VMMK-3603	Low Noise Amplifier	U1, U5 (1)
AS225-313LF	SPDT Switch	U3, U6 (1)
HMC624LP4E	ATTENUATOR	U7, U2 (1)
MGA82563	Amplifier	U4 (1)
GVA-84+	Amplifier	U9 (1)
PHA-1+	Amplifier	U8 (1)
ADL5380ACPZ	Quadrature Demodulator	U11 (2)
ADA4927-2YCPZ	Differential ADC Driver	U10 (2)
AD8591ARTZ-REEL	Amplifiers	U31 (2)
NC7WZ04P6X	Dual Inverter	U26 (3); U15, U16 (4); U21 (5); U27 (6)
MAX2870ETJ+	Fractional/Integer-N Synthesizer	U23 (3); U24 (6)
SKY13267-321	Diversity Switch	U12 (3); U25 (6)
LFCN-2000+	Low Pass Filter	FL13 (3); FL12 (6)
LP3878MR-ADJ	Voltage Regulator	U13, U14 (4); U19, U20 (5)
24LC024	EEPROM	U17 (4); U22 (5)
ADL5375-05	Quadrature Modulator	U18 (5)

41.12

- File:cu ettus-cca-cbx.pdf

41.13

- The CBX daughterboard features female SMA connectors for both the TX/RX and RX2 connectors.

41.14

41.14.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

41.14.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

41.15

41.15.1

- Media:volatility UBX CBX WBX SBX r1 1.pdf

41.16

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

42 LFRX

42.1

The LFTX daughterboard utilizes two high-speed operational amplifiers to allow transmission from 0-30 MHz. The boards accept real-mode only signals. The LFTX is ideal for applications in the HF band, or for applications using an external front end to up-convert and amplify the intermediate signal. The outputs of the LFTX can be processed independently, or as a single I/Q pair. Example applications include HF communications, radios with external front ends and direct signal generation below 30 MHz. The LFTX/LFRX daughterboards are supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.

42.2

- DC-30Mhz coverage

42.3

42.3.1

LFRX

The LFRX has 4 frontends:

- **Frontend A**: real signal on antenna RXA
- **Frontend B**: real signal on antenna RXB
- **Frontend AB**: quadrature frontend using both antennas (IQ)
- **Frontend BA**: quadrature frontend using both antennas (QI)

LFTX

The LFTX has 4 frontends:

- **Frontend A**: real signal on antenna TXA
- **Frontend B**: real signal on antenna TXB
- **Frontend AB**: quadrature frontend using both antennas (IQ)
- **Frontend BA**: quadrature frontend using both antennas (QI)

42.3.2

LFRX

- The LFRX has no tunable elements or programmable gains. Through the magic of aliasing, you can down-convert signals greater than the Nyquist rate of the ADC.

LFTX

- The LFTX has no tunable elements or programmable gains. Through the magic of aliasing, you can up-convert signals greater than the Nyquist rate of the DAC.

42.3.3

LFRX

- **For Real-Mode (A or B frontend)**: 33 MHz
- **For Complex (AB or BA frontend)**: 66 MHz

LFTX

- **For Real-Mode (A or B frontend)**: 33 MHz
- **For Complex (AB or BA frontend)**: 66 MHz

42.3.4

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

42.3.5

- The maximum input power for the LFRX is +10 dBm.

42.4

- Ettus Research recommends to always use the latest stable version of UHD

42.4.1

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

42.4.2

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

42.5

42.5.1

- 0-40 °C

42.5.2

- 10% to 90% non-condensing

42.6

42.6.1

- N or X Series

42.6.2

- N or X Series

42.7

42.7.1

LFRX Schematics

42.7.2

LFTX Schematics

42.8

Part Number	Description	Schematic ID (Page)
AD813x	Differential ADC Driver	U2, U3 (1)
LT3462	DC/DC Converter	U3 (1)
24LC025B	EEPROM	U1 (1)

42.9

- The LFTX / LFRX daughterboard features female SMA connectors for both the TX and RX connectors.

42.10

42.10.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

42.10.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

42.11

LFRX / LFTX Letter of Volatility

42.12

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

43 SBX

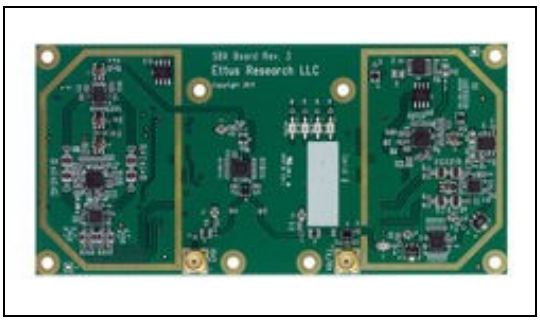
43.1

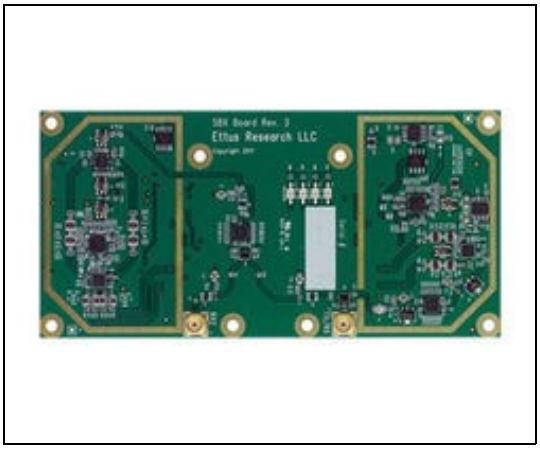
The SBX is a wide bandwidth transceiver that provides up to 100 mW of output power, and a typical noise figure of 5 dB. The local oscillators for the receive and transmit chains operate independently, which allows dual-band operation. The SBX is MIMO capable, and provides 40 MHz or 120 MHz of bandwidth. The SBX is ideal for applications requiring access to a variety of bands in the 400 MHz-4400 MHz range. Example application areas include WiFi, WiMax, S-band transceivers and 2.4 GHz ISM band transceivers. The SBX daughterboard is supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.

The SBX is capable of phase coherent operation, and therefore is suitable for MIMO and Phased Array applications.

43.2

- Frequency Range: 400MHz - 4.4GHz
- Versions: 40MHz / 120MHz
- Power Output: 100mW





43.3

43.3.1

- 2 quadrature frontends (1 transmit, 1 receive)
  - Defaults to direct conversion
  - Can be used in low IF mode through lo_offset with uhd::tune_request_t
- Independent receive and transmit LO's and synthesizers
  - Allows for full-duplex operation on different transmit and receive frequencies
  - Can be set to use Integer-N tuning for better spur performance with uhd::tune_request_t

43.3.2

Transmit: **TX**/**RX**

Receive: **TX**/**RX** or **RX2**

- **Frontend 0:** Complex baseband signal for selected antenna
- **Note:** The user may set the receive antenna to be TX/RX or RX2. However, when using an SBX board in full-duplex mode, the receive antenna will always be set to RX2, regardless of the settings.

43.3.3

- Transmit Gains: **PGA0**, Range: 0-31.5dB
- Receive Gains: **PGA0**, Range: 0-31.5dB

43.3.4

- SBX: 40 MHz, RX & TX
- SBX-120: 120 MHz, RX & TX

43.3.5

- **lo_locked:** boolean for LO lock state

43.3.6

- All LEDs flash when daughterboard control is initialized
- **TX LD**: Transmit Synthesizer Lock Detect
- **TX**/**RX**: Receiver on TX/RX antenna port (No TX)
- **RX LD**: Receive Synthesizer Lock Detect
- **RX1**/**RX2**: Receiver on RX2 antenna port

43.4

43.4.1

- 400MHz - 4.4GHz

43.4.2

- 4 - 5 dB @ (400MHz ~ 1.5GHz)
- 5 - 7 dB @ (1.5GHz ~ 3GHz)
- 7 - 12dB @ (3GHz ~ 4.4GHz)

43.4.3

- 16 ~ 22 dBm

43.4.4

- -20 dBc @ (400MHz ~ 600MHz)
- -30dBc @ (600MHz ~ 4.4GHz)

43.4.5

- 20 dBm @ (400MHz ~ 3.5GHz)
- 18 - 20dBm @ (3.5GHz ~ 4.4GHz)

43.4.6

- 26 - 30dBm @ (400MHz ~ 1.5GHz)
- 30 dBm @ (1.5GHz ~ 4.4GHz)

43.4.7

- -20 dBc @ (400MHz ~ 600MHz)
- -30dBc @ (600MHz ~ 4.4GHz)

43.4.8

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

43.4.9

- The maximum input power for the SBX is -15 dBm.

43.5

- SBX without UHD Corrections

43.6

- Ettus Research recommends to always use the latest stable version of UHD

43.6.1

- Current Hardware Revision: 5.1
- Minimum version of UHD required: 3.8.0

43.7

43.7.1

- 0-40 °C

43.7.2

- 10% to 90% non-condensing

43.8

43.8.1

- N or X Series

43.8.2

- X Series only

43.9

The SBX daughterboard is capable of phase-synchronous operation, and is recommended for phase-coherent applications. The UBX and TwinRX daughterboards are also recommended for phase-coherent applications.

43.10

43.10.1

SBX Schematics

43.11

Part Number	Description	Schematic ID (Page)
MGA82563	Amplifier	U1, U5, U4 (1)
AS225-313LF	SPDT Switch	U3, U6 (1)
HMC624LP4E	ATTENUATOR	U2, U7 (1)
LFCN-5850+	Low Pass Filter	FL1 (1)
PHA-1+	Amplifier	U8 (1)
GVA-84+	Amplifier	U9 (1)
ADL5380ACPZ	Quadrature Demodulator	U11 (2)
ADA4927-2YCPZ	Differential ADC Driver	U10 (2)
AD8591ARTZ-REEL	Amplifiers	U31 (2)
ADF4350BCPZ	Synthesizer with Integrated VCO	U23 (3); U24 (6)
SKY13267	Diversity Switch	U12 (3); U25 (6)
LFCN-1200+	Low Pass Filter	FL13 (3); FL12 (6)
TC1-1-43A+	RF Transformer	T3 (2); T2, T5 (3); T4, T6 (6)
LP3878MR-ADJ	Voltage Regulator	U13, U14 (4); U19, U20 (5)
NC7WZ04P6X	Dual Inverter	U15, U16 (4); U21 (5); U27 (6); U26 (3)
24LC024	EEPROM	U17 (4); U22 (5)
ADL5375	Quadrature Modulator	U18 (5)

43.12

- File:cu ettus-cca-sbx.pdf

43.13

- The SBX daughterboard features female SMA connectors for both the TX/RX and RX2 connectors.

43.14

43.14.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

43.14.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

43.15

43.15.1

- Media:volatility UBX CBX WBX SBX r1 1.pdf

43.16

FPGA Resources

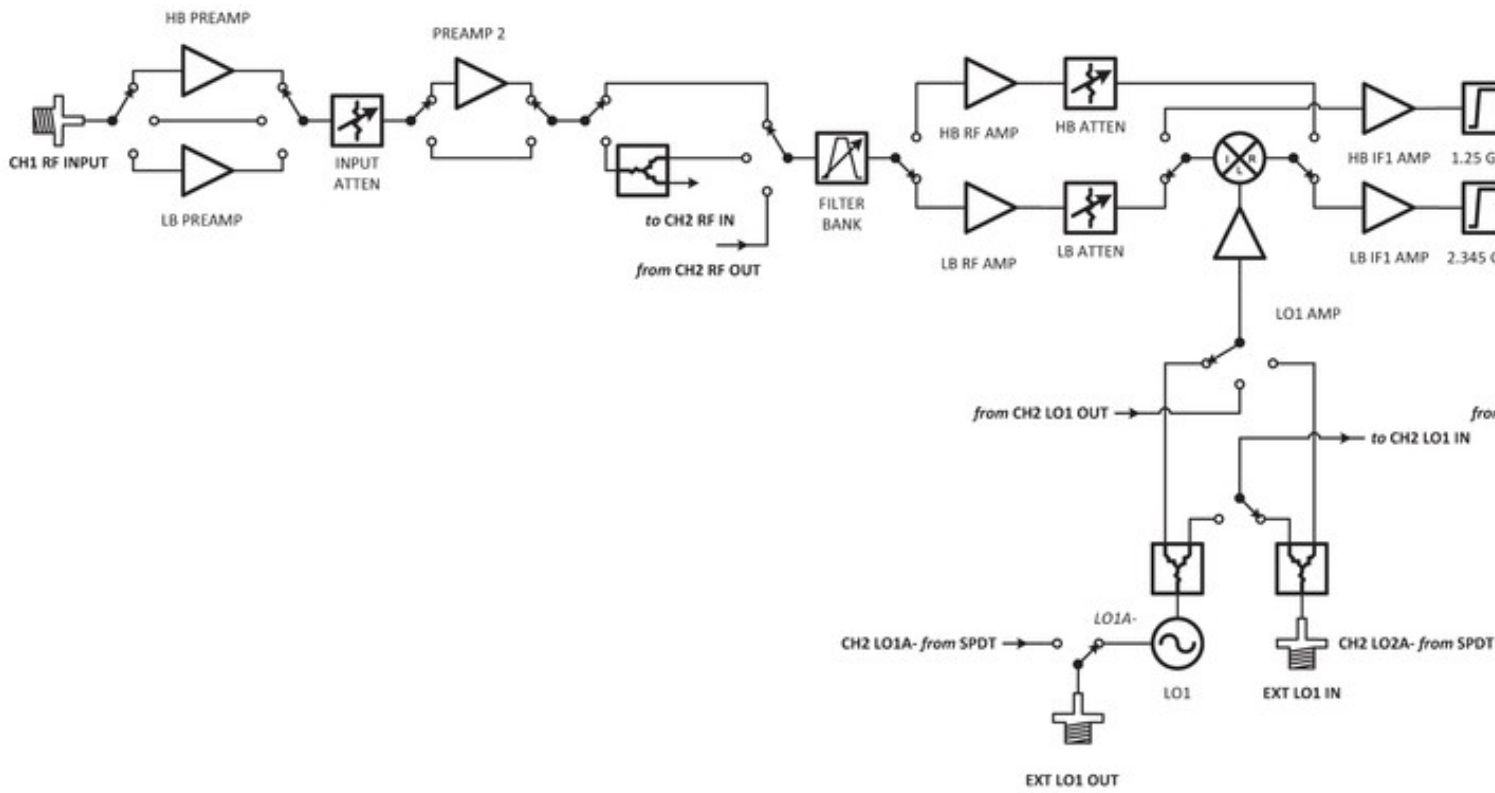
UHD Stable Binaries

UHD Source Code on Github

44 TwinRX

44.1

The TwinRX daughterboard for the USRP X300 and X310 SDR platform is a two-channel superheterodyne receiver designed for high performance spectrum monitoring and direction finding applications. The receiver is tunable from 10 MHz - 6 GHz and has 80 MHz of instantaneous bandwidth per channel, providing the versatility necessary to analyze a variety of signals in multiple bands of interest. Each channel has an independent RF signal chain with preampilifiers, preselectors, and two mixer stages for superior selectivity. Users can tune the two channels independently to simultaneously monitor uplink and downlink communication with a combined bandwidth of 160 MHz. The ability to share the LO between channels across multiple daughterboards enables the phase-aligned operation required to implement scalable multi-channel phased-arrays. The receiver is capable of fast frequency hopping to detect frequency agile emitters. Configurable RF attenuation and preamplication allow users to optimize dynamic range in favor of noise figure for faint signals, or IP3 for stronger signals. UHD automatically configures the RF signal path for optimized performance in the pre-defined use cases, and provides the flexibility to adjust settings manually. Support for RFNoC on the X Series motherboard enables deterministic FPGA-accelerated computations for real-time spectrum analysis. The TwinRX daughterboard is supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.



44.2

- Two-Channel Superheterodyne Receiver
- Frequency Range: 10 MHz - 6 GHz
- Bandwidth: 80 MHz per channel (160 MHz total)
- RF shielding
- Independent RF signal channels with optional LO sharing

44.3

44.3.1

- 2 superheterodyne frontends (2 receive)
- 80 MHz per channel
- Independent tuning
- LO Sharing Capability
- Coherent and phase-aligned operation
- Preselection Filters
- RF Shielding

44.3.2

Receive: **RX1** or **RX2**

44.3.3

- Receive Gains Range: 0-93dB

44.3.4

- TwinRX: 80 MHz per channel (160 MHz total)

44.3.5

- **lo_locked:** boolean for LO lock state

44.3.6

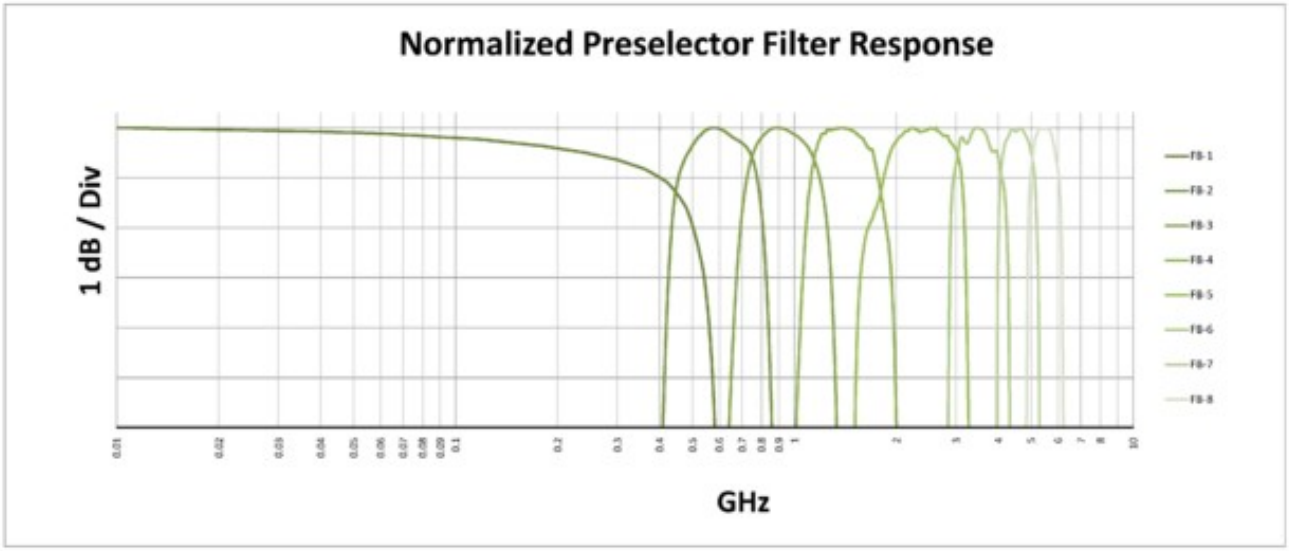
The TwinRX has six MMCX RF connectors on it.

Connector	Description	Min	Nominal	Damage
J1	LO2 Export	0 dBm	3 dBm	NA (Output)
J2	LO2 Input	0 dBm	2 dBm	20dBm
J3	LO1 Export	-12 dBm	5 dBm	NA (Output)
J4	LO1 Input	-10 dBm	-5 dBm	10dBm
J5	Antenna 1 connector RX 1			10 dBm
J6	Antenna 2 connector RX 2			10 dBm

44.3.7

Band	Range
LB1	10 - 500 MHz
LB2	500 - 800 MHz
LB3	800 - 1.2 GHz
LB4	1.2 - 1.8 GHz
HB1	1.8 - 3.0 GHz
HB2	3.0 - 4.1 GHz
HB3	

	4.1 - 5.1 GHz
HB4	5.1 - 6.0 GHz



- Each preselector filter has a ±40 MHz band overlap.

44.4

44.4.1

- 10MHz - 6GHz

44.4.2

Frequency	Preamp Enabled
10 MHz - 3 GHz	< 5
3 GHz - 5 GHz	< 4
5 GHz - 6 GHz	< 8

44.4.3

Frequency	Full Scale = -45 dBm	Full Scale = -30 dBm	Full Scale = -20 dBm
10 MHz - 1.8 GHz	-8	-2	16
1.8 GHz - 3 GHz	-10	-1	14
3 GHz - 6 GHz	-13	-1	12

44.4.4

Frequency Offset	0.9 GHz	2.4 GHz	5.8 GHz
10 kHz	-88	-86	-82
100 kHz	-105	-107	-103
1 MHz	-124	-127	-127

44.4.5

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

44.4.6

- The maximum input power for the TwinRX is +10 dBm.

44.5

- Ettus Research recommends to always use the latest stable version of UHD. Minimum UHD version is 3.10.0.0

44.5.1

- Current Hardware Revision: 2
- Minimum version of UHD required: 3.10.0.0
- Minimum version of GNU Radio required: 3.7.10

44.6

44.6.1

- 0-40 °C

44.6.2

- 10% to 90% non-condensing

44.7

- X300 and X310 USRP only
- `master_clock_rate` of 200e6 only. The TwinRX daughterboard does not operate with the master clock rate of 184.32e6; operation with this master clock rate will result in UHD errors such as the following:

```
[ERROR] [DBMGR] The daughterboard manager encountered a recoverable error in init.
Loading the "unknown" daughterboard implementations to continue.
The daughterboard cannot operate until this error is resolved.
ValueError: TwinRX clock rate 92160000.000000 is not a multiple of the pfd freq 12500000.000000.
```

44.8

The TwinRX daughterboard is capable of phase-synchronous operation, and is recommended for phase-coherent applications. Please note that the TwinRX is receive-only. The SBX and UBX daughterboards are also recommended for phase-coherent applications, and are capable of both transmit and receive operations.

44.9

The TwinRX daughterboard is composed of two PCBs, the Intermediate Frequency (IF) board and the RF board. There are two functionally identical revisions of TwinRX at the moment. Please check the back of your TwinRX to see which you have. The schematics for TwinRX revision B should answer most possible questions. If you have any further questions please email support@ettus.com.

TwinRX Revision A - (159685A-01)

- IF Board Revision B - (158671B-01L)
- RF Board Revision C - (156263C-01L)

TwinRX Revision B - (159685B-01)

- IF Board Revision C - (158671C-01L)
- RF Board Revision D - (156263D-01L)

File:TwinRX IF Board Rev C.pdf

File:TwinRX RF Board Rev D.pdf

44.10

The antenna ports are MMCX connectors with 50 ohm input impedance. By default Antenna 1 (RX1) is routed to Channel 1 and Antenna 2 (RX2) to Channel 2. This routing can be changed to swap the antennas or to share a single antenna to both channels. The damage threshold for the antenna inputs is 10 dBm. In practice the available gain makes much lower input powers recommended for achieving the best dynamic range and noise figure.

44.11

Note: LO sharing cables are not required for a single TwinRX setup. LO sharing cables are only required with two TwinRX daughterboards in a single USRP X300/X310.

Connector	Description	Min	Nominal	Damage
J1	LO2 Export	0 dBm	3 dBm	NA (Output)
J2	LO2 Input	0 dBm	2 dBm	20dBm
J3	LO1 Export	-12 dBm	5 dBm	NA (Output)
J4	LO1 Input	-10 dBm	-5 dBm	10dBm
J5				10 dBm

J6	Antenna 1 connector RX 1 Antenna 2 connector RX 2	10 dBm

44.11.1

TwinRX (A Slot)	TwinRX (B Slot)
J1 LO2 Export	J2 LO2 Input
J2 LO2 Input	J1 LO2 Export
J3 LO1 Export	J4 LO1 Input
J4 LO1 Input	J3 LO1 Export

44.12

44.12.1

The advanced functionalities of the TwinRX will be exposed through new functions implemented in Multi-USRP.

44.12.2

The two channels of the TwinRX can be independently configured to use either of the two antenna ports, **RX1** and **RX2** using the standard antenna selection function in `multi_usrp`.

```
virtual void set_rx_antenna(const std::string &ant, size_t chan = 0) = 0;
```

Select the RX antenna on the frontend.

- **ant** the antenna name
- **chan** the channel index 0 to N-1

44.12.3

The TwinRX has two channels, **CH1** and **CH2** and each channel has two local oscillators, **LO1** and **LO2**. The local oscillators for a channel can be sourced from that channel's **internal** synthesizers, the **companion** channel's synthesizers, or **external** inputs. The value `multi_usrp::ALL_LOS` can be used to specify that the command be run on both synthesizers for a channel. The defaults are to operate on

```
virtual std::vector<std::string> get_rx_lo_names(size_t chan = 0) = 0;
```

Get a list of possible LO stage names

- **chan** the channel index 0 to N-1
- Returns a vector of strings for possible LO names

```
virtual void set_rx_lo_source(const std::string &src, const std::string &name = ALL_LOS, size_t chan = 0) = 0;
```

Set the LO source for the usrp device. For USRPs that support selectable LOs, this function allows switching between them. Supported options for source: internal, external, companion.

- **src** a string representing the LO source
- **name** the name of the LO stage to update
- **chan** the channel index 0 to N-1

```
virtual const std::string get_rx_lo_source(const std::string &name = ALL_LOS, size_t chan = 0) = 0;
```

Get the currently set LO source.

- **name** the name of the LO stage to query
- **chan** the channel index 0 to N-1
- Returns the configured LO source

```
virtual std::vector<std::string> get_rx_lo_sources(const std::string &name = ALL_LOS, size_t chan = 0) = 0;
```

Get a list of possible LO sources.

- **name** the name of the LO stage to query
- **chan** the channel index 0 to N-1
- Returns a vector of strings for possible settings

```
virtual double set_rx_lo_freq(double freq, const std::string &name, size_t chan = 0) = 0;
```

Set the RX LO frequency.

- **freq** the frequency to set the LO to
- **name** the name of the LO stage to update
- **chan** the channel index 0 to N-1
- Returns a coerced LO frequency

```
virtual double get_rx_lo_freq(const std::string &name, size_t chan = 0) = 0;
```

Get the current RX LO frequency.

- **name** the name of the LO stage to query
- **chan** the channel index 0 to N-1
- Returns the configured LO frequency

```
virtual freq_range_t get_rx_lo_freq_range(const std::string &name, size_t chan = 0) = 0;
```

Get the LO frequency range of the RX LO.

- **name** the name of the LO stage to query
- **chan** the channel index 0 to N-1
- Returns a frequency range object

44.12.4

```
virtual void set_rx_lo_export_enabled(bool enabled, const std::string &name = ALL_LOS, size_t chan = 0) = 0;
```

Set whether the LO used by the usrp device is exported For USRPs that support exportable LOs, this function configures if the LO used by chan is exported or not.

- **enabled** if true then export the LO
- **name** the name of the LO stage to update
- **chan** the channel index 0 to N-1 for the source channel

```
virtual bool get_rx_lo_export_enabled(const std::string &name = ALL_LOS, size_t chan = 0) = 0;
```

Returns true if the currently selected LO is being exported.

- **name** the name of the LO stage to query
- **chan** the channel index 0 to N-1

44.13

- File:cu usrp twinrx cca.pdf

44.14

44.14.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

44.14.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

44.15

TwinRX Letter of Volatility

44.16

FPGA Resources

UHD Stable Binaries

45 UBX

45.1

The UBX daughterboard is a full-duplex wideband transceiver that covers frequencies from 10 MHz to 6 GHz. Coherent and phase-aligned operation across multiple UBX daughterboards on USRP X Series motherboards enables users to explore MIMO and direction finding applications. The UBX daughterboard is supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.

The UBX is capable of phase coherent operation, and therefore is suitable for MIMO and Phased Array applications, on the X Series. Additionally this capability is only available on the X Series devices.

45.2

- Frequency Range: 10 MHz - 6 GHz
- Versions: 40MHz / 160MHz
- RF shielding
- Full duplex operation with independent TX and RX frequencies
- Synthesizer synchronization for applications requiring coherent or phase-aligned operation, supported on USRP X Series motherboards only

45.3

45.3.1

- 2 quadrature frontends (1 transmit, 1 receive)
  - Defaults to direct conversion
  - Can be used in low IF mode through lo_offset with uhd::tune_request_t
- Independent receive and transmit LO's and synthesizers
  - Allows for full-duplex operation on different transmit and receive frequencies
  - Can be set to use Integer-N tuning for better spur performance with uhd::tune_request_t

45.3.2

Transmit: **TX**/**RX**

Receive: **TX**/**RX** or **RX2**

- **Frontend 0:** Complex baseband signal for selected antenna
- **Note:** The user may set the receive antenna to be TX/RX or RX2. However, when using a UBX board in full-duplex mode, the receive antenna will always be set to RX2, regardless of the settings.

45.3.3

- Transmit Gains: **PGA0**, Range: 0-31.5dB
- Receive Gains: **PGA0**, Range: 0-31.5dB

45.3.4

- UBX: 40 MHz, RX & TX
- UBX-160: 160 MHz, RX & TX

45.3.5

- **lo_locked:** boolean for LO lock state

45.3.6

- **LOCK**: Synthesizer Lock Detect
- **TX**/**RX TXD**: Transmitting on TX/RX antenna port
- **TX**/**RX RXD**: Receiving on TX/RX antenna port
- **RX2 RXD**: Receiving on RX2 antenna port

45.4

45.4.1

- 10MHz - 6GHz

45.4.2

- 10 MHz - 500 MHz: 3 - 4 dB
- 500 MHz - 1.5 GHz: 2 - 3 dB
- 1.5GHz - 6GHz: 4 - 10 dB

45.4.3

- 10 MHz - 6 GHz: 8 - 9 dBm

45.4.4

- 10 MHz - 6 GHz: < -30dBc

45.4.5

- 10 MHz - 3 GHz: 20 dBm
- 3 - 6 GHz: 8 - 20 dBm

45.4.6

- 10 - 500 MHz: 41 dBm
- 0.5 - 3 GHz: 36 dBm
- 3 - 6 GHz: 26 dBm - 36 dBm

45.4.7

- 10 MHz - 6 GHz: < -30 dBc

Note: The UBX 160 transmitter path has 160 MHz of bandwidth throughout the full frequency range of the device; the receiver path has 84 MHz of bandwidth for center frequencies from 10 MHz to 500 MHz.

45.4.8

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

45.4.9

- The maximum input power for the UBX is -15 dBm.

45.5

- UBX without UHD Corrections

45.6

- Ettus Research recommends to always use the latest stable version of UHD

45.6.1

- Current Hardware Revision: 2
- Minimum version of UHD required for UBX rev-1: 3.8.2
- Minimum version of UHD required for UBX rev-2: 3.9.5

45.6.2

- Current Hardware Revision: 2
- Minimum version of UHD required for UBX rev-1: 3.8.2
- Minimum version of UHD required for UBX rev-2: 3.9.5

45.7

45.7.1

- 0-40 °C

45.7.2

- 10% to 90% non-condensing

45.8

45.8.1

- N or X Series

45.8.2

- X Series only

45.9

The UBX daughterboard is capable of phase-synchronous operation, and is recommended for phase-coherent applications. The SBX and TwinRX daughterboards are also recommended for phase-coherent applications.

If you are operating the UBX at frequencies below 1 GHz and need phase synchronization, then it is necessary to select a 20 MHz daughterboard clock rate, instead of using the default 50 MHz rate. Note that this is only required for phase synchronization below 1 GHz. The UBX can still operate below 1 GHz without setting this lower daughterboard clock rate, but it will operate without any phase synchronization capability.

If you're using a UHD program, then you can specify the lower daughterboard clock rate on the command line of the program, with `--args="dboard_clock_rate=20e6"`.

If you're using the UHD API from a C++ program, then you can include `"dboard_clock_rate=20e6"` in the device arguments parameter when first invoking `multi_usrp::make()`.

If you're using GNU Radio, then you can add `"dboard_clock_rate=20e6"` to the `"Device Arguments"` field of the properties for the UHD Sink and UHD Source blocks.

45.10

45.10.1

UBX Schematics

45.11

Part Number	Description	Schematic ID (Page)
MAX2871	Fractional/Integer-N Synthesizer/VCO	U3 (3); U9 (5); U19 (7); U23 (9)
ADL5375-05	Quadrature Modulator	U22 (8)
LFCN-2250+	Low Pass Filter	F1 (3); F24 (7); F34, F35 (10)
LTC5510	Active Mixer	U15 (6)
LFCN-490+	Low Pass Filter	F12 (5); F15 (6); F26 (7); F31 (9); F33, F36 (10)
HMC624LP4E	ATTENUATOR	U16 (6)
NBB-400	Amplifier	U13 (6); U30 (11)
PHA-1+	Amplifier	U31 (11)
ADA4927-2	Differential ADC Driver	U6 (4)
ADL5380	Quadrature Demodulator	U8 (4)
MGA-62563	Low Noise Amplifier	U36 (11)
LFCN-1700+	Low Pass Filter	F41 (11)
VMMK-3603	Low Noise Amplifier	U34 (11)
LFCN-2600+	Low Pass Filter	F14, F17 (6)
855916	SAW Filter	F16 (6)
LTC5510	Active Mixer	U15 (6); U28 (10)
LFCN-2600+	Low Pass Filter	F14, F17 (10)
TCM1-63AX+	RF Transformer	T1 (3); T2, T3 (4); T7 (8)
ADA4927-2	Differential ADC Driver	U6 (4)
AD8591	Operational Amplifiers	U7 (4)
ADL5380	Quadrature Demodulator	U8 (4)
ZXTC2062E6	TRANSISTORS	Q1 (6)
HMC624ALP4E	ATTENUATOR	U16 (6); U29 (10)
LFCN-800+	Low Pass Filter	F2 (3); F25 (7)
ADP7104-3.3	CMOS LDO	U4, U5 (3); U10, U11 (5); U20, U21 (7); U24, U25 (9); U48 (13)
ADL5375-05	Quadrature Modulator	U22 (8)
LTC5510	Active Mixer	U28 (10)
24LC024	EEPROM	U38, U39 (12)
ADP7104-5.0	CMOS LDO	U41, U42, U43, U44, U45, U46, U47 (13)
ZXTC2062E6	TRANSISTORS	Q2, Q3, Q4, Q5 (13)

45.12

45.12.1

- PDF Format
- STP Format

45.13

- The UBX daughterboard features female SMA connectors for both the TX/RX and RX2 connectors.

45.14

45.14.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

45.14.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

45.15

45.15.1

- Media:volatility UBX CBX WBX SBX r1 1.pdf

45.16

- A larger 24W (6V, 4A) power supply is required when using a UBX-40 daughterboard and integrated GPS Disciplined Oscillator accessory together in a USRP2, USRP N200, or USRP N210 device.
- The UBX-160 transmitter path has 160 MHz of bandwidth throughout the full frequency range of the device; the receiver path has 84 MHz of bandwidth for center frequencies from 10 MHz to 500 MHz.

45.17

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

46 WBX

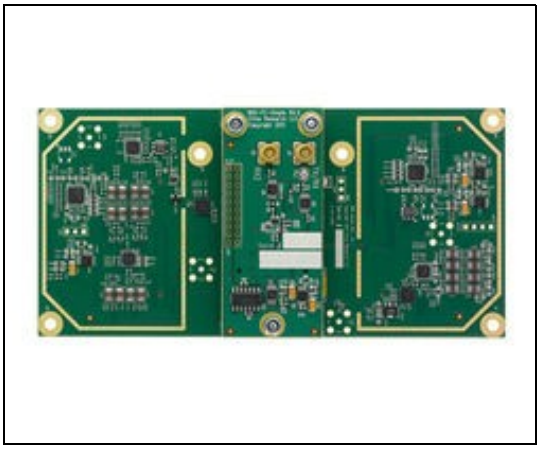
46.1

The WBX is a wide bandwidth transceiver that provides up to 100 mW of output power and a noise figure of 5 dB. The LO's for the receive and transmit chains operate independently. The WBX daughterboard is supported by the USRP Hardware Driver? (UHD) software API for seamless integration into existing applications.

The WBX provides phase coherent operation, although with a 180-degree ambiguity, which must be calibrated out in the application. For phase-coherent applications, Ettus Research recommends the SBX with the N200/N210 or the SBX or UBX with the X300/X310.

46.2

- Frequency Range: 50MHz - 2.2GHz
- Versions: 40MHz / 120MHz
- Power Output: 100mW
- Noise Figure: 5dB



46.3

46.3.1

- 2 quadrature frontends (1 transmit, 1 receive)
  - Defaults to direct conversion
  - Can be used in low IF mode through lo_offset with uhd::tune_request_t
- Independent receive and transmit LO's and synthesizers
  - Allows for full-duplex operation on different transmit and receive frequencies
  - Can be set to use Integer-N tuning for better spur performance with uhd::tune_request_t

46.3.2

Transmit: **TX**/**RX**

Receive: **TX**/**RX** or **RX2**

- **Frontend 0:** Complex baseband signal for selected antenna
- **Note:** The user may set the receive antenna to be TX/RX or RX2. However, when using a WBX board in full-duplex mode, the receive antenna will always be set to RX2, regardless of the settings.

46.3.3

- Transmit Gains: **PGA0**, Range: 0-31.5dB
- Receive Gains: **PGA0**, Range: 0-31.5dB

46.3.4

- WBX: 40 MHz, RX & TX
- WBX-120: 120 MHz, RX & TX

46.3.5

- **lo_locked:** boolean for LO lock state

46.4

46.4.1

- 50MHz - 2.2GHz

46.4.2

- 2 - 4 dB @ (50MHz ~ 1.2GHz)
- 4 - 8 dB @ (1.2GHz ~ 2.2GHz)

46.4.3

- 10 - 18 dBm

46.4.4

- -30 dBc

46.4.5

- 18 - 20 dBm @ (50MHz ~ 1.4 GHz)
- 12 - 18dBm @ (1.4GHz ~ 2.2 GHz)

46.4.6

- 30 - 32 dBm @ (50MHz ~ 800MHz)
- 25 - 30 dBm @ (800MHz ~ 2.2GHz)

46.4.7

- -30 dBc @ (50MHz ~ 1.9GHz)
- - 24 dBc @ (1.9GHz ~ 2.2GHz)

46.4.8

- All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

46.4.9

- The maximum input power for the WBX is -15 dBm.

46.5

- WBX without UHD Corrections

46.6

- Ettus Research recommends to always use the latest stable version of UHD

46.6.1

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

46.6.2

- Current Hardware Revision: 1
- Minimum version of UHD required: 3.8.0

46.7

46.7.1

- 0-40 °C

46.7.2

- 10% to 90% non-condensing

46.8

46.8.1

- N or X Series

46.8.2

- X Series only

46.9

The WBX daughterboard is capable of phase-synchronous operation, but with a with a 180-degree ambiguity, which must be calibrated out in the user application. The SBX, UBX, TwinRX daughterboards are recommended for phase-coherent applications.

46.10

46.10.1

WBX Schematics

WBX FE Schematics

46.11

Part Number	Description	Schematic ID (Page)
ADA4937	Ultralow Distortion Differential ADC Driver	U304 (2)
ADP3336	High Accuracy Ultralow IQ, 500 mA anyCAP® Adjustable Low Dropout Regulator	U306, U308 (2); U503, U505 (4)
ADL5387	50 MHz to 2 GHz Quadrature Demodulator	U307 (2)
ADL5385	50 MHz to 2200 MHz Quadrature Modulator	U501 (4,5)
ADF4350 / ADF4351	Wideband Synthesizer with Integrated VCO	U201 (3); U401 (5)
HMC472LP4	Attenuator	U302 (2); U504 (4,5)
MGA82563	Amplifier	U313 (2)
24LC024	EEPROM	U202 (3); U403 (5)
GVA?84+	Amplifier	U502 (4)

46.12

46.12.1

- Media:cu ettus-wbx-cca.pdf

46.13

- The WBX daughterboard features female MCX connectors for both the TX/RX and RX2 connectors.

46.14

46.14.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

46.14.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

46.15

46.15.1

- Media:volatility UBX CBX WBX SBX r1 1.pdf

46.16

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

47 OctoClock CDA-2990

47.1

The OctoClock CDA-2990 is an affordable solution for high-accuracy time and frequency reference distribution. The OctoClock accepts 10 MHz and PPS signals from an external source, and distributed each signal 8 ways. The OctoClock is a useful accessory for users that would like to build multi-channel systems that are synchronized to a common timing source.

The OctoClock-G CDA-2990 provides the same functionality as the OctoClock CDA-2990, but includes a GPSDO that can be used to produce internal 10 MHz and PPS signals, as well as a front-panel switch to select between internal and external sources for the signals. Note that *both* signals must come from the same chosen source: either *both* are generated internally from the GPSDO, or *both* are provided from the external inputs.

Note: Both of the CDA-2990 devices are functionally identical to the previous generation OctoClock, which contained an Ettus Research logo.

47.2

- 8-Way Time and Frequency Distribution (1 PPS and 10 MHz)
- Convenient Solution for Multi-Channel Synchronization
- Use with MIMO Capable N-Series Devices for Coherent System
- External 10 MHz/1 PPS Source Required
- 19" Rackmount ? 1U

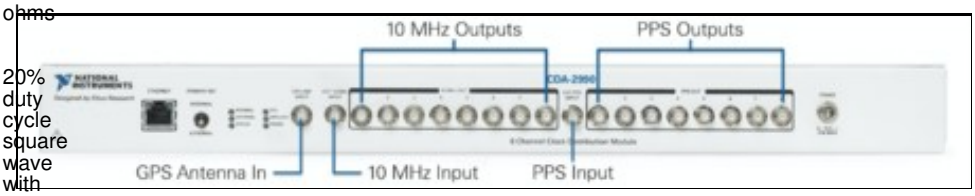
47.3

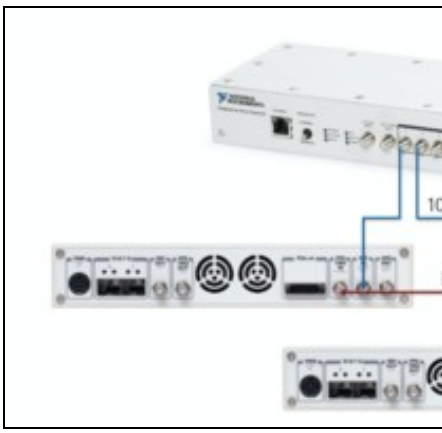
10 MHz output

1.25 Vpp at 50 ohms, 3.3Vpp at 1M ohms

1 PPS output

20% duty cycle square wave with amplitude 5 V





10 MHz input

0-20 dBm

1 PPS input

2.5-5 V

47.4

47.4.1

4 x 17.187x 1.75 inches

47.4.2

2.6 lbs

47.4.3

- File:cu ettus octoclock cca.pdf
- File:cu ettus-octoclock.pdf

47.4.4

47.4.4.1

- Motherboard

47.4.4.2

- Enclosure

47.5

47.5.1

- 0-40 °C

47.5.2

- 10% to 90% non-condensing

47.6

All RF Ports are matched to 50 Ohm with -10dB or better return loss generally. Detailed test is pending.

47.7

The Ethernet port on the OctoClock and OctoClock-G allows the device to be connected to the network. When connected, the `uhd_find_devices` utility can find any OctoClock devices on the network, and the device's firmware may be updated over Ethernet using the `octoclock_burn_eeprom` utility. In addition, with the OctoClock-G, NMEA strings may be obtained from the internal GPSDO via the UHD API. Note that there is no host CPU in the OctoClock, so it is not possible to SSH into the device.

47.8

47.8.1

OctoClock Schematics

47.9

Part Number	Description	Schematic ID (Page)
ENC28J60?DIG	Ethernet Controller	U103 (1)
ATmega128	Microcontroller	U102 (1)
LC_XO Spec Sheet Manual	Jackson Labs LC_XO (OCXO)	U206 (2)
SN74AUP1T57	VOLTAGE-LEVEL TRANSLATOR	U204, U203 (2)
CDCE18005?PWR	Output Clock Programmable Buffer	U205 (2)
74HC4020	Binary Ripple Counter	U207 (2)
LMZ12001	Power Module	U101 (1)

47.10

47.10.1

As of December 1st, 2010 all Ettus Research products are RoHS compliant unless otherwise noted. More information can be found at http://ettus.com/legal/rohs-information

47.10.2

Management Methods for Controlling Pollution Caused by Electronic Information Products Regulation

Chinese Customers

National Instruments is in compliance with the Chinese policy on the Restriction of Hazardous Substances (RoHS) used in Electronic Information Products. For more information about the National Instruments China RoHS compliance, visit ni.com/environment/rohs_china.

47.11

47.11.1

- Media:OctoClock CoV.pdf

47.12

The OctoClock's firmware is divided into two image files: octoclock_bootloader.hex and octoclock_r4_fw.hex. All pre-built image files can be found here, in version-specific ZIP files. Download the version corresponding to the version of UHD that you're running. You must use at least version 3.9.2.

- Full instructions on updating the OctoClock's firmware is located here
- Source of the firmware for the OctoClock is located here

47.13

OctoClock Spec Sheet

FPGA Resources

UHD Stable Binaries

UHD Source Code on Github

47.14

- **What are the OctoClock and OctoClock-G?**

The OctoClock is a USRP-compatible accessory that allows you to easily synchronize up to 8 USRP radios. Multiple OctoClock devices can be combined for synchronization of larger numbers of USRP radios.

The OctoClock-G is an OctoClock with added GPSDO module.

In this FAQ we will use "OctoClock" to refer to either unless there is a specific need to use either name.

- **When would I used the OctoClock?**

The OctoClock is useful for synchronizing multiple USRP devices for high channel count systems.

The following applications can benefit from OctoClock clock distribution:

- Direction Finding
- Beamforming
- Adaptive Beamforming
- Multiple-In-Multiple-Out (MIMO) Prototyping
- Geolocation Systems that Use Time-Difference-of-Arrival (TDOA)
- Multi-Channel, Multi-Static, and Passive RADAR
- Multi-Band GPS Record and Playback
- Multi-Band Cellular Monitoring

Essentially, anything that requires synchronization or the distribution of timing information would benefit from the use of the OctoClock.

- **How does the OctoClock work?**

The OctoClock accepts 10 MHz and PPS signals from an external source. Active circuits are used to amplify and split the signals 8-ways. Matched-length traces minimize phase differences between all 10 MHz and 1 PPS outputs

The OctoClock-G includes an internal GPSDO (GPS Disciplined Oscillator) which provides an internal source for 10 MHz and PPS from an OCXO high precision oscillator. Add a GPS antenna (optional) with a clear view of the sky for GPS Disciplining of the OCXO that futher enhances frequency accuracy of the OCXO and global time synchronization.

- **Does the OctoClock-G provide power for using an active GPS antenna?**

Yes. The OctoClock-G's GPSDO module is the Labs LC_XO, which can provide 5 V at up to 50 mA to the external active GPS antenna. The non-G OctoClock does not have this GPSDO module, and thus cannot use a GPS antenna if attached, nor provide power to one if attached.

- **Where can I find user manuals for the OctoClock and USRP?**

Synchronization and MIMO Capability with USRP Devices

UHD Manual: Multiple USRP configurations

- **What USRP model is recommended for MIMO systems?**

The USRP N200 + N210, N310, N320 + N321, X300 / X310 are recommended for building high channel count and MIMO systems. These models provide external PPS and 10 MHz reference inputs. The USRP N200 and N210 support the USRP MIMO cable.

The USRP B100, B200, B210, E100, E110, and E310 can be synchronized with 10 MHz/PPS but are not phase coherent MIMO capable devices. The USRP1 cannot be synchronized with 10 MHz/PPS.

- **How does the automatic switchover functionality work?**

When using the OctoClock-G, the Internal/External switch on the front panel allows the user to choose between the internal GPSDO and external source 10 MHz/PPS source. If the selected source is unavailable, the device will automatically switch over to the backup frequency source. When switchover is active the corresponding LED indicator will illuminate.

If neither source is present, the internal, external and status LEDs will not be illuminated and the user will not received valid 10 MHz/PPS outputs.

- **What do the LEDs status indicators mean?**

The following list describes the behavior when each of the 6 LED status indicators is illuminated:

- Internal - internal GPSDO is selected and present.
- External - external source is selected and present
- Status - Either the internal GPSDO or external source is selected. If neither source is present this LED will turn off (no signals are output).
- PPS - selected PPS pulse high.
- GPS Locked - GPS is receiving signals and has valid time/position lock.
- Power - Power is applied.

- **What are the input and output specifications?**

- 10 MHz Input ? 0-10 dBm
- 10 MHz Outputs - ~1.4 Vpp Square Wave, Impedance 50 ohm nominal
- 1 PPS Input - Logic-level pulse, 2.5 V - 5 V
- 1 PPS Outputs - Logic-level pulse, 2.5 V - 5 V

- **What is the input voltage rating?**

The OctoClock can be powered at any voltage between 6 and 15 Vdc.

- **Are the design files open source?**

As with all of our products, the driver code is free & open source, and can be found in our UHD repository. The schematics are also available.

48 GPSDO

48.1

48.1.1

- N200/N210
- E100/E110

48.1.2

Module Specifications	
1 PPS Accuracy	±50ns to UTC RMS (1-Sigma) GPS Locked
Holdover Stability	<±11?s over 3 hour period at +25C
1 PPS Output (OCXO Flywheel Generated)	3.3VDC CMOS
RS-232 Control	NMEA & SCPI-99 Control Commands, Integrated into UHD
GPS Frequency	L1, C/A 1574MHz
GPS Antenna	Active (3V compatible) or Passive
GPS Receiver	50 Channels, Mobile, WAAS, EGNOS, MSAS capable
Sensitivity	Acquisition -144dBm, Tracking -160dBm
TTFF	Cold Start: <45 sec, Warm Start: 1 sec, Hot Start: 1 sec
ADEV	1E-11 at 1s
Warm Up Time / Stabilization Time	<5 min at +25C to 1E-08 Accuracy
Supply Voltage (Vdd)	6VDC
Power Consumption	<1.8W Max, 1.35W Typical
Operating Temperature	0C to +60C
Storage Temperature	-45C to 85C

Oscillator Specifications		
Frequency Output	10MHz	
10MHz Reference	±1E-08 after 1 hour at 25C	
Frequency Stability Over Temperature (Unlock Condition)	±7.5E-08	
Warm Up Time	< 1 min at +25C	
Phase Noise at 10MHz	1Hz	-80dBc/Hz
	10Hz	-110dBc/Hz
	100Hz	-135dBc/Hz
	1kHz	-145dBc/Hz
	10kHz	<-145dBc/Hz

48.1.3

Media:gpsdo-kit 4.pdf

48.2

48.2.1

- X300/X310

48.2.2

Module Specifications	
1 PPS Accuracy	±50ns to UTC RMS (1-Sigma) GPS Locked
Holdover Stability	<±20?s over 3 hour period at +25C
1 PPS Output (OCXO Flywheel Generated)	3.3VDC CMOS
RS-232 Control	NMEA & SCPI-99 Control Commands, Integrated into UHD
GPS Frequency	L1, C/A 1574MHz
GPS Antenna	Active (5V compatible) or Passive
GPS Receiver	50 Channels, Mobile, WAAS, EGNOS, MSAS capable
Sensitivity	Acquisition -142dBm, Tracking -168dBm
TTFF	Cold Start: <45 sec, Warm Start: 1 sec, Hot Start: 1 sec
ADEV	1E-11 at 1s
Warm Up Time / Stabilization Time	<5 min at +25C to 1E-08 Accuracy

Supply Voltage (Vdd)	3.3/5VDC
Power Consumption	0.7W Max
Operating Temperature	0C to +60C
Storage Temperature	-45C to 85C

Oscillator Specifications	
Frequency Output	10MHz
10MHz Reference	±2E-08 after 1 hour at 25C
Frequency Stability Over Temperature (Unlock Condition)	±2.5E-08
Warm Up Time	≤ 1 min at +25C
Phase Noise at 10MHz	1Hz -75dBc/Hz
	10Hz -110dBc/Hz
	100Hz -132dBc/Hz
	1kHz -142dBc/Hz
	10kHz <-145dBc/Hz

48.2.3

- Media:GPSDO Jackson Labs LCXO and FireFly1A CoV.pdf

48.3

48.3.1

- B200/B210

48.3.2

Module Specifications	
1 PPS Accuracy	±50ns to UTC RMS (1-Sigma) GPS Locked
1 PPS Output (OCXO Flywheel Generated)	3.3VDC CMOS
RS-232 Control	NMEA & SCPI-99 Control Commands, Integrated into UHD
GPS Frequency	L1, C/A 1574MHz
GPS Antenna	Active (5V compatible) or Passive
GPS Receiver	50 Channels, Mobile, WAAS, EGNOS, MSAS capable
Sensitivity	Acquisition -142dBm, Tracking -168dBm
TTFF	Cold Start: <45 sec, Warm Start: 1 sec, Hot Start: 1 sec
ADEV	1E-11 at 1s
Warm Up Time / Stabilization Time	<5 min at +25C to 1E-08 Accuracy
Supply Voltage (Vdd)	3.3V
Power Consumption	0.5W Max
Operating Temperature	0C to +60C
Storage Temperature	-45C to 85C

Oscillator Specifications	
Frequency Output	10MHz
10MHz Reference	±2E-08 after 1 hour at 25C
Frequency Stability Over Temperature (Unlock Condition)	±7.5E-08
Warm Up Time	≤ 1 min at +25C
Phase Noise at 10MHz	1Hz -65dBc/Hz
	10Hz -102dBc/Hz
	100Hz -132dBc/Hz
	10kHz <-152dBc/Hz

48.3.3

- Media:GPSDO Jackson Labs LCXO and FireFly1A CoV.pdf

48.4

Manufacturer: Samtec

PN: BBL-115-G-E

- https://www.samtec.com/products/bbl-115-g-e
- Datasheet: File:JP6 JP7 pins.pdf

49 Antennas

49.1

Frequency: 824 to 960 MHz, 1710 to 1990 MHz

Gain: 3dBi

Pattern: Omni

Datasheet: Media:ettus research vert900 datasheet.pdf

49.2

Frequency: 2.4 to 2.48 GHz, 4.9 to 5.9 GHz

Gain: 3dBi

Pattern: Omni

Datasheet: Media:ettus research vert2450 datasheet.pdf

49.3

Frequency: 2.4 to 2.48 GHz, 4.9 to 5.9 GHz

Gain: 5-6dBi

Pattern: Directional

Datasheet: Media:ettus research lp0410 datasheet.pdf

49.4

Frequency: 850 MHz to 6.5 GHz

Gain: 5-6dBi

Pattern: Directional

Datasheet: Media:ettus research lp0965 datasheet.pdf

49.5

Datasheet:Media:ettus research gps antenna datasheet.pdf

50 UHD

50.1

The Ettus Research USRP? family of Software Defined Radios (SDRs) are versatile devices that allow users to transmit and receive many different and custom waveforms at various frequencies and settings on a common hardware platform. Commercial, academic, and military customers employ the flexible and reusable USRP hardware for research, wireless exploration, and field deployments.

The USRP comes in many form factors and configurations. The main categories of USRPs are:

- Bus (B) Series ? Connected to a host computer via a USB connection
- Network(N) Series ? Connected to a host computer via an Ethernet connection
- High Performance (X) Series ? Connection can be Ethernet or a x4 PCI-Express connection
- Embedded (E) Series ? Meant to run stand-alone (without a host computer).

The USRP Hardware Driver (UHD) is a user-space library that runs on a general purpose processor (GPP) and communicates with and controls all of the USRP device family. The B, N and X series USRPs send and receive samples from a host computer as illustrated in figure 1. And since our embedded series USRPs house an internal GPP you can run these radios without the need of a host computer (stand-alone mode).

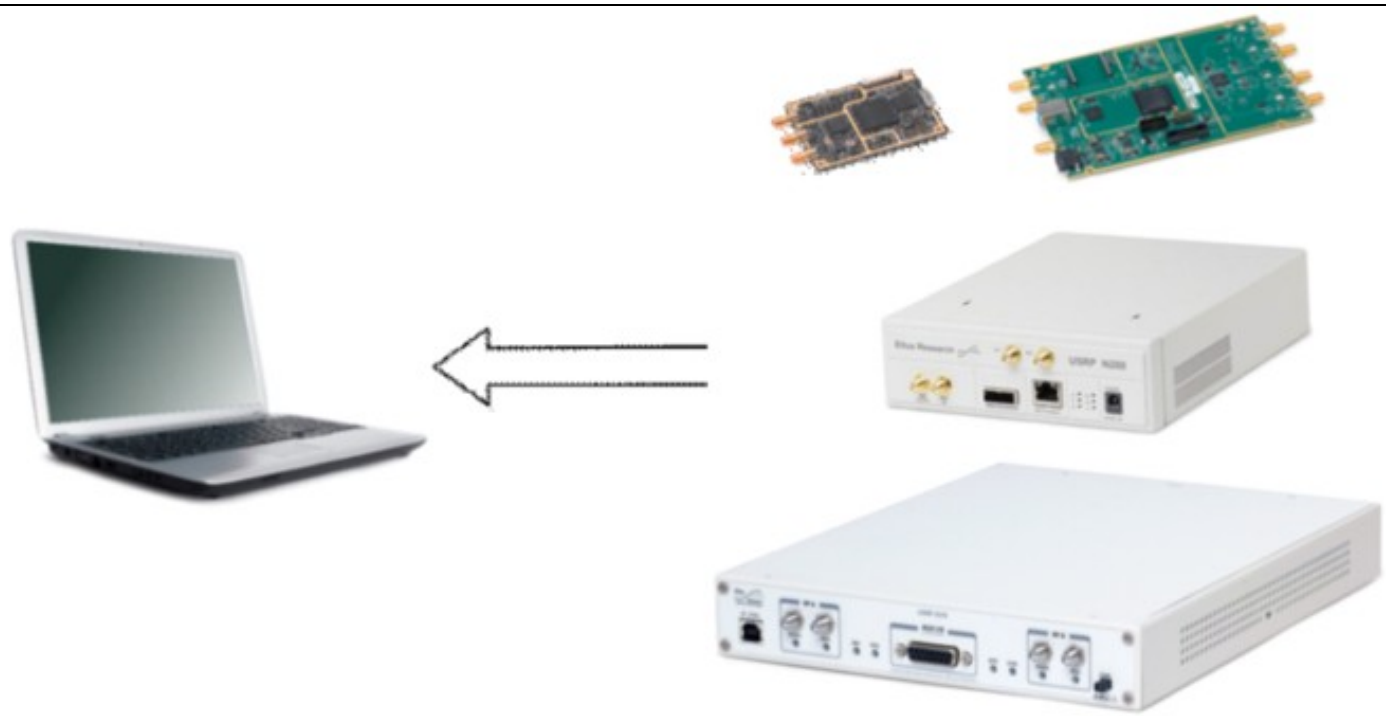


Figure 1. USRP Host Based System

USRPs are transceivers, meaning that they can both transmit and receive RF signals. UHD provides the necessary control used to transport user waveform samples to and from USRP hardware as well as control various parameters (e.g. sampling rate, center frequency, gains, etc) of the radio.

UHD GPP driver and firmware code is written in C/C++ while the code developed for the FPGA (Field Programmable Gate Array) is written in Verilog. There is a C/C++ API that can interface to other software frameworks, as in the case of GNU Radio, or a user can simply build custom signal processing applications directly on top of the UHD C/C++ API. Figure 2 illustrates this concept:

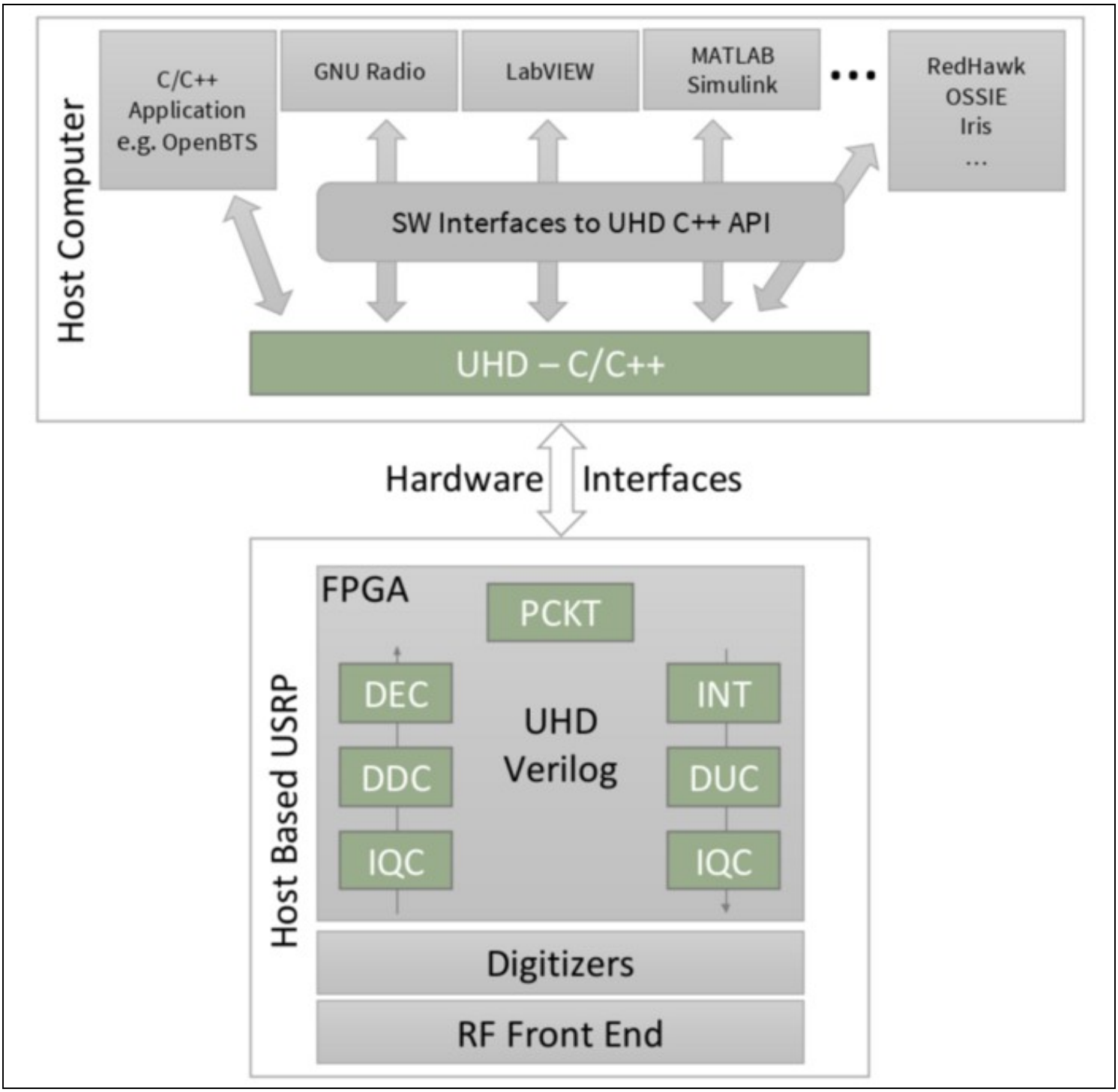


Figure 2. UHD components

50.2

The majority of the UHD code base is open source, including code that executes on the host, as well as code targeted to the USRP hardware (FPGA and microcontroller firmware). As dual-licensed software, UHD is available under the open-source GNU Public License version 3 (GPLv3), as well as an alternate, less-restrictive license offered only by Ettus Research. For more information on our licensing policy, please contact info@ettus.com.

50.3

Due to the open source nature of UHD, the entire development process is also open, and it is possible to track UHD?s development through our Git version control system. Users can make a choice if they prefer the latest development code, which is most feature-rich but can be unstable at times, or the more thoroughly tested code that does not include the latest development. Versioned releases (e.g., UHD version 3.9.2) happen approximately every two months, and usually only include bugfixes compared to the previous version (e.g., UHD 3.9.2 has the same feature set as 3.9.1, but is more stable). These bugfix releases are done off of the maint branch, which is where all the bugfix development happens.

New feature development is done on the master branch. This branch should not be considered stable, even though we use continuous integration systems to monitor its state. However, it is possible that APIs or dependency requirements change on the master branch.

At the end of a feature development cycle, the master branch is frozen, and only bugfixes are accepted into the master branch. Once the master branch is considered stable, the maint branch is reset to master, and a new versioned release is produced from the previous master branch. This is indicated by

a major version number jump (e.g., when going from 3.8.3 to 3.9.0). Major version releases occur 1-2 times per year, and usually accompany new product releases.

Aside from the maint and master branches, new feature branches are sometimes published (temporarily) to give insight into upcoming features, and earlier access to new development.

50.4

The UHD software API supports application development on all USRP SDR products. Using a common software interface is critical, enabling code portability and, allowing applications to transition seamlessly to other USRP SDR systems when development requirements expand or new systems become available. Hence, it enables a significant reduction in development effort by allowing you to preserve and reuse your legacy code. UHD can be installed on Linux, Windows, or a Mac. Installation packages for these platforms as well as instructions for building from source can be found here.

Figure 3. One driver for all USRPs and USRP Configurations

50.5

50.5.1

Compatible with Python 2.7 and above (Python 3 is supported). Note Python is required for building, and some utilities that ship with UHD are written in Python. It is not, in general, required for running UHD applications.

50.5.2

This is dependent on the distribution you are using, but most, if not all, of the dependencies should be available in the package repositories for your package manager. See the Linux build instructions for more information.

50.5.3

Install the Xcode app to get the build tools (GCC and Make). Use MacPorts to get the Boost and Mako dependencies. Other dependencies can be downloaded as DMG installers from the web or installed via MacPorts. See the UHD OS X build instructions for more information.

50.5.4

The dependencies can be acquired through installable EXE files. Usually, the Windows installer can be found on the project's website. Some projects do not host Windows installers, and if this is the case, follow the auxiliary download URL for the Windows installer (below). See the Windws build instructions for more information.

50.5.5

- Required to check out the repository (not necessary if building from tarballs).
- On Windows, install Cygwin with Git support to checkout the repository or install msysGit from http://code.google.com/p/msysgit/downloads/list.

50.5.6

UHD Version	GCC	Clang	MS Visual C++	CMake	Boost	LibUSB	Mako	Doxygen	Python	Xilinx Vivado
3.9.X	>= 4.4	>= 3.3	>= 2012 (11.0)	>= 2.8	>= 1.46	>= 1.0	>= 0.5.0	>= 1.8 (recommended)	>= 2.7	== 2014.4
3.10.X	>= 4.8	>= 3.3	>= 2012 (11.0)	>= 2.8	>= 1.53	>= 1.0	>= 0.5.0	>= 1.8 (recommended)	>= 2.7	== 2015.4

Other compilers (or lower versions) may work, but are unsupported.

Note on MSVC: The free Visual Studio Express Edition for Desktop works.

Note on Boost: Older versions of UHD may not work with newer versions of Boost, as the Boost API sometimes changes, and we can't retroactively fix older UHD versions. For example, UHD 3.8.X (and older) won't work with Boost 1.60.

50.6

UHD handles the control for transporting I and Q samples (see here for information on I and Q samples) by using standard interface methods such as Ethernet, USB and PCI-Express. Samples can be sent in a continuous stream as in figure 4a or in bursts, figure 4b. In addition the user has the ability to specify when samples are received or transmitted by using built in burst and timed commands.

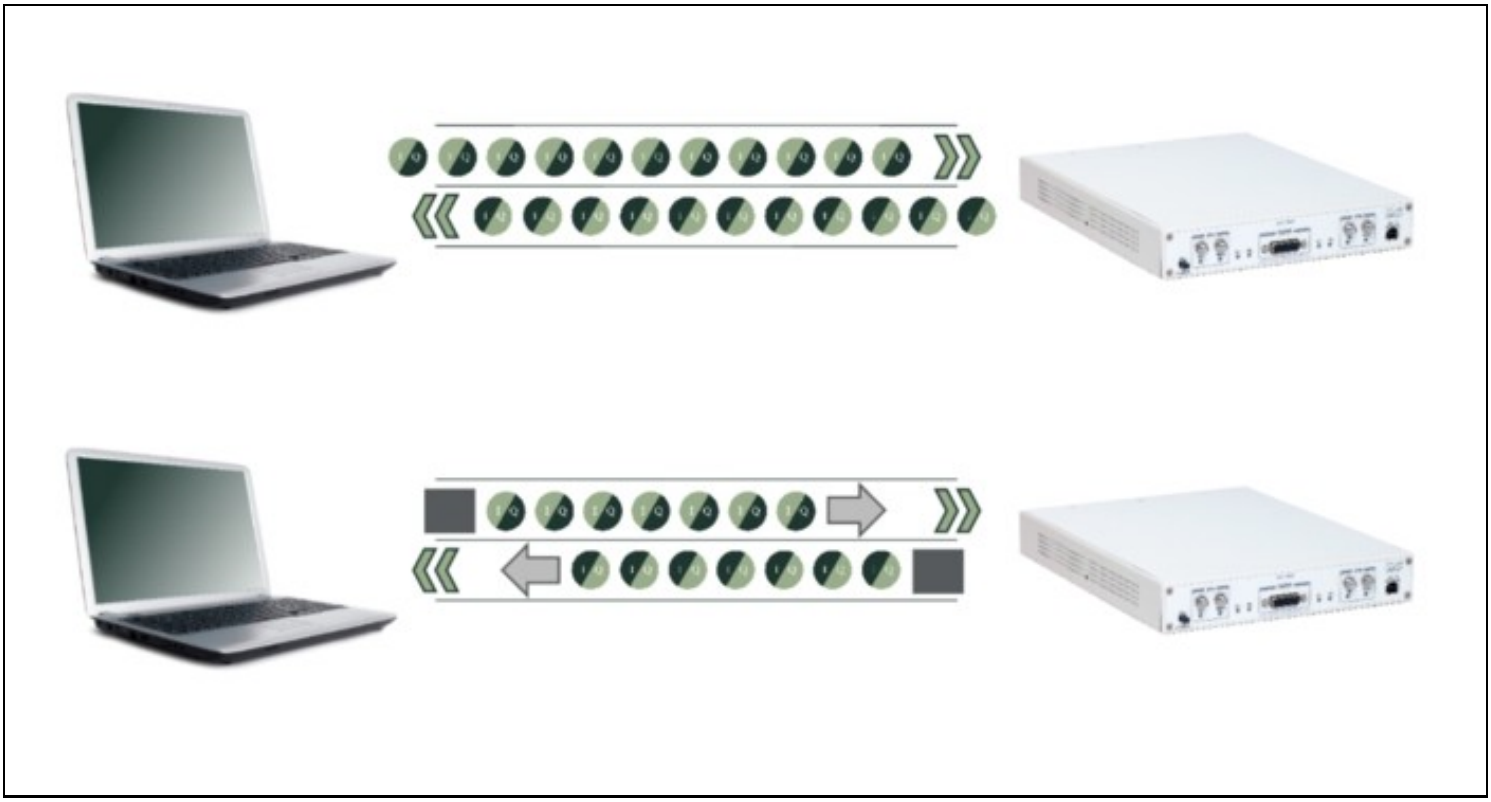


Figure 4. a) Continuous Sample Streaming. b) Burst Sample Mode (SOB = Start of Burst while EOB = End of Burst).

50.7

USRP RF front ends may support a certain frequency step size that does not meet all or many of a user?s requirements. For this reason, UHD includes Digital Up-Conversion (DUC) and Digital Down-Conversion (DDC) DSP blocks in the FPGA for fine tuning the RF frequency (see here). This allows users to:

- Have a sub-Hz RF frequency step size
- Mitigate the DC problem that exists on Direct Conversion (Zero IF) hardware (see here).
- Fast tune inside the available bandwidth

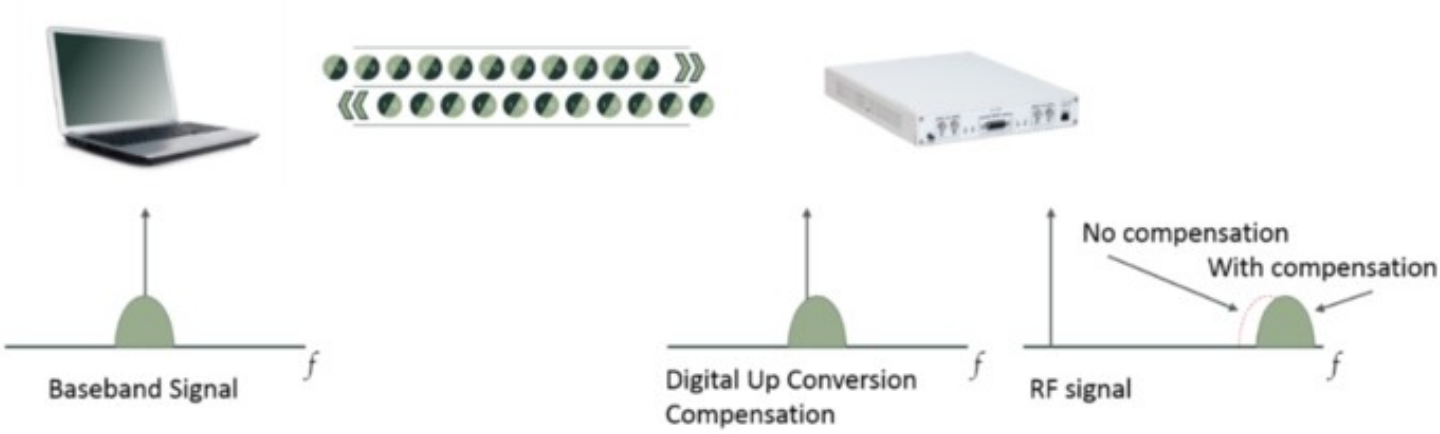


Figure 5. Center Frequency Fine Tuning

50.8

Different SDR configurations, waveforms, and applications require different sample rates. For example, a user may wish to monitor 100 MHz of instantaneous RF bandwidth, but their host PC may only be capable of analyzing 20 MHz of real-time bandwidth. For this and other cases, UHD allows users to set various sample rates to meet their custom application. Within the FPGA, UHD includes decimation and interpolation blocks in order to perform these sample rate translations.

50.9

Inherent to all direct conversion (zero IF) RF architectures is the effect of IQ imbalance. A video entitled ?Effects of Quadrature Impairments on 802.11ac EVM?, found here, demonstrates this attribute. Find additional information here. In summary, any given signal on a device using a direct conversion architecture has two paths for TX and two paths for RX. One path is the In-phase or ?I? and the other is Quadrature ?Q?, also referred to as Real and Imaginary.

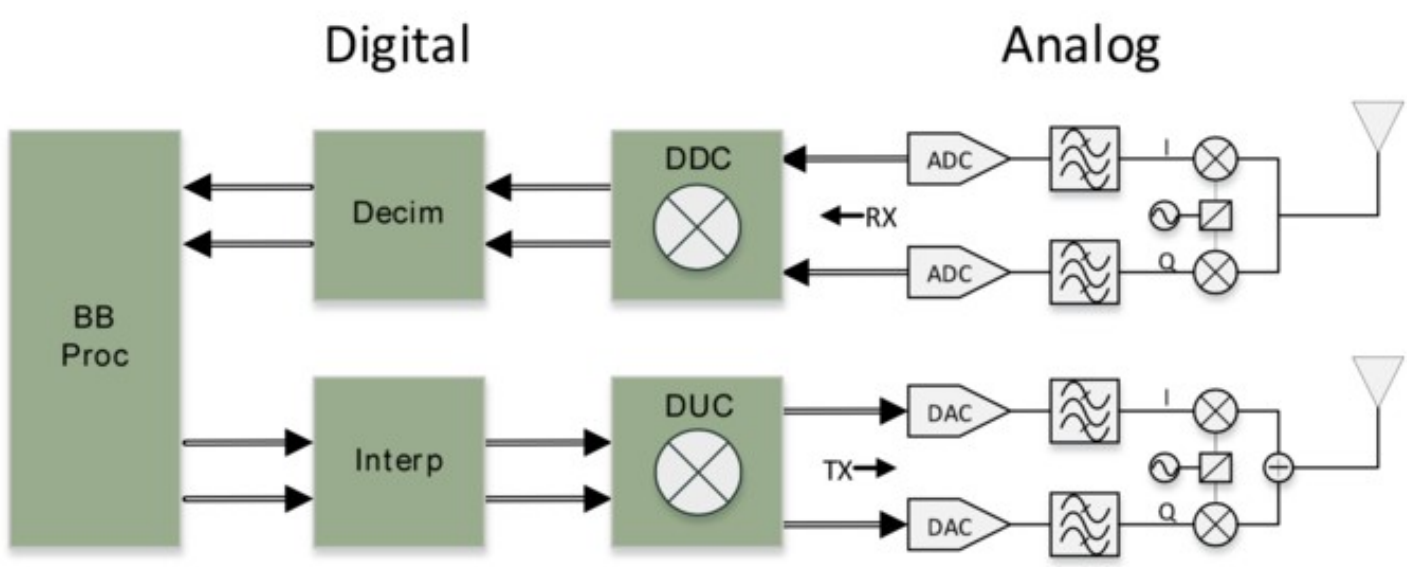


Figure 6. In-phase and Quadrature data paths

Because of slight differences due to variances in components, temperature, and other factors, the I and Q signal paths are subject to different conditions, thus altering the original signal that existed at initial capture. When the phase or amplitude of either the I or Q get altered the results show up as signals not actually present in the original signal. For example, note the following IQ impairment simulation in this GNU Radio Flowgraph created in GNU Radio Companion.

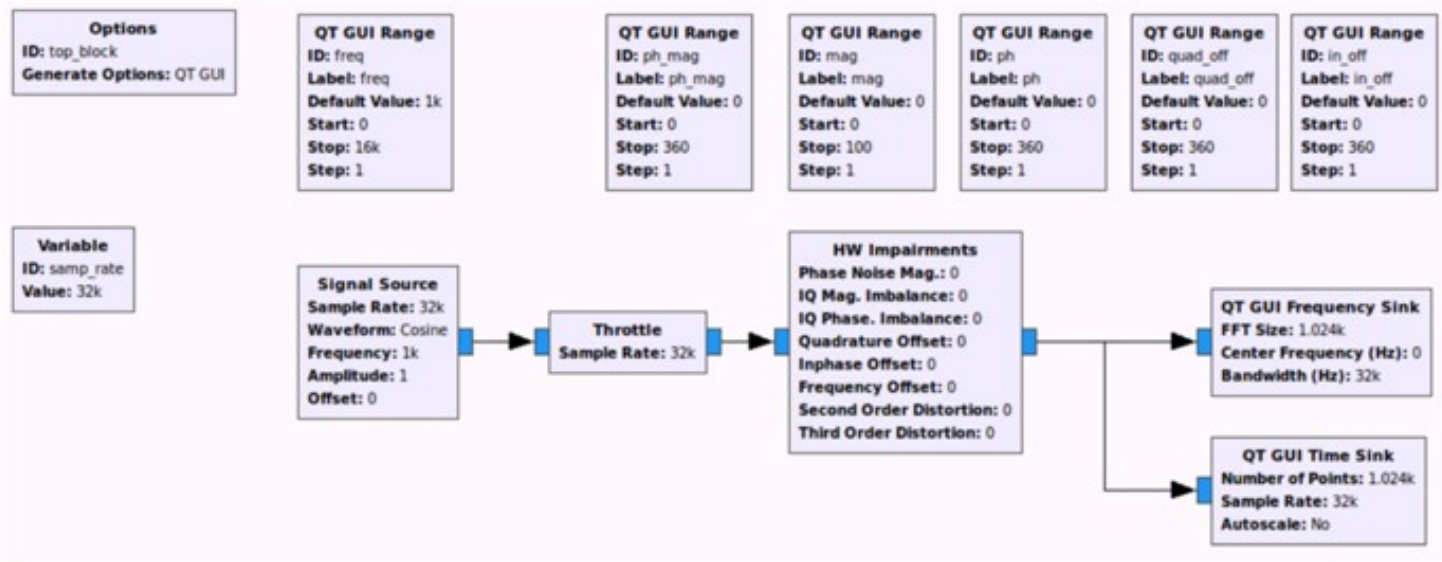


Figure 7. GNU Radio Flowgraph, IQ imbalance simulation

By using the ?HW Impairments? block, you can simulate what happens when the I and Q signal paths are disrupted. In figure 8a there are no impairments added. However, when the amplitude of the I path is increased slightly, you get the artifact as show in figure 8b

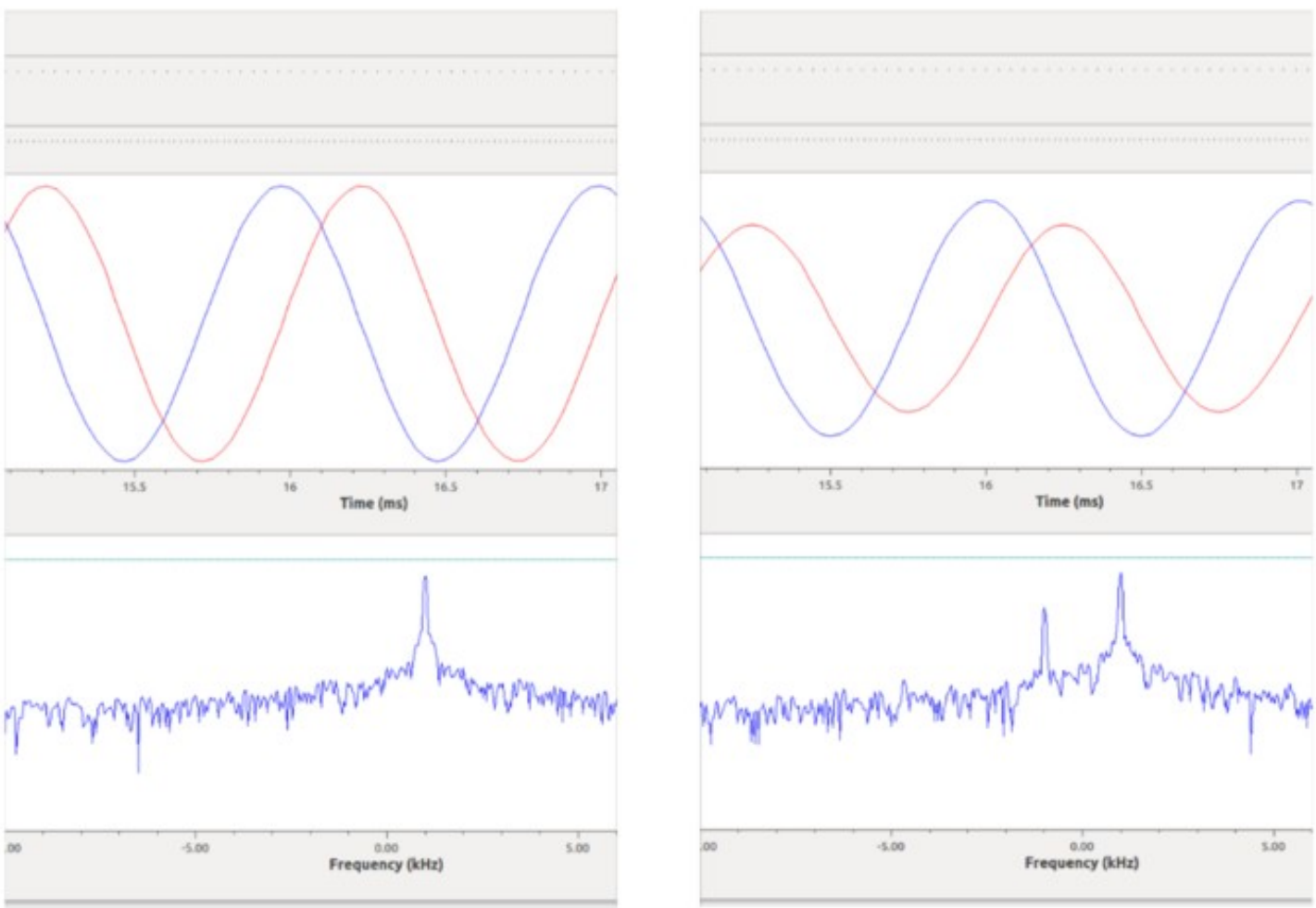


Figure 8. a) No IQ imbalance b) IQ imbalance

It is possible to mitigate some of these unwanted effects in the digital domain. UHD contains function blocks within the FPGA to compensate for IQ impairments; these blocks appeared in figure 1 as IQC blocks. A user can customize the parameters of these blocks based on empirical measurement or allow UHD to perform an automatic analysis and provide parameters based on a built-in IQ correction algorithm. See the following functions in the UHD manual for more information:

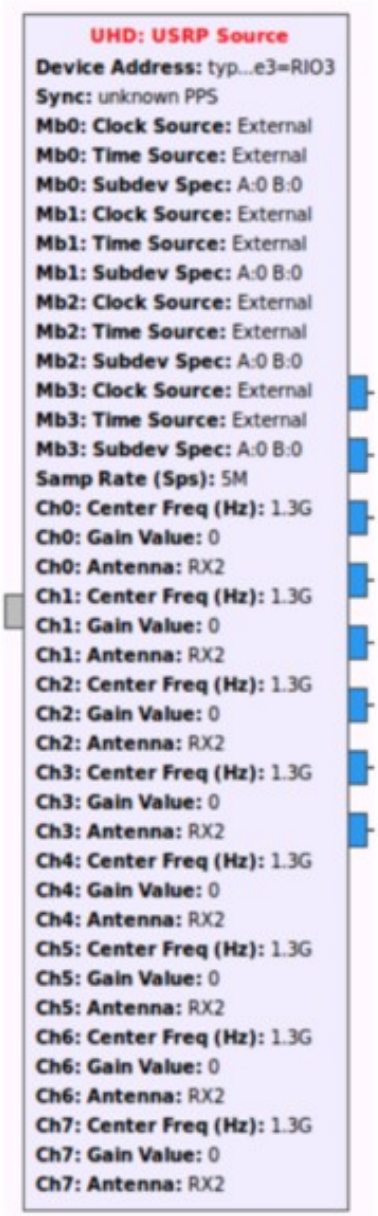
- uhd_cal_rx_iq_balance: - mimimizes RX IQ imbalance vs. LO frequency
- uhd_cal_tx_dc_offset: - mimimizes TX DC offset vs. LO frequency
- uhd_cal_tx_iq_balance: - mimimizes TX IQ imbalance vs. LO frequency

50.10

UHD makes scaling the number of channels on a USRP system simple by treating them all as one composite device, see figure 9 below. In the case of the X300/X310, when USRPs are used in this multi-USRP configuration users can use an external clocking source (having both a 10 MHz clock reference and a PPS signal) to synchronize all the devices together.

Figure 9. USRP: scaling number of channels

As an example, in an 8 channel receive system the ?USRP Source? block in GNU Radio Companion would look like figure 10 below



1 Channel

8 Channel

Figure 10. GNU Radio Flowgraph USRP source block, 1 channel vs 8 RX channels

50.11

General Purpose Input Output (GPIO) pins can be controlled manually through UHD or can be set from UHD to be automatically triggered when events such as TX or RX occur. An example of where or how to use this automatic triggering: when a user connects an RF amplifier to the TX or RX port of the USRP, the amplifier can be powered on only when the USRP is transmitting. You can find out more about the aux GPIO on the E3x0/X3x0 here.

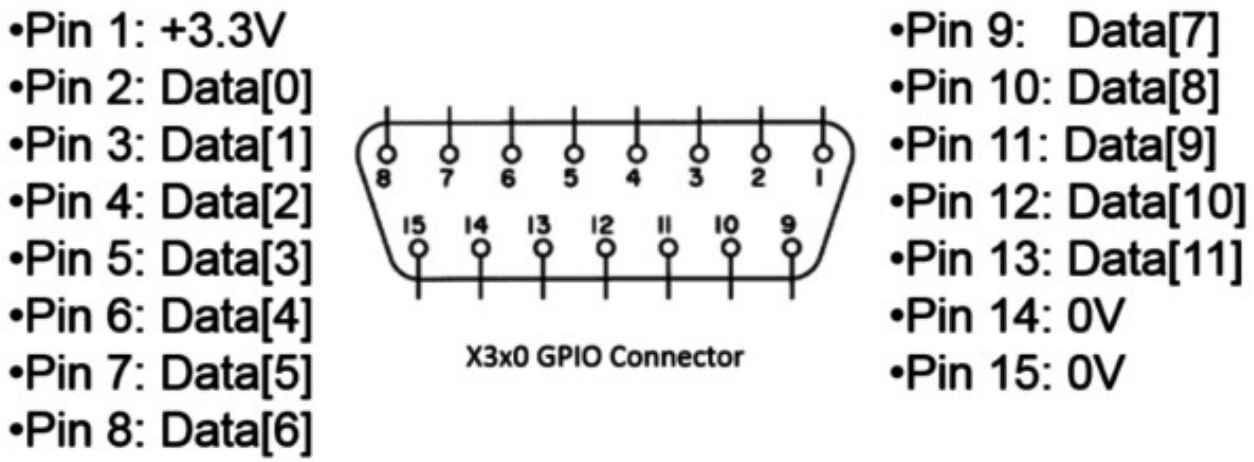


Figure 11. X3x0 GPIO Connector

50.12

While mostly a function of HW, the USRP can interface with different RF front ends from 3rd parties. By using the BasicRX and BasicTX daughterboards, a user can send or receive a Baseband or IF (Intermediate Frequency) analog signal to 3rd party front ends.

50.13

In addition to leading development of the gr-uhd module that integrates the USRP family (UHD) into GNU Radio, Ettus Research frequently contributes to the GNU Radio project. Although USRP radios are often associated with the GNU Radio software framework, many USRP applications (such as OpenBTS) run without GNU Radio. For more information on GNU Radio see here.

50.14

- Licensing FAQ

50.15

UHD aids commercial, academic, military, hobbyists and other SDR users with rapid prototyping or deployment of wireless protocols by providing a flexible feature rich SDR API.

50.16

Github: https://github.com/EttusResearch/uhd

Manual: https://files.ettus.com/manual/index.html

51 UHD Python API

51.1

As the name suggests, it exposes the UHD API into Python. We use `pybind11` to generate a Python module which exposes most of the C++ API, and some extra features. The Python API is part of stable releases.

The USRP Hardware Driver and USRP Manual covers most information about the UHD Python API and can be found here: https://files.ettus.com/manual/page_python.html

51.2

In order to test the Python API, check out the `master` branch and build it like always. When running CMake, make sure that the Python API was enabled (`-DENABLE_PYTHON_API=ON`).

The output from CMake should look something like this:

```
-- ####################################################
-- # UHD enabled components
-- ####################################################
--   * LibUHD
--   * LibUHD - C API
--   * LibUHD - Python API
--   * Examples
--   * Utils
--   * Tests
--   * USB
--   * B100
--   * B200
--   * USRP1
--   * USRP2
--   * X300
--   * N230
--   * OctoClock
--   * Manual
--   * API/Doxygen
--   * Man Pages
--
-- ####################################################
-- # UHD disabled components
-- ####################################################
--   * GPSD
--   * E100
--   * E300
```

Please refer to the USRP Manual for extended instructions especially when installing on Windows. Once it's built and installed, you'll be able to import the `uhd` Python module.

We have some examples in `host/examples/python`. The examples are very simple, but concise.

51.3

This Python example is based on the C++ example `uhd/host/examples/rx_samples_to_file.cpp`.

51.4

Documentation is currently pretty sparse. The best we can do right now is to ask users to infer the documentation from the C++ API. For example, the Python has an object called `MultiUSRP` which is an equivalent of the C++ `multi_usrp` API. The methods on both classes are the same, and take the same arguments.

51.5

Does it support Python 2 and 3?

Starting with UHD 4, Python 2 support has been removed.

Does it require GNU Radio?

No.

Does it use SWIG?

No, it uses `pybind11`. It also doesn't require the C API. Pybind11 is vendored with UHD so as to not require installing another dependency.

How does this relate to the Python API in gr-uhd?

It serves an entirely different purpose. This Python API is for people writing standalone applications for USRPs that *don't* use GNU Radio. `gr-uhd` is staying the way it is, and is going nowhere. If you're using GNU Radio, you probably don't care about this.

Are the UHD Python API and the gr-uhd Python API compatible?

Short answer: No. Long answer: There are very few cases where it makes sense to mix these APIs, so no. However, this means that a `TimeSpec` from the `Boost.Python` API is not convertible into a `time_spec_t` from the `gr-uhd` API.

Does it support RFNoC API?

For sure!

What's the streaming performance?

Worse than straight C++, but not a lot, thanks to NumPy. You can run `host/examples/benchmark_rate.py` if you want to see for yourself. Overall, `recv()` calls are pretty efficient if you've pre-allocated a NumPy array, because we can cast that to a straight pointer (and also skip any type checking!) and then it's not that different from a `recv()` call in a C++ app. However, consuming the data is limited by how fast you can handle that in Python.

52 Getting Started with RFNoC in UHD 4.0

52.1

AN-400 by Sugandha Gupta, Brent Stapleton, Wade Fife, and Michael Dickens

52.2

This guide describes how to get started with FPGA and Software development for RF Network-on-Chip (RFNoC?). It gives a brief introduction to RFNoC and explains the steps needed to generate, build, and use custom RFNoC images and introduces the process for creating and integrating new RFNoC IP blocks.

52.3

This guide is written for hardware and software engineers who want to use the RF Network-on-Chip (RFNoC?) architecture or want to develop intellectual property (IP) using the RFNoC architecture. For more details on the architecture please refer to the RFNoC Specification.

This guide assumes that you have some basic familiarity with USRPs, such as connecting them, configuring your network interfaces, etc., so that your USRP is ready for use. See the Getting Started Guide for your USRP if you are just getting started with USRPs.

52.4

The RFNoC code base is open source, including code that executes on the host, as well as code targeted to the USRP hardware (FPGA and microcontroller firmware). RFNoC is available under the open-source GNU Lesser General Public License (LGPL). For more information on our licensing policy, please contact info@ettus.com.

52.5

RFNoC is currently supported on the USRP X410 series, and all the Generation-3 USRPs in the X series (X3xx), E series (E3xx) and N series (N3xx). For details on the hardware, software, and process required to build custom USRP FPGA images that include RFNoC blocks, see the USRP Build Documentation in the UHD and USRP Manual.

It is recommended that you learn how to build an FPGA image for your USRP and download it to the device before starting this guide if you have never done so before. This will ensure you have all the necessary software installed.

52.6

52.6.1

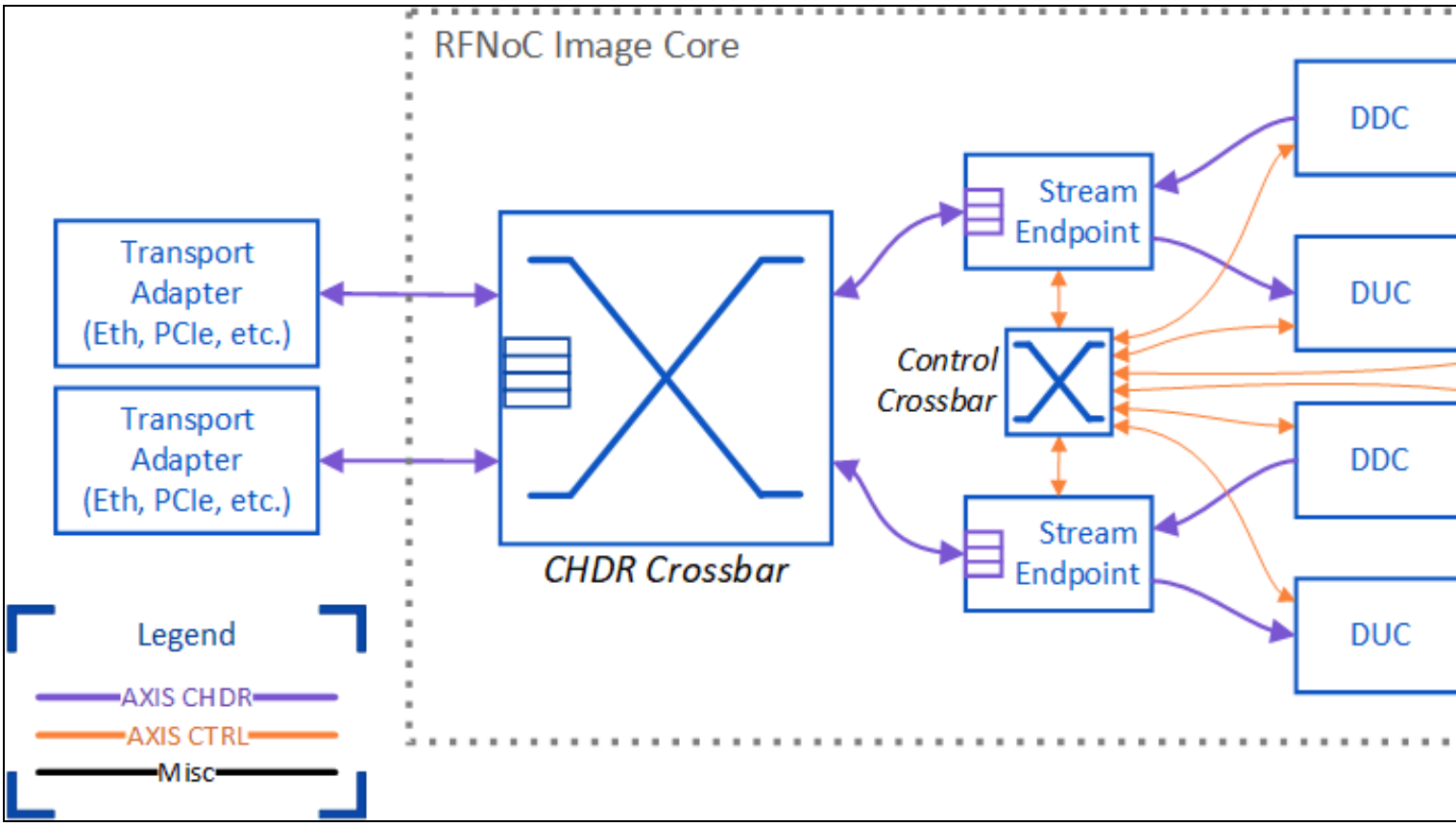
RFNoC? is a heterogeneous processing framework used to implement high-throughput DSP in the FPGA for Software Defined Radio (SDR) systems in an easy-to-use and flexible way. It provides all the infrastructure required to insert signal-processing IP into the FPGA logic and communicate with it through software. It also provides highly-optimized software and FPGA code to enable high-performance streaming to and from IP blocks on the USRP device.

The IP blocks in RFNoC are called *RFNoC blocks*. The RFNoC blocks wrap the IP and provide a custom interface to the RFNoC infrastructure through a tool-generated interface called the *NoC Shell*. Many standard blocks are included in UHD 4.0. These blocks enable typical operation of the USRP and allow RFNoC to connect to the different hardware components of the USRP. Several of the included blocks are described in the section Available RFNoC Blocks. You can mix the available blocks for your application, or develop your own custom RFNoC blocks with your own IP to open up new applications. The NoC Shell hides the complexity of RFNoC from your block, making it easy to plug your IP into the USRP.

RFNoC is used on all Generation-3 USRPs and is installed with UHD 4.0. USRPs ship with a default RFNoC image that can be used as is, or can be modified and/or customized to suite your application. So if you're using a modern USRP, it's running RFNoC under the hood even if you haven't customized anything.

52.6.2

The diagram below shows a simplified view of an RFNoC FPGA image that is similar to the default images available on USRPs.



Notice that an RFNoC FPGA is made up of several components. A description of each one is provided below.

- **RFNoC Image Core**
  This is the main block of the RFNoC framework and instantiates the components that make up RFNoC. The contents of the image core are described by an RFNoC image core YAML file, and the Verilog code that represents this block is automatically generated by the RFNoC Image Builder tool based on the YAML description. In other words, you provide a description of what you want to be included in RFNoC and the tools generate the code to implement that description.
- **Transport Adapter**
  The transport adapter is a component of the USRP that allows communication with an outside interface. These vary depending on the USRP model in use. For example, this could be a 10 Gbps Ethernet link for SFP+ port of the USRP, or a cabled PCIe interface.
- **CHDR Crossbar**
  The CHDR crossbar is a dynamic router for RFNoC traffic. This is a high-throughput crossbar designed for RF streaming applications. It is reconfigurable at run time via software and can be used to configure routes between stream endpoints and transport adapters. The number of ports available is configurable by changing the RFNoC image core YAML file. CHDR (Condensed Hierarchical Datagram for RFNoC) refers to the network protocol that is used for RFNoC.
- **Stream Endpoint (SEP)**
  The stream endpoint (SEP) provides the high-level flow control for traffic over the network. It also separates control traffic from data traffic to create a separate AXIS-Ctrl network for control traffic. Control traffic refers to things like register reads and writes for configuring and monitoring RFNoC blocks.
- **Control Crossbar**
  The control crossbar is similar to the CHDR Crossbar, but is only for control traffic. It has a much lower throughput and has been optimized for control-traffic, making it much less resource intensive than the CHDR crossbar.
- **Other Blocks**
  The DDC (digital down converter), DUC (digital up converter), and Radio in this example are RFNoC blocks that are included with UHD. These are standard components included the default USRP images and enable typical RF applications. Most blocks only communicate with the RFNoC network, but some blocks require access to external hardware interfaces, such as the radio or DRAM. In this example, the radio blocks connect to the radio hardware on the USRP.

52.6.3

The routes between blocks that go through the crossbar are *dynamic*. That is, the routes can be changed at run time and are controlled by software. The CHDR crossbar is very powerful in that it allows any of its ports to communicate with each other. It allows for streaming between blocks on the same FPGA, between RFNoC blocks and a host computer, or between blocks on different USRPs. New signal processing chains can be added to this crossbar, as needed, for the application.

Similarly, the control crossbar supports dynamic routing, allowing any RFNoC block to send control traffic to any other RFNoC block, even to blocks on a different USRP. Control traffic can also be sent from the host computer to the RFNoC blocks, for example to read/write registers. Additionally, RFNoC blocks can send control to the host computer. In this example, the Radio block sends control traffic to the computer to report specific events, such as overflow or underflow in the radio.

The block connections that don't go through a crossbar (e.g., the connection from the radio to the DDC, and the DDC to the stream endpoint, etc.) are *static* connections. That is, they cannot be changed at run time. Making a connection static has the advantage that it does not require additional ports on the crossbar and that the connection can be made much simpler because the high-level network flow-control protocol used by CHDR is not required. This reduces latency between blocks and leads to FPGA resource savings, allowing more logic be included in a single FPGA image. The static connections are described by the RFNoC image core YAML file. In order to change the static connections, the YAML description needs to be updated and the FPGA image needs to be rebuilt.

52.6.4

The RFNoC framework makes it easy to customize the RFNoC image and add your own IP blocks. Later on in this guide, we'll explore how you can customize the RFNoC image to add or remove blocks, as well as create your own RFNoC blocks and include them in your FPGA builds. This allows you to create highly-customized and high-performance FPGA images for your USRP.

52.7

To use RFNoC in UHD 4.0 you need to perform the following:

- Clone the UHD repository
- Install UHD
- Update your device's filesystem and FPGA image

Building custom RFNoC images requires the FPGA source code and build system. These are included in the UHD repository, located in `<repo>/fpga/usrp3/`, where `<repo>` refers to the location where you cloned the UHD repository.

Please see the Binary Installation for UHD installation documentation or Building and Installing instructions to build and install UHD from source. Additionally, AN-445 provides step-by step instructions for cloning the repository and installing UHD from source.

Starting with UHD 3.15, RFNoC is enabled by default, and starting with UHD 4.0, it can no longer be disabled, so no additional instructions are necessary to install support for RFNoC in UHD.

Many of the UHD utilities we will use in this guide are based on Python. Please ensure that your `PYTHONPATH` is set correctly. If not set correctly, you will see errors indicating that Python could not find the `uhd` or `image_builder` libraries. The correct setting may depend on where you installed UHD and which Linux distribution you are using. If you installed UHD from source to the default location on Ubuntu, you may need to add `/usr/local/lib/python3/dist-packages` to your `PYTHONPATH` variable. For example:

```
$ export PYTHONPATH=/usr/local/lib/python3/dist-packages
```

Instructions for updating your device's filesystem (applicable to E3xx and N3xx) and FPGA image can be found in that device's Getting Started Guide. If your filesystem is already up to date, or your device does not use a filesystem (e.g., X3xx), then instructions on flashing an FPGA image to a device can be found in the following locations:

- X3xx series
- E3xx series
- N3xx series

NOTE: FPGA images are specific to the USRP device, NOT the USRP series. For example, a USRP X300 FPGA image will NOT work on a USRP X310 and vice versa. Loading an image that does not correspond to your USRP device will likely lead to an error message but can brick the device under some circumstances.

52.7.1

Make sure you are using the `Bash` shell. Many of the build scripts used by RFNoC are written for `Bash`. See Reconfigure Default Shell in AN-315 for detailed instructions.

52.8

Before continuing, please verify that you have downloaded and flashed the latest FPGA image for your version of UHD. All FPGA images for Generation-3 and above devices are RFNoC images, so there is no requirement to download a special image for RFNoC.

52.8.1

Before we start customizing our FPGA, let's get familiar with the default FPGA image, which is pre-built with a set of RFNoC blocks.

Run the following command, with your USRP connected to your PC, to see what is available on the device.

```
$ uhd_usrp_probe --args type=x300
```

Note that your `args` may be different, depending on the USRP device you're using. The example here works for X300/X310. Refer to the manual page on Identifying USRP Devices for more details. If an RFNoC image was successfully loaded onto the USRP, the output will show something like the following:

```
  ___________________________________________________
 /
|       Device: X-Series Device
|     _____________________________________________________
|    /
|   |       Mboard: X310
|   |   revision: 11
|   |   revision_compat: 7
|   |   product: 30818
|   |   mac-addr0: 00:80:2f:17:40:6d
|   |   mac-addr1: 00:80:2f:17:40:6e
|   |   gateway: 192.168.10.1
|   |   ip-addr0: 192.168.10.2
|   |   subnet0: 255.255.255.0
|   |   ip-addr1: 192.168.20.2
|   |   subnet1: 255.255.255.0
|   |   ip-addr2: 192.168.30.2
|   |   subnet2: 255.255.255.0
|   |   ip-addr3: 192.168.40.2
|   |   subnet3: 255.255.255.0
|   |   serial: 311EF81
|   |   FW Version: 6.0
|   |   FPGA Version: 38.0
|   |   FPGA git hash: be53058
|   |
|   |   Time sources:  internal, external, gpsdo
|   |   Clock sources: internal, external, gpsdo
|   |   Sensors: ref_locked
|     _____________________________________________________
```

```
|    /
|   |       RFNoC blocks on this device:
|   |
|   |   * 0/DDC#0
|   |   * 0/DDC#1
|   |   * 0/DUC#0
|   |   * 0/DUC#1
|   |   * 0/Radio#0
|   |   * 0/Radio#1
|   |   * 0/Replay#0
|    ____________________________________________________
|    /
|   |       Static connections on this device:
|   |
|   |   * 0/SEP#0:0==>0/DUC#0:0
|   |   * 0/DUC#0:0==>0/Radio#0:0
|   |   * 0/Radio#0:0==>0/DDC#0:0
|   |   * 0/DDC#0:0==>0/SEP#0:0
|   |   * 0/Radio#0:1==>0/DDC#0:1
|   |   * 0/DDC#0:1==>0/SEP#1:0
|   |   * 0/SEP#2:0==>0/DUC#1:0
|   |   * 0/DUC#1:0==>0/Radio#1:0
|   |   * 0/Radio#1:0==>0/DDC#1:0
|   |   * 0/DDC#1:0==>0/SEP#2:0
|   |   * 0/Radio#1:1==>0/DDC#1:1
|   |   * 0/DDC#1:1==>0/SEP#3:0
|   |   * 0/SEP#4:0==>0/Replay#0:0
|   |   * 0/Replay#0:0==>0/SEP#4:0
|   |   * 0/SEP#5:0==>0/Replay#0:1
|   |   * 0/Replay#0:1==>0/SEP#5:0
...
```

More than this will likely be shown. The specifics of the output depend on the UHD version and which device you're running on. However, we're interested in the following sections:

- **Device description**. At the top, there is a section with basic information about your device, such as serial number, revision, etc. You can use this information to verify that you are indeed communicating with the correct device, and that the device has been updated. The FPGA git hash identifies the commit from which the FPGA image was built.
- **RFNoC blocks on this device**. This section lists all the RFNoC blocks in the loaded FPGA image. The blocks are listed by their block IDs. For example, `0/Radio#0` means that we're on device zero (`0/`), and we're talking about the first radio (with index `#0`). Unless you probe multiple devices at once by specifying multiple addresses in the `--args` argument, the device number will always be zero, but the block index will change depending on the number of blocks of that type. For example, on X310, `Radio#0` corresponds to RF A and `Radio#1` corresponds to RF B.
- **Static connections on this device**. This section lists how the blocks are pre-connected in the FPGA image. For example, the line `0/Radio#0:0==>0/DDC#0:0` means that radio zero, port zero (`:0`), is connected statically to DDC zero, port zero. Static connections can only be changed at compile time when building the FPGA image.

The connection endpoints labeled `SEP` are not RFNoC blocks. They are Stream Endpoints (SEPs). SEPs can send data packets to one another, or to streamers in software, using the CHDR crossbar, which is a dynamic router on the FPGA.

52.8.2

Many RFNoC blocks come with UHD. The HDL source code for these blocks resides in `<repo>/fpga/usrp3/lib/rfnoc/blocks/`. several of the blocks are described below.

Block Name	HDL Name	Description
AddSub	`rfnoc_block_addsub`	Add/Subtract Verilog/VHDL/HLS Example
DmaFIFO	`rfnoc_block_axi_ram_fifo`	FIFO that uses an AXI4 memory-mapped interface for storage. For use with external DRAM or on-chip SRAM.
DDC	`rfnoc_block_ddc`	Digital Down Converter
DUC	`rfnoc_block_duc`	Digital Up Converter
FFT	`rfnoc_block_fft`	Fast Fourier Transform
FIR	`rfnoc_block_fir_filter`	Finite Impulse Response Filter
Fosphor	`rfnoc_block_fosphor`	FFT and waterfall display tool
KeepOneInN	`rfnoc_block_keep_one_in_n`	Keep one sample/packet in N
LogPwr	`rfnoc_block_logpwr`	Computes an estimate of log2(i^2+q^2)
MovingAverage	`rfnoc_block_moving_avg`	Outputs the running average of the N most recent inputs of data stream
NullSrcSink	`rfnoc_block_null_src_sink`	Data source generator, sink, and loopback for testing
Radio	`rfnoc_block_radio`	Radio Interface
Replay	`rfnoc_block_replay`	Record/Playback using AXI4 memory-mapped interface. For use with external DRAM or on-chip SRAM.
SigGen	`rfnoc_block_siggen`	Signal Generator. Supports sinusoidal, constant, and random outputs, with configurable gain.
SplitStream	`rfnoc_block_split_stream`	Splits a single data stream into two
Switchboard	`rfnoc_block_switchboard`	A configurable RFNoC datapath switch for testing
VectorIIR	`rfnoc_block_vector_iir`	Implements an IIR filter with variable length delay line
Window	`rfnoc_block_window`	Windowing module for use with FFT block

52.8.3

In the UHD installation directory, you'll find example applications (e.g., in `/usr/lib/uhd/examples/` or `/usr/local/lib/uhd/examples/`). We'll look at `rfnoc_rx_to_file` to familiarize ourselves with the RFNoC API.

If we look at the source code for this example (located at `<repo>/host/examples/rfnoc_rx_to_file.cpp`), we can see the components necessary in an RFNoC application. In the `UHD_SAFE_MAIN` function, we create a few crucial objects, such as an RFNoC graph, a radio block controller, a DDC block controller, and an RX streamer.

```
auto graph = uhd::rfnoc::rfnoc_graph::make(args);
// ...
auto radio_ctrl = graph->get_block<uhd::rfnoc::radio_control>(radio_ctrl_id);
```

```
// ...
uhd::rfnoc::ddc_block_control::sptr ddc_ctrl;
// ...
auto rx_stream = graph->create_rx_streamer(1, stream_args);
```

We also make connections in our graph and configure our blocks.

```
// Connect blocks and commit the graph
for (auto& edge : chain) {
    if (uhd::rfnoc::block_id_t(edge.dst_blockid).match(uhd::rfnoc::NODE_ID_SEP)) {
        graph->connect(edge.src_blockid, edge.src_port, rx_stream, 0);
    } else {
        graph->connect(
            edge.src_blockid, edge.src_port, edge.dst_blockid, edge.dst_port);
    }
}
...
rate = radio_ctrl->set_rate(rate);
...
radio_ctrl->set_rx_frequency(freq, radio_chan);
```

Once our graph and blocks are configured, we use the `recv_to_file` function to receive data from our device and write it to a file on our host computer. Running this example with the `--help` argument shows the available options. For example, to receive 3 seconds of data at a specific frequency and sample rate, we can run:

```
rfnoc_rx_to_file --args type=x300 --freq 2.4e9 --rate 10e6 --duration 3
```

Now you are ready to move to the next step and build your own blocks and FPGA images.

52.9

UHD provides tooling to help develop custom RFNoC images. In this guide we will demonstrate how to use the RFNoC Image Builder, which will allow you to change which blocks are included and the connections between blocks. Let's go over common decisions you'll have to make with this tool.

- **Blocks Included**
  The particular blocks to be included in the image is the most impactful decision you'll make in the image building process, both in terms of image functionality and FPGA resource utilization. Adding more blocks means more processing in the image, but it also means longer FPGA build times and less space in the FPGA.
- **Static Connections**
  Recall that blocks can be statically connected, as opposed to connecting every block to the CHDR crossbar (dynamically connected). The number of dynamic connections has a large impact on the size of the CHDR crossbar, so static connections are used to lower resource utilization. However, once a block is statically connected in a chain of blocks, data cannot be sent between arbitrary blocks; it must traverse and be processed by each block in the order defined by the static connections.
- **Hardware Connections**
  All blocks share some connections defined by the RFNoC Specification. These connections are made automatically and cannot be changed. Many blocks require additional connections, such as clocks for the internal DSP or external DRAM interfaces. These connections must also be specified.

Now that we've gone over some of the considerations, let's look at how to actually build a custom RFNoC image. This begins with the RFNoC image core YAML description.

52.9.1

The image YAML file defines RFNoC blocks in the FPGA image. It is designed to be both human-readable and machine-parseable, which allows us to make edits quickly and easily. Each device type has a default YAML image file, which is named `<DEVICE>_rfnoc_image_core.yml` (e.g., `x310_rfnoc_image_core.yml`). Take a look at the RFNoC image core YAML file for the X310 by opening `<repo>/fpga/usrp3/top/x300/x310_rfnoc_image_core.yml`. Notice that it has the following sections.

- **General Parameters**
  Here we define the device, the bit width of our CHDR connections, as well as some versioning and licensing.
- **Stream Endpoints**
  In the `stream_endpoints` section we list each stream endpoint (SEP) that we wish to instantiate. These will be directly connected to the CHDR crossbar and will allow us to make dynamic connections within the CHDR crossbar. Typically you will have one SEP that forms the start and end of a single DSP chain of RFNoC blocks.
  You can also add some per-stream-endpoint configuration here, such as the ingress buffer size, which affects streaming performance from your computer to that SEP. For example, if we know that one SEP will be receiving data transferred from your computer to the USRP then a data buffer is needed to accept those incoming packets, in which case we specify the buffer size by setting the `buff_size` option on that SEP. Alternatively, if we know that a particular SEP only sends data from the USRP to the computer, then we won't need the ingress data buffer and we can set `buff-size` to 0, thus saving FPGA resources.
  Each SEP can have an AXIS-Ctrl and an AXIS-CHDR port, as indicated by the `ctrl` and `data` options. At least one AXIS-Ctrl port is required to communicate with the RFNoC blocks, so `ctrl` typically enabled on just the first SEP. Every SEP will usually have AXIS-CHDR connections to one or more RFNoC blocks, so `data` is usually enabled on all SEPs.
- **NoC Blocks**
  In the `noc_blocks` section we specify all the RFNoC blocks to include in the FPGA image. We'll need to give each block a unique name and reference a block definition YAML file (which we'll discuss in a later section). The blocks chosen have the greatest impact on the functionality of the image, as well as providing very coarse control over the FPGA resource utilization. We'll look at how to change the blocks included in the image as part of our example below.
- **Static Connections**
  The `connections` section defines static connections between blocks, stream endpoints, and various hardware interfaces on the USRP. Statically connected chains of blocks will usually start and end at a stream endpoint (SEP). SEPs are automatically connected to the CHDR crossbar by the RFNoC infrastructure. Remember that data must be passed through the entire statically connected chain; dynamic connections cannot be made to the statically connected blocks in the middle of the chain. Other hardware connections, such as to the external DRAM, would also be specified here. We'll take a closer look at making connections as part of our example below.
- **Clock Domains**
  The `clk_domains` section defines which clocks to connect to each block's clock inputs. Some blocks do not need any clock connections beyond the base clocks required by RFNoC. These required connections are not listed here, since they are always the same for each block. Other blocks may require additional clocks. For example, the radio blocks should be connected to the `radio` clock. Many other blocks require a `ce` (Compute Engine) clock, which is used for the block's internal DSP.

52.9.2

Before we look at editing the image core YAML file, let's go over how to use the RFNoC image builder to generate a bitstream from that YAML file. Additional details on how to run the image builder can be found with the `--help` option:

```
$ rfnoc_image_builder --help
```

Here are some of the options provided:

- **`-y`**: Path to the RFNoC image core YAML file
- **`-t`**: The image target you would like to build. These are the same targets that are used in the FPGA `make` process. More details on these can be found in the Generation 3 USRP Build Documentation of the UHD and USRP Manual. If not specified, the default specified in the image core YAML file will be used.
- **`-l debug`**: Use the `debug` log level. This prints more information about the FPGA connections and available port names, which can be useful for debugging connection errors in the YAML file.
- **`--generate-only`**: Generate the HDL files required, but do not build the FPGA. Users can then build the FPGA later using `make <target>`.
- **`-I`**: Path to the directory containing out-of-tree block YAML descriptions (the YAML files installed with UHD are included by default). This option is only needed if using an out-of-tree RFNoC block.
- **`-F`**: Path to the FPGA source code (e.g., `<repo>/fpga`). This path is only required if the current working directory is not within the UHD repository.

For example, to build the default RFNoC image for X310, you might use the following command:

```
$ cd <repo>/fpga/usrp3/top/x300/
$ rfnoc_image_builder -y ./x310_rfnoc_image_core.yml -t X310_XG
```

In this example `<repo>` refers to the location where you cloned the UHD repository and should be replaced by the location you used. `x310_rfnoc_image_core.yml` is the default RFNoC image core YAML file for the X310, which is in the x300 project directory. `X310_XG` is the make target to use for the build. `XG` in this case refers to dual 10 Gbps Ethernet.

For an out-of-tree RFNoC block, you will also need to specify the location of the block information. For example:

```
$ rfnoc_image_builder -F <repo>/fpga -I <repo>/host/examples/rfnoc-example -y <repo>/host/examples/rfnoc-example/icores/x310_rfnoc_image_c
```

This example shows how to build an FPGA with the Gain example out-of-tree RFNoC block, which is located in `<repo>/host/examples/rfnoc-example/`. The `-F` option is added to specify the location of the FPGA source, and `-I` specifies the location of the out-of-tree block YAML.

The image builder performs several steps in order to build the FPGA image from the image coure YAML:

1. It generates the `<device>_rfnoc_image_core.v` file. This file includes the Verilog code described by the YAML image core file. A `<device>_static_router.hex` file is also generated. This file describes the static connections that should be made by the Verilog code. In the case of the X310 examples above, the output files would be named `x310_rfnoc_image_core.v` and `x310_static_router.hex`, and would be placed in the project directory for the X310 (`<repo>/fpga/usrp3/top/x300/`).
2. The image builder will configure the environment for building the FPGA. This is equivalent to sourcing the `setupenv.sh` script for the device type being built. For example, in our example above, it would run `source <repo>/fpga/usrp3/top/x300/setupenv.sh`.
3. The image builder runs `make <target>` to build the IP and the FPGA bitstream for the indicated target. The generated `rfnoc_image_core.v` and `static_router.hex` are automatically pulled into this build. The completed bitstream will be located in `<repo>/fpga/usrp3/top/{project}/build` directory, which is the same directory created through the normal make process. For example, for X310 it would be located in `<repo>/fpga/usrp3/top/X300/build`.

52.9.3

As an example, let's run through how to modify the YAML file to modify the RFNoC image. Suppose we have an application that we'd like to run on a USRP X310 and we would like to offload the FFT processing to the FPGA. UHD provides an FFT RFNoC block, and the default X310 image has some extra space in it, so we should be able to add this block. First, we copy the default X310 image core file, named `x310_rfnoc_image_core.yml`.

```
$ cd <repo>/fpga/usrp3/top/x300/
$ cp x310_rfnoc_image_core.yml x310_with_fft.yml
```

Now open `x310_with_fft.yml` in your favorite text editor. We'll start by making some room for our new block. The default images use very large ingress FIFO buffers for the main SEPs to maximize streaming performance. But we want to make sure we have enough memory buffer space for our new blocks. So Start by reducing the `buf_size` parameters for `ep0` and `ep2` from 65536 to 32768. If using a device other than X310, the numbers may be different, but you can similarly reduce the `buff_size` parameter by half. On smaller devices, it may be necessary to remove the Replay block to make room for the FFT blocks. After making the changes on X310, the result should look like the following:

```
stream_endpoints:
  ep0:                      # Stream endpoint name
    ctrl: True                     # Endpoint passes control traffic
    data: True                     # Endpoint passes data traffic
    buff_size: 32768               # Ingress buffer size for data
  ...
  ep2:                      # Stream endpoint name
    ctrl: False                    # Endpoint passes control traffic
    data: True                     # Endpoint passes data traffic
    buff_size: 32768               # Ingress buffer size for data
```

Now we can add our FFT block. We'll put it on its own stream endpoint, so we first need to add a new stream endpoint. Add the following to the `stream_endpoints` section:

```
stream_endpoints:
  ...
  ep_fft:                # The name can be incremented from previous SEP
    ctrl: False          # Only the first SEP needs control traffic
    data: True           # We do want to pass data through this SEP
    buff_size: 32768     # Ingress buffer size for data
```

In this example, we've named the SEP `ep_fft`. Other names could be given, as long as the name is unique (it does not have to be numbered). Now that we've allocated an SEP for our block, we need to instantiate the actual block. In the next section, add the following:

```
noc_blocks:
  ...
  fft0:                          # FFT block name
    block_desc: 'fft_1x64.yml'   # Block YAML descriptor file
    parameters:                  # Specify any Verilog module parameters (optional)
```

```
      EN_FFT_SHIFT: 1
```

Again, the name `fft0` is arbitrary, but the name must be unique. In some cases, there will also be block parameters you'll want to pass, such as the data format of the FFT output data. In our example, we're setting `EN_FFT_SHIFT` parameter to 1, which causes the FFT block to center the zero frequency bin.

Now that we've added our FFT block, we need to connect it to the stream endpoint. In the next section of the YAML file, we add the static connections:

```
connections:
  ...
  - { srcblk: ep_fft, srcport: out0,  dstblk: fft0,   dstport: in_0 }
  - { srcblk: fft0,   srcport: out_0, dstblk: ep_fft, dstport: in0  }
```

This connects the output of SEP `ep_fft` to the input of block `fft0`, and the output of `fft0` to the input of `ep_fft`. Since we're placing the FFT block on its own SEP, the only connections we need to make are between the SEP and the FFT. All SEPs are automatically connected to the CHDR crossbar, so this effectively connects the FFT block to the crossbar, allowing it to communicate with anything on the RFNoC network.

The names of block ports are defined in the YAML descriptions for the blocks. Blocks can use any names for their ports, and they don't have to be numbered (unless the number of ports is parameterized). Generally, the block ports are named `in_N` for inputs to the block and `out_N` for outputs. SEP ports are named `in0` for the input `out0` for the output.

If you are having trouble connecting a block due to an unresolved connection, running `rfnoc_image_builder` with the `-l debug` option will list all available block ports.

Finally, the FFT block has an additional clock input port named `ce` that is used for the FFT signal processing. We need to connect it to a clock domain. Any clock that's at least as fast as the incoming data rate should be sufficient. For example, X3xx devices have a `ce` clock (214.286 MHz) that is usually a good choice, but `rfnoc_chdr` clock (200 MHz) should also work for this example. For N31x and E3xx devices, `rfnoc_chdr` clock is a good choice (200 MHz on N31x and 100 MHz on E3xx). For N32x, `radio` clock (250 MHz) is a good choice. For example, this is how you would connect the X310's `ce` clock to the `ce` port of the FFT block:

```
clk_domains:
  ...
  - { srcblk: _device_, srcport: ce, dstblk: fft0, dstport: ce }
```

And this is how you would connect `rfnoc_chdr` clock to the `ce` port:

```
clk_domains:
  ...
  - { srcblk: _device_, srcport: rfnoc_chdr, dstblk: fft0, dstport: ce }
```

And finally, this is how you would connect `radio` clock:

```
clk_domains:
  ...
  - { srcblk: _device_, srcport: radio, dstblk: fft0, dstport: ce }
```

The source block name `_device_` is special and refers to the USRP device itself. Choose a clock that's apprpriate for your device and connect it as shown above.

And that's it! The next step will be to run the image builder on our modified YAML file. Once that's done, and the bitstream has been created, you can load it onto your USRP X310 device, and verify the blocks, as we did in the Inspect the Default Image section.

For example, from the X300 directory, you would run the following command to run the image builder:

```
$ rfnoc_image_builder -y x310_with_fft.yml -t X310_XG
```

To download the FPGA bitstream to the X310, run the following:

```
$ uhd_image_loader --args "type=x300,addr=192.168.30.2" --fpga-path ./build/usrp_x310_fpga_XG.bin
```

You may need to change the IP address to match your device, depending on your configuration. After completing the flash update and power-cycling the USRP, run `uhd_usrp_probe` to confirm that the FFT block was recognized.

```
$ uhd_usrp_probe --args "type=x300,addr=192.168.30.2"
```

Take a look at the RFNoC blocks and the static connections on the device. You should see the following new blocks and connections:

```
|     ____________________________________________________
|    /
|   |       RFNoC blocks on this device:
...
|   |   * 0/FFT#0
...
|     ____________________________________________________
|    /
|   |       Static connections on this device:
...
|   |   * 0/SEP#6:0==>0/FFT#0:0
|   |   * 0/FFT#0:0==>0/SEP#6:0
...
```

52.9.4

Now that we've added the FFT block and verified that it is operating as expected, let's see if we can modify it a little. Let's say that we're sure that we always want to receive the FFT bins from our device, and we don't want to see the raw samples. In order to save some FPGA resources, we can move our FFT block to be between the DDC and the SEP, on the RX data path. Instead of the previous modifications to our YAML file, we want the following:

```
stream_endpoints:
  ...
  # Unchanged from the default image core (no need for ep_fft).

noc_blocks:
  ...
  fft0:                          # FFT block name
    block_desc: 'fft_1x64.yml'   # Block YAML descriptor file
    parameters:                  # Specify any Verilog module parameters (optional)
      EN_FFT_SHIFT: 1
```

```
connections:
  ...
  # Change this line:
  # - { srcblk: ddc0, srcport: out_0, dstblk: ep0, dstport: in0  }
  # Change it to the following to add fft0 between ddc0 and ep0:
  - { srcblk: ddc0, srcport: out_0, dstblk: fft0, dstport: in_0 }
  - { srcblk: fft0, srcport: out_0, dstblk: ep0,  dstport: in0 }

clk_domains:
  ...
  # As before, we still connect our FFT block to the clock domain
  - { srcblk: _device_, srcport: rfnoc_chdr, dstblk: fft0, dstport: ce }
```

This last line is valid for E310/E320; for X300/X310/N300/N310/N320/N321 use the following:

```
  - { srcblk: _device_, srcport: ce, dstblk: fft0, dstport: ce }
```

And as simply as that, we have added the FFT block to our RX block chain. Remember that this is a static connection, so when the resulting FPGA image is loaded onto your device, you will no longer be able to receive samples from the DDC directly; all samples on that chain will be processed by the FFT block, and you will only receive FFT bins from this radio chain.

52.10

52.10.1

Now that we've gone over what is provided by UHD, let's start to look at custom RFNoC development. It's recommended that custom RFNoC development be kept separate from the in-tree UHD RFNoC infrastructure and examples, in order to simplify version control and licensing. Modules located outside of the repository are called *out-of-tree* (OOT).

In order to understand how OOT modules for RFNoC are created, let's take a look at the example that's provided with UHD. You can find the example located in `<repo>/host/examples/rfnoc-example/`. This example includes a simple Gain RFNoC block that we can use as a reference for creating our own OOT blocks.

52.10.2

Take a look at the `rfnoc-example` directory structure. An OOT module is a collection of block implementations, the software controllers for those blocks, and sometimes applications to demonstrate their functionality. These different components within the OOT module are organized into the directories described below.

- HDL and Image Resources
  - `fpga/`: Contains the HDL (e.g., Verilog) required for the module. Each RFNoC block should have its own subdirectory here, such as `rfnoc_block_gain`, for the Gain block.
  - `blocks/`: Contains the YAML RFNoC block definition files. These describe the block's interfaces and are used by the image builder. We'll go into more detail with regards to what goes into these YAML files in the Creating Your Own RFNoC Block section.
  - `icores/`: Contains example FPGA image core YAML files, to demonstrate how to create an FPGA image using your custom blocks. These image core files may demonstrate important parameters or show recommended configurations for chains of blocks.
- Block Controller and Example Software
  - `include/`: Contains the headers for the block controller software. Once installed, these headers will be used by UHD and other applications to interface with and control your custom RFNoC blocks.
  - `lib/`: Contains the block controller implementations. The files here should implement the features outlined in the headers located in the `include/` directory. All unit test code should also be located here.
  - `apps/`: Contains example applications for your custom blocks. These applications may demonstrate how to use the block controllers, common configurations, or how to process data from your blocks.
- Infrastructure
  - `cmake/`: Contains any custom CMake commands the module may need. In general, simple modules will not need to modify this directory at all. More complicated modules with additional external dependencies may need to modify this.

Take some time to explore the files in the `rfnoc-example`. Note the following files and directories related to the Gain example:

- `rfnoc-example/fpga/rfnoc_block_gain/`
  This is the HDL for the Gain RFNoC block.
  - `rfnoc_block_gain.v`
    The top-level synthesizable file for the Gain block
  - `noc_shell_gain.v`
    The NoC Shell for the Gain block
  - `rfnoc_block_gain_tb.sv`
    The simulation testbench
  - `Makefile`
    The simulation Makefile
- `rfnoc-example/blocks/gain.yml`
  The YAML block definition file for the Gain block
- `rfnoc-example/icores/x310_rfnoc_image_core.yml`
  An example RFNoC image core YAML description showing how to connect the Gain block
- `rfnoc-example/include/rfnoc/example/gain_block_control.hpp`
  The software block controller header
- `rfnoc-example/lib/gain_block_control.cpp`
  The software block controller implementation
- `rfnoc-example/apps/init_gain_block.cpp`
  An example showing how to find and initialize the Gain block

52.10.3

To create your own OOT module, make a copy of the `rfnoc-example` directory. This will contain all the subdirectories and infrastructure files that you need to create your custom module. The directory name for your copy should be renamed to your desired name; we recommend something like `rfnoc-foo`, for example, where `foo` is the name for your OOT module. You'll also need to change `rfnoc-example` within the module's `CMakeLists.txt` files to your module's name.

Let's start by creating our own copy of the `rfnoc-example` to work with. We'll call our module `demo`. You can put the copy wherever you like, but for this example we'll put it in our home directory.

```
$ cp -r <repo>/host/examples/rfnoc-example ~/
$ mv ~/rfnoc-example ~/rfnoc-demo
```

Now we need to edit our `CMakeLists.txt` files to rename our module. Edit the following files and change all instances of `rfnoc-example` to `rfnoc-demo`:

```
~/rfnoc-demo/CMakeLists.txt
~/rfnoc-demo/lib/CMakeLists.txt
```

52.10.4

The OOT module will need to be built and installed in order to use its RFNoC blocks. To build and install the OOT module we just created, do the following:

```
$ mkdir ~/rfnoc-demo/build
$ cd ~/rfnoc-demo/build
$ cmake -DUHD_FPGA_DIR=<repo>/fpga/ ../
```

This configures the project to be installed in the default location. You may want to provide a different install directory to CMake using the `-DCMAKE_INSTALL_PREFIX`. Please refer to the UHD Installation Guide or the installation guide for other CMake-based projects for instructions on configuring CMake.

Note that we specify the location of the FPGA source code using the `-DUHD_FPGA_DIR` option. This allows you to run the FPGA testbenches and to build the example FPGA image provided with the OOT example.

At this point, you can run `make help` to see what options are available. A few of the available make targets are described below:

- `make rfnoc-demo`: Build the block controllers for the RFNoC blocks
- `make testbenches`: Run the testbenches for the RFNoC blocks
- `make x310_rfnoc_image_core`: Build the FPGA example image
- `make install`: Install the module

To build and install the block, perform the following steps.

```
$ cd ~/rfnoc-demo/build
$ make
$ make install
```

Note that `sudo` may be required depending on the install location.

52.10.5

At this point you can build the example FPGA that's included in the `icores` directory. Take a look at `~/rfnoc-demo/icores/x310_rfnoc_image_core.yml`. Note that the `gain` block is added in exactly the same way that the FFT block was in Example: Adding an FFT Block. You can build this example FPGA for the X310 using the following steps:

```
$ cd ~/rfnoc-demo/build
$ make x310_rfnoc_image_core
```

Alternatively, you can call `rfnoc_image_builder` directly. In order to get the image builder to find your custom blocks, you may need to supply some additional arguments to find the OOT module, as shown in the following example:

```
$ rfnoc_image_builder -F <repo>/fpga ?I ~/rfnoc-demo/include/rfnoc -y ~/rfnoc-demo/icores/x310_rfnoc_image_core.yml
```

The `-I` option is only required if the OOT module has not been installed on the system.

If you have an X310, you can build and use the provided example to test the Gain block. If you have a different USRP, you can use its default `rfnoc_image_core.yml` as a starting point and follow the same steps that we followed for the FFT block in Example: Adding an FFT Block, then build it using the process described above.

52.10.6

After building an FPGA image for your USRP and downloading it to your device, the Gain block should be available. Test this by running `uhd_usrp_probe`:

```
$ uhd_usrp_probe --args "type=x300,addr=192.168.30.2"
```

Again, note that your `args` may be different or may not be required if you only have a single USRP connected.

Take a look at the RFNoC blocks and the static connections on the device. You should see the following new blocks and connections:

```
|     ____________________________________________________
|    /
|   |       RFNoC blocks on this device:
...
|   |   * 0/Block#0
...
|     ____________________________________________________
|    /
|   |       Static connections on this device:
...
|   |   * 0/SEP#4:0==>0/Block#0:0
|   |   * 0/Block#0:0==>0/SEP#4:0
...
```

52.11

The Gain block is a useful example and could even be used as the starting point for a new block, but rather than trying to copy and edit the Gain block, it is best to use the RFNoC ModTool to create a new block template. This is also required if you want to change the interfaces provided to your IP that are exposed by the NoC Shell.

52.11.1

The first step in creating a new block is to write a YAML description for the block that you want to create. The options available are described RFNoC Specification. You can also look at the blocks available in `<repo>/host/include/uhd/rfnoc/blocks/` for examples. We'll provide a brief overview here.

Open up and take a look at the Gain block definition located in `~/rfnoc-demo/blocks/gain.yml`. Notice the different sections, which are described below:

- **General Parameters**
  In this section we give our block a name (`module_name`) and a unique block ID (`noc_id`). The NoC ID is an arbitrary 32-bit number that will be used by the software to identify the block during system discovery. You can also specify the CHDR bus width, which should be 64 for Generation-3 USRPs.
- **Clocks**
  In the `clocks` section, we define the clocks that will be used by the block. The `rfnoc_chdr` and `rfnoc_ctrl` clocks are required. Many blocks also have a `ce` clock for internal DSP.
- **Control Interface**
  The `control` section defines what type of control interface to make available to your block. The control interface is used for register reads and writes. Most blocks do not need to modify this section and will use the `ctrlport` interface type. Control Port is the standard register interface used by RFNoC. The `clk_domain` parameter allows you to specify which clock domain to use for the register interface exposed to your block. Typically this is the same clock that is used for the `data` interface, to avoid clock crossings.
- **Data Interface**
  The `data` section defines what type of data interface to make available to your block and describes the ports. Most blocks use either the `axis_pyld_ctxt` or the `axis_data` interface types.
  The `clk_domain` parameter allows you to specify which clock domain to use for the data interfaces exposed to your block. Typically this is the same clock that is used for the `control` interface, to avoid clock crossings.
  Under the `inputs` and `outputs` sections you can describe the input and output ports.
- **IO Ports**
  The `io_ports` section is used to describe other device-specific ports to which your block needs to connect (such as the DRAM or radio interfaces). For most blocks, this section is not used.

A few other sections might be present in the YAML file (e.g., `registers` or `properties`). These sections are reserved for future use and can be left empty.

52.11.2

Let's start by making a copy of the Gain block YAML for our use case.

```
$ cp ~/rfnoc-demo/blocks/gain.yml ~/rfnoc-demo/blocks/demo.yml
```

Now open the `demo.yml` file you just created and make the following changes:

1. Change the name of the block from `gain` to `demo`
2. Change the `noc_id` to `0x0000DE30`
3. Change the `format` for the `in` and `out` ports from `int32` to `sc16`.

This YAML file now describes a block with the following features:

- The block named "demo"
- NoC ID of 0xDE30
- CtrlPort register interface on the `rfnoc_chdr` clock domain
- A single 32-bit input port named "In" on the `rfnoc_chdr` clock domain that will provide sc16 samples to your IP
- A single 32-bit output port named "Out" on the `rfnoc_chdr` clock domain that will receive sc16 samples from your IP

52.11.3

Now that we have a block definition YAML file, we can generate the source code templates and NoC Shell for our block. To do so, run the following command:

```
$ python3 <repo>/host/utils/rfnoc_blocktool/rfnoc_create_verilog.py -c ~/rfnoc-demo/blocks/demo.yml -d ~/rfnoc-demo/fpga/rfnoc_block_demo
```

This will create a folder named `~/rfnoc-demo/fpga/rfnoc_block_demo` with an RFNoC block template for you to use. Explore the folder that was created. You should see the following files:

- **`Makefile`**: This is the Makefile for the simulation testbench. See Running a Testbench in the UHD and USRP Manual for instructions on how to run a testbench.
- **`Makefile.srcs`**: This is another makefile that identifies the HDL source code that makes up your RFNoC block.
- **`noc_shell_demo.v`**: This is the NoC Shell that was generated for your block. It provides the interfaces described in the block definition YAML.
- **`rfnoc_block_demo.v`**: This is a template for your RFNoC block. It includes all the top-level ports required by RFNoC and instantiates the NoC Shell.
- **`rfnoc_block_demo_tb.sv`**: This is a testbench template for your RFNoC block. It instantiates your RFNoC block and the bus functional models (BFMs) needed to communicate with your block in simulation.

Take a moment to look at the `rfnoc_block_demo.v` and `rfnoc_block_demo_tb.sv` files that were generated for your block. Notice the "User Logic" section towards the end of the `rfnoc_block_demo` module. This is the location where you can insert your own IP.

The input data samples come in on the `m_in_payload` AXI4-Stream bus. The output data samples go out on the `s_out_payload` AXI4-Stream data bus.

If your input packets are the same size as your output packets (i.e., for every sample in you generate one sample out) then the `m_in_context` data bus can be used to drive `s_out_context`. The context bus is described in detail in the RFNoC Specification.

Similarly, the testbench has a section where you can add your own test sequences to the test bench.

If you decide you need to change the RFNoC interfaces for your block (e.g., add ports, or change the interface type), then it is recommended to update the block definition YAML then regenerate the block. Please take care to not overwrite any code you have written when regenerating the files for your block! Rerunning the tool will cause new files to be generated, with a new NoC Shell, and will demonstrate how to update your code for the new interfaces.

Refer to the Gain example and other RFNoC blocks for examples of how to complete your RFNoC block logic and testbenches.

52.12

In this guide we have given a brief introduction on how to develop for RFNoC in UHD 4.0. There's a lot more you can do than is described here, but this will hopefully get you started. Happy coding!

53 RFNoC

1. REDIRECT RFNoC (UHD 3.0)

54 GNU Radio

GNU Radio is a free software development framework that provides signal processing functions for implementing software defined radios. The framework offers a graphical design approach in addition to supporting development in Python and C++. Supported globally by the open-source community and widely used in government, commercial and academic environments, GNU Radio gives users access to a diverse set of existing projects focused on wireless communications research and implementation of real-world radio systems.

For more information on GNU Radio, please visit the GNU Radio Wiki.

The Comprehensive GNU Radio Archive Network (CGRAN) is a free open source repository for 3rd party GNU Radio applications a.k.a Out Of Tree ("OOT") Modules that are not officially supported by the GNU Radio project.

55 LabVIEW

LabVIEW Communications System Design Suite, designed by National Instruments?, supports select USRP motherboard and daughterboard configurations with the goal of accelerating productivity by providing a seamless tool flow from the desktop PC to FPGA. The software combines intuitive graphical programming with tools for managing complex system configurations, multi-rate DSP design of the FPGA and float-to-fixed point conversion. Application Framework add-ons and reference designs are available for LTE, 802.11, and high channel count MIMO configuration. LabVIEW supports the USRP X300/X310, N200/N210, B200/B210, NI-USRP and USRP-RIO devices. For more information on the USRP-RIO, please see ni.com/sdr/usrp-rio. For technical support, please see ni.com/sdr/support.

Visit ni.com/sdr to learn more.

56 Simulink

MATLAB® and Simulink® connect to the USRP family of software-defined radios to provide a radio-in-the-loop environment for SISO and MIMO wireless system design, prototyping, and verification. Communications System Toolbox? supports the USRP X300/X310, N200/N210, and B200/B210 SDRs to transmit and receive RF signals in real time, enabling the use of MATLAB and Simulink to configure radio parameters, generate waveforms, design algorithms, and measure and analyze signals. For the B200mini and UBX, be sure to use Matlab R2016a. for further details, contact mathworks technical support at support@mathworks.com.

To learn more, visit their USRP Support from Communications System Toolbox page.

57 OpenBTS

57.1

OpenBTS (Open Base Transceiver Station) is a software-based GSM access point, allowing standard GSM-compatible mobile phones to be used as SIP endpoints in Voice over IP (VOIP) networks. OpenBTS is an open-source software that was developed and is maintained by Range Networks. The public release of OpenBTS is notable for being the first free-software implementation of the lower three layers of the industry-standard GSM protocol stack. It is written in C++ and released as free software under the terms of version 3 of the GNU Affero General Public License.

OpenBTS is supported on the USRP B200, B210, B200mini, N200, N210, X300, X310 devices, with the WBX, SBX, CBX, UBX daughterboards.

57.2

- OpenBTS Wikipedia
- OpenBTS.org
- OpenBTS Quick Start Guide
- OpenBTS Radio Integration
- OpenBTS Wiki
- OpenBTS Build Install Run Guide
- OpenBTS Github

58 Eurecom OpenAirInterface (OAI)

58.1

EURECOM has created the OpenAirInterface (OAI) Software Alliance (OSA), a separate legal entity from EURECOM, which aims to provide an open-source ecosystem for the core (EPC) and access-network (EUTRAN) protocols of 3GPP cellular systems with the possibility of interoperating with closed-source equipment in either portion of the network. In addition to the huge economic success of the open-source model, the Alliance will be a tremendous tool used by both industry and academia. More importantly it will ensure a much-needed communication mechanism between the two in order to bring academia closer to complex real-world systems which are controlled by major industrial players in the wireless industry. In the context of the evolutionary path towards 5G, there is clearly the need for open-source tools to ensure a common R&D and prototyping framework for rapid proof-of-concept designs.

58.2

- http://openairinterface.eurecom.fr
- OpenAirInterface Wiki
- Software Installation/Build Support
- Mailing List Overview
- OAI Mailing Lists

59 srsUE

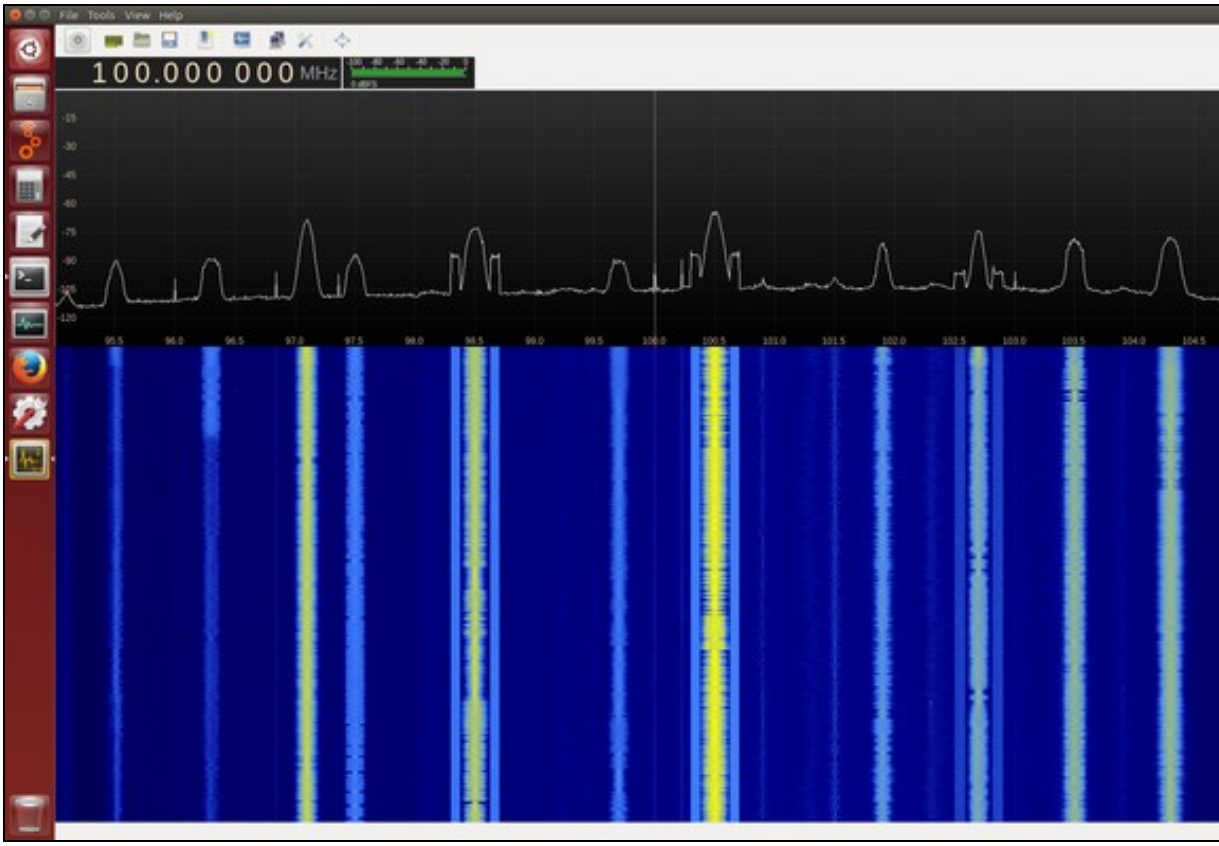
59.1

srsLTE is a free and open-source LTE library for SDR UE and eNodeB developed by SRS (www.softwareradiosystems.com). The library is highly modular with minimum inter-module or external dependencies. It is entirely written in C and, if available in the system, uses the acceleration library VOLK distributed in GNURadio.

59.2

- Software Radio Systems
- srsLTE Github

60 Gqrx

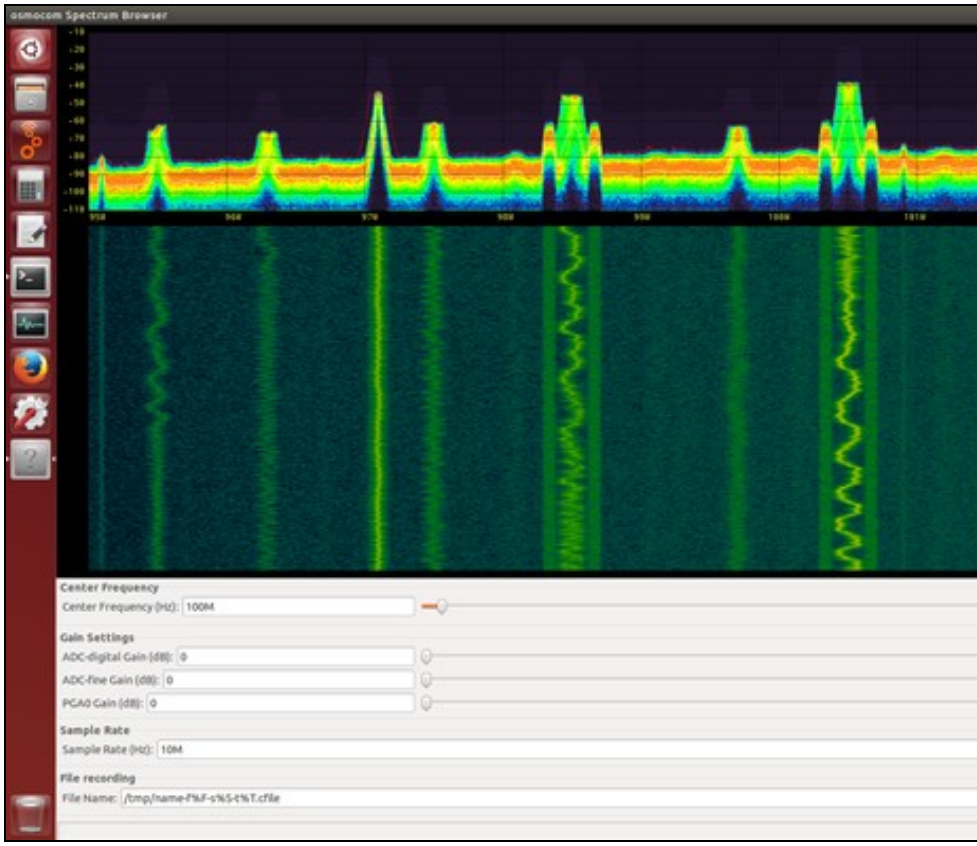


- Free open-source SDR receiver built on GNU Radio and Qt
- AM, SSB, FM-N and FM-W (mono and stereo) demodulators
- Real-time FFT plot and waterfall
- Record and playback IQ file
- Record and playback audio to/from WAV file
- Basic remote control through TCP socket connection
- Created by Alexandru Csete in Denmark

Site: http://gqrx.dk/

Github: https://github.com/csete/gqrx

61 Fosphor



- Open-source, GPU-accelerated FFT and Waterfall display tool
- GNU Radio block for RTSA-like spectrum visualization
- Uses OpenCL and OpenGL for acceleration
- High frame rate, great for visualizing fast-changing signals
- Created by Sylvain Munaut, @tnt, https://github.com/smunaut

Wiki page:
https://projects.osmocom.org/projects/sdr/wiki/Fosphor

Github mirror: https://github.com/osmocom/gr-fosphor

Video demo:
https://www.youtube.com/watch?v=mjD-l3GAghU

62 Multichannel RF Reference Architecture

62.1

AN-822 by Dylan Baros, Michael Dickens, Joseph Paller, Neel Pandeya, and Jovian Wysocki

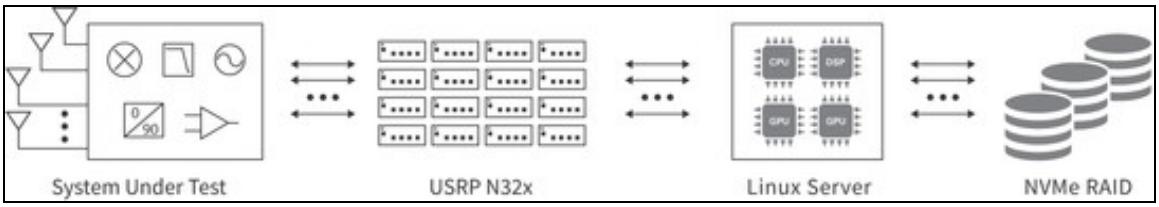
This document provides guidance for designing a system that uses the NI Multichannel RF Reference Architecture version 1.2.

This document describes how to design a system that enables synchronization between Receive (Rx) channels, Transmit (Tx) channels, and Transmit-Receive (Tx-Rx) channels, as well as high-throughput data sample transport from the software-defined radio (SDR) to a server system over 10 GbE data links. This document provides a starting point in the creation of your specific system configuration. A basic understanding of Linux, command line, C++, Git, and networking is required to build and develop on the NI Multichannel RF Reference Architecture system.

63

The Multichannel RF Reference Architecture allows engineers to quickly scale from software simulation to hardware demonstration. It features direct conversion RF sampling (200 MHz instantaneous bandwidth) with NI USRPs for multiple RF applications, enabling coherent operation through shared local oscillator (LO), PPS, and 10 MHz reference clock signals.

Figure 1. System Overview



The architecture features the following components:

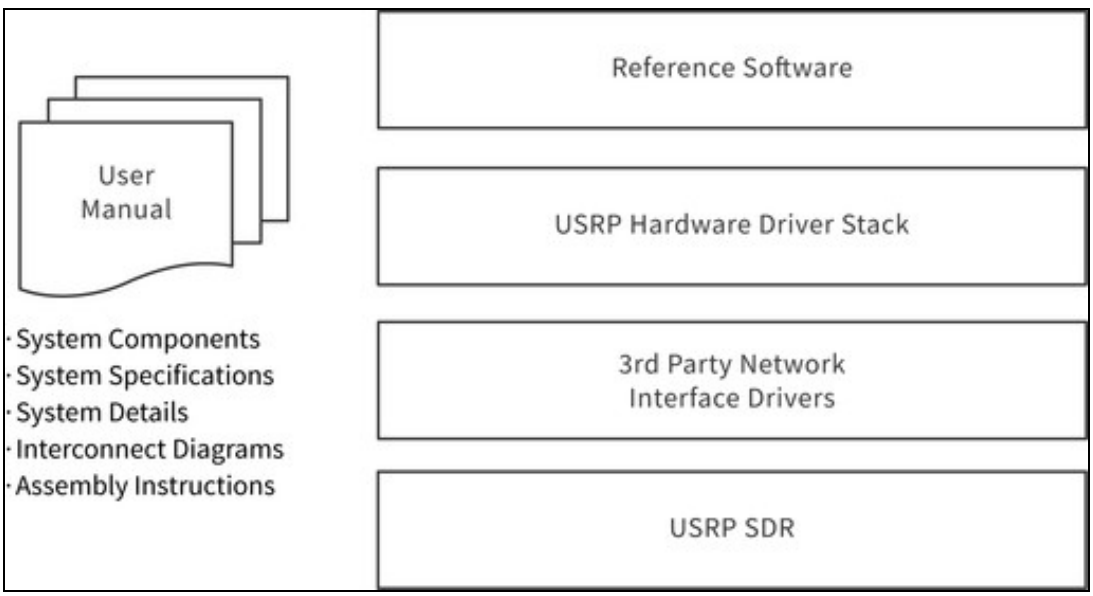
- Document, bill of materials, and interconnect diagrams that describe how to set up a system with up to 32 channels of Rx-Rx, Tx-Rx, and Tx-Tx synchronization.
- Reference library that contains functions to configure and control the USRP N321s and USRP N320s to maintain phase coherence between multiple channels.
- Reference library that contains functions to configure and control the USRP N321s and USRP N320s for high-rate data transfer from multiple channels to memory or storage.
- C++ examples that demonstrate how to use reference library functions to configure and control a multichannel system.
- Documentation that lists installation and configuration information for a 32x32 system, as well as best practices for development and validation of the system. *N*x*M* refers to systems, where *N* is the number of transmit channels and *M* is the number of receive channels.
- Documentation that lists performance benchmarks with validated results for systems with up to 32 channels for the following specifications:

  o Synchronization between 32 Rx-Rx channels, 16 Tx-Tx channels, and 32 Tx-Rx channels
  o Bidirectional streaming of IQ data between 32 channels and server

- Bill of materials for systems with up to 32 channels.

The reference architecture demonstrates using RFNoC to transmit from both the replay block as well as the host.

Figure 2. Reference Architecture Overview

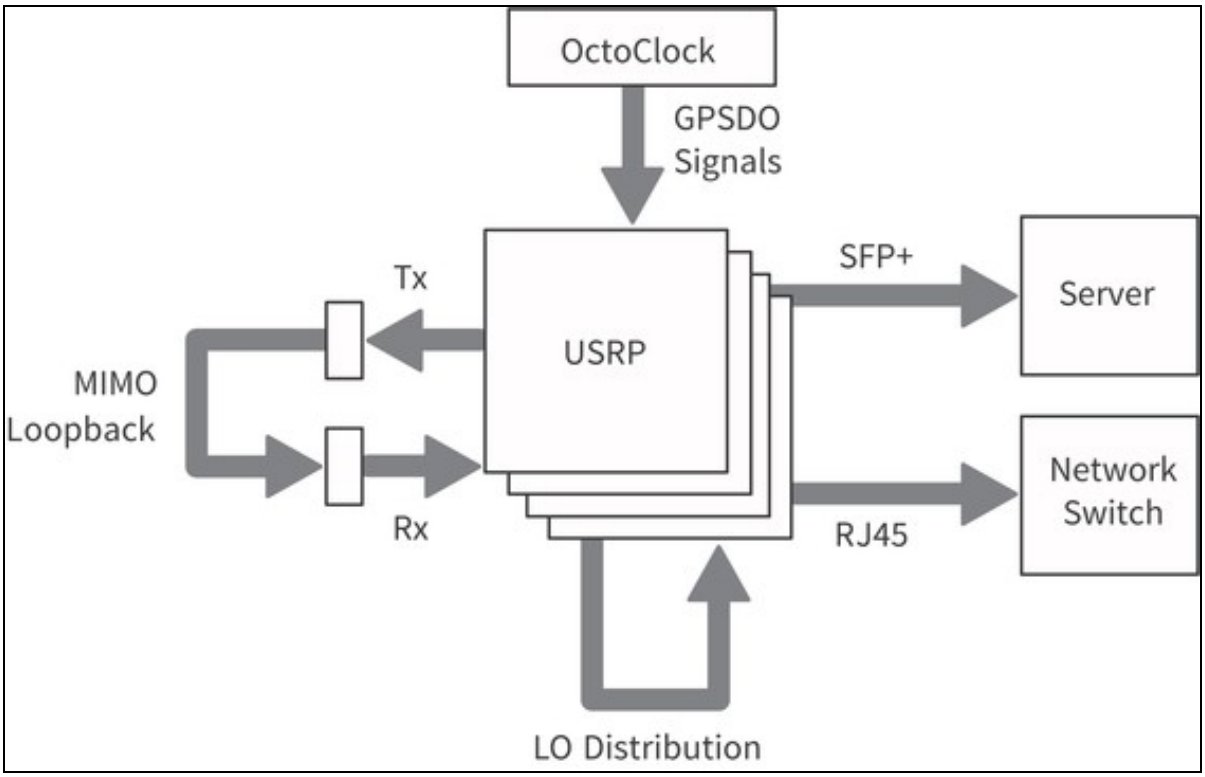


63.1

The reference architecture uses USRP N321s and USRP N320s that scale from 2 to 32 phase-coherent transmit and receive channels.

The following diagram provides an overview of the topography of the connections and primary hardware components.

Figure 3. Hardware Overview



For a complete list of required hardware for the Multichannel RF Reference Architecture, refer to the bill of materials in the *<REFARCH>*/docs/ folder, where *<REFARCH>* is an alias for the following GitHub file folder location:
https://github.com/EttusResearch/refarch-multich

63.1.1

This section describes the NI/Ettus Research products required for generating and acquiring RF signals as well as the synchronization signals?PPS, 10 MHz, and Local Oscillator (LO)?shown in Figure 3:

- **USRP N321**?Networked software-defined radio with 2-channel RF front end and 250 MSample/s on both Tx and Rx. Features a power splitter for LO distribution, 10 MHz shared reference clock for coherent MIMO functionality, shared PPS for enabling shared triggering, dual 10 GbE ports for data transfer, and an RJ45 management port for device management.
- **USRP N320**?Networked software-defined radio with 2-channel RF front end and 250 MSample/s on both Tx and Rx. Features LO sharing and 10 MHz shared reference clock for coherent MIMO functionality, shared PPS for enabling shared triggering, dual 10 GbE ports for data transfer, and an RJ45 management port for device management.
- **CDA-2990 with GPSDO**?Clock distribution device called an OctoClock that generates and distributes the GPSDO signals?10 MHz reference (REF) and 1 Hz (PPS)?and features a management port.
- **NI RF Torque Screwdriver and SMA Driver Bit** (NI part number 780895-01)?A torque screwdriver is *strongly* recommended to securely tighten all connections to/from the USRPs in the system. You can also order the RF SMA Driver Bit Only (NI part number 780894-01).

63.1.2

Testing and deploying a system requires hardware from both NI and third-party vendors. The following component options have been validated and are recommended for most Multichannel RF Reference Architecture systems:

- Server

  o **Supermicro 4124GS-TNR**?Server with dual AMD EPYC processors and 10 PCIe 4 x16 slots. Numerous PCIe lanes from the CPU support many 10 GbE connections and the PCIe processing required for high-channel-count systems.
  o **Trenton Systems 3U BAM**?Server with Intel dual 3rd generation Xeon Ice Lake processors, 11 Gen 4.0 PCIe slots (5 x16 slots, 6 x8 slots), and 24x DDR4-3200 ECC RDIMM slots for high-channel-count systems.

- Network Interface Cards (NIC)

  o **Intel X710-DA4**?Network card that supports x4 10 GbE connections.
  o **Intel E810-CQDA2**?Network card that supports two 100 GbE connections, or, with a breakout cable, x8 10 GbE connections.
  o **NVIDIA MCX4121A-ACAT ConnectX-4 Lx EN**?Network card that supports two 25 GbE connections.

- Storage

  o **HighPoint SSD7505**?PCIe carrier board that holds x4 M.2 drives and supports either a software or hardware RAID configuration.
  o **Sabrent 2TB Rocket 4 PLUS**?NVME M.2 SSD drive that supports consecutive writes rates of over 6 GB/s.

- Power Splitter/Combiners

  o **Mini-Circuits 8-Way ZN8PD1-63W-S+**?DC pass, 8-way -0°, 50 ?, 500 MHz to 6,000 MHz power splitter/combiner.
  o **Mini-Circuits 4-Way ZN4PD1-63HP-S+**?High power, DC pass, 4-way -0°, 50 ?, 30 W, 500 MHz to 6,000 MHz power splitter/combiner.

- Cabling

  o **SMA Cable, Loop Back Cable Kit** (NI part number 782781-01)?Loopback cable kit includes 2 SMA-M to SMA-M cables (60 cm/2 ft) and 2 SMA-F to SMA-M Attenuators (30 dB, 50 Ohm, DC-6 GHz).

o **SFP+ Cable, 10 Gigabit Ethernet Cable w/ SFP+ Terminations (1 Meter)** (NI part number 783343-01)?SFP+ cable recommended for USRPs with 10 GigE interfaces. It can be used to connect the USRP to a 10 GigE switch or network adapter.

- Rack Accessories

o **Tripp Lite Rackmount Rack Extension Bracket** (B018-000-1P5)?Bracket required to mount the USRPs in a rack.
o **Z-Bracket**?Bracket required to mount the USRPs in a rack to meet environmental specifications.

63.2

The reference software, shown in Figure 2, consists of a C++ reference library and examples, using USRP Hardware Driver (UHD) version 4.2. The reference software is supported on Ubuntu Linux 20.04. It demonstrates how to assemble multi-USRP systems through the UHD RFNoC API and replay block. It contains a Python-based binary int16 data file (.dat) generator and viewer. The reference software is written to be scalable for systems 2x2 to 32x32 and beyond, as well as Tx or Rx only (2 to 32 channels and beyond).

The software provides examples that stream data from the replay block as well as the host, as shown in the following figures.

Figure 4. Replay Block Data Movement

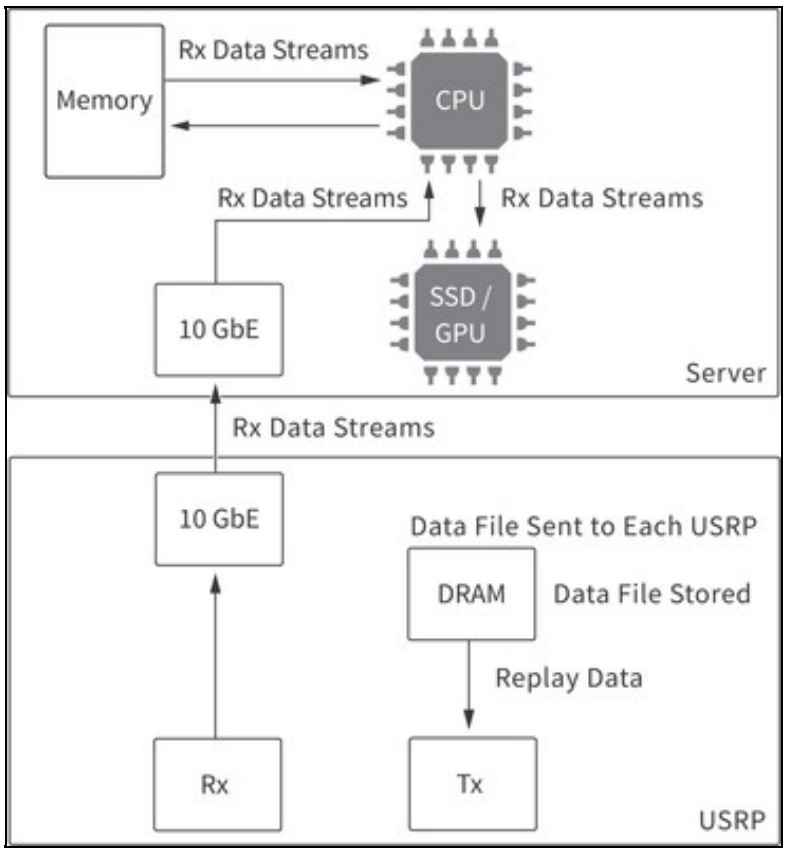
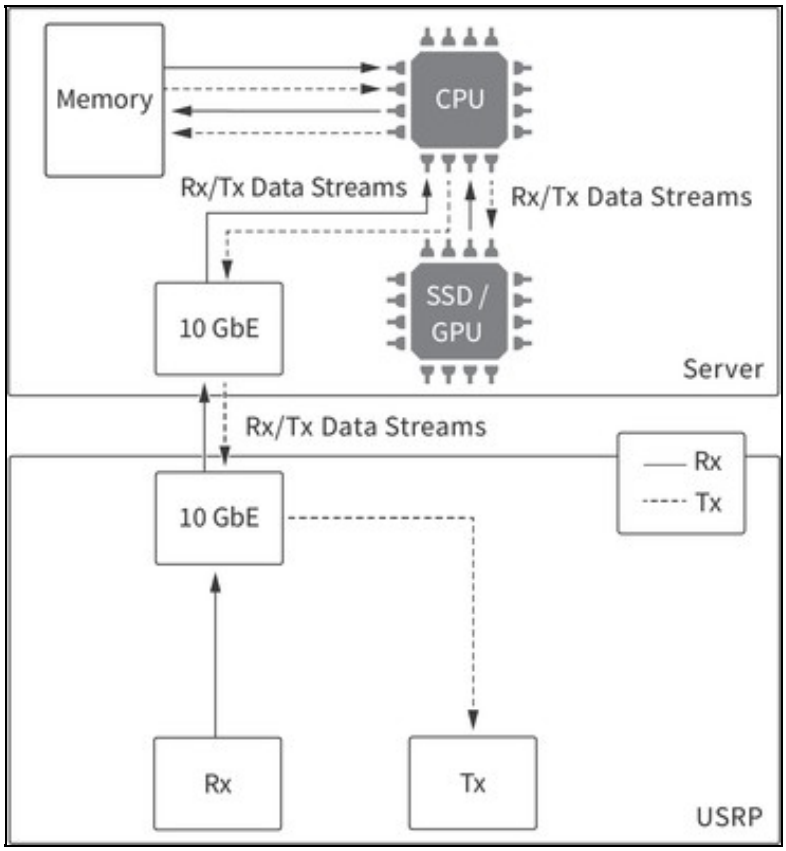


Figure 5. Stream from Host Data Movement


Memory
CPU
Rx/Tx Data Streams
Rx/Tx Data Streams
10 GbE
SSD /
GPU
Server
Rx/Tx Data Streams
Rx
Tx
10 GbE
Rx
Tx
USRP

64

The following figures show a 32x32 system layout. NI recommends using a rack for this system to simplify connections between the different hardware components.

Figure 6. Multichannel RF Reference Architecture System

Figure 7. Rear View of System with Cabling

Figure 8. Rack Layout for 32x32 System

32	Basic PDU		32
31	Network Switch		31
30	Server		30
29			29
28			28
27			27
26	KVM		26
25		USRP N321 1	25
24	USRP N320 0		24
23		USRP N321 3	23
22	USRP N320 2		22
21	OctoClock CDA-2990 Unit 1		21
20	Shelf for Cables Unit 0 8-way Splitter Unit 0 8-way Splitter Unit 2	 8-way Splitter Unit 1 8-way Splitter Unit 3	20
19		USRP N321 5	19
18	USRP N320 4		18
17		USRP N321 7	17
16	USRP N320 6		16
15	OctoClock-G CDA-2990 Unit 0		15
14	Shelf for Cables Unit 1 4-way Splitter Unit 0	 4-way Splitter Unit 1	14
13		USRP N321 9	13
12	USRP N320 8		12
11		USRP N320 11	11
10	USRP N320 10		10
9	OctoClock CDA-2990 Unit 2		9
8	Shelf for Cables Unit 2 8-way Splitter Unit 4 8-way Splitter Unit 6	 8-way Splitter Unit 5 8-way Splitter Unit 7	8
7		USRP N321 13	7
6	USRP N320 12		6
5		USRP N320 15	5
4	USRP N320 14		4
3			3
2			2
1	Switched PDU		1

64.1

Adhere to the following best practices when installing, setting up, and connecting components to your system:

- Number and label the USRPs with serial numbers on the front of the devices. Visit *<REFARCH>/docs/32_Channel_Template_layout_BOM_and_wiring.ods* for a customizable template.
- While running the system in loopback, attach 30 dB attenuators to all output channels to prevent damage to the USRPs.
- Arrange the equipment in the rack to optimize proper airflow. USRP N321s and USRP N320s facilitate airflow from right to left, and the server facilitates airflow from front to back. NI recommends using Z-brackets to vertically stagger installation of USRPs.
- Position the lowest USRP in the rack a few units up from the floor to allow for adequate device connector accessibility.
- NI recommends positioning the USRPs halfway to 2/3 toward the rear of the rack to allow easier bundling of cables.
- Label the NICs and 10 GbE cables to identify each USRP connection, as described in *Connecting the SFP+ Ports*.
- Connect the RX LO OUT and TX LO OUT cabling from Distribution Tier of USRP N321s to the nearest USRP N320 devices, as described in *Connecting the LO Distribution*
- Keep the end-to-end path of each Rx, Tx, LO, PPS, and 10 MHz signal identical in length, using the shortest cable possible per tier, as described in *Cabling the Hardware*.
- Use a calibrated torque wrench to connect all SMA connections to industry standard torque measurements.
- Periodically check all cabling to ensure a secure connection.

64.2

The following topics describe the connections for 8x8, 16x16, and 32x32 systems.

> **Note** Ensure that you use a calibrated torque wrench for all cabling connections. NI recommends the RF Torque Screwdriver and SMA Driver Bit (NI part number 780895-01).

64.2.1

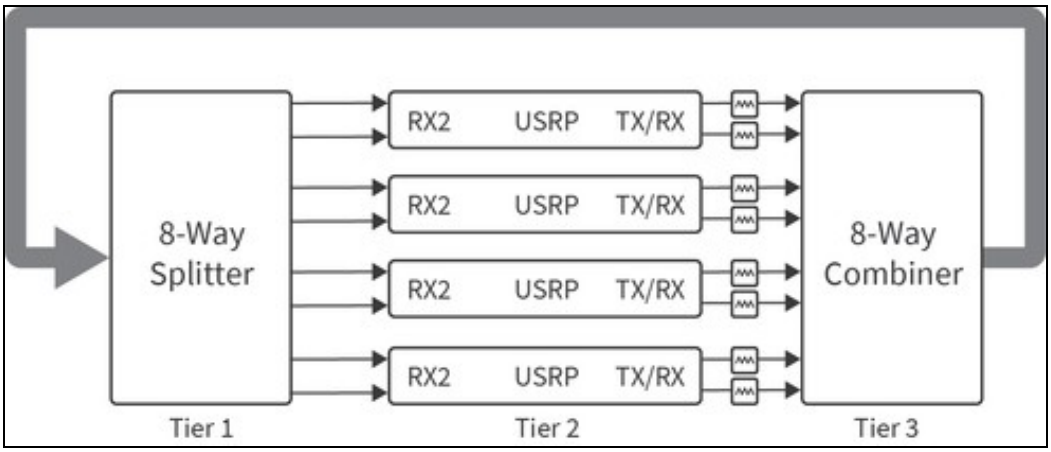
You can use the MIMO loopback to validate the setup of the system and provide a baseline for synchronization calibration. Create the MIMO loopback by connecting the USRP TX/RX connectors (used for transmitting signals) to a combiner, connecting the combiner to a splitter, and connecting the splitter to the USRP RX2 connectors (used for receiving signals).

> **Notice** 30 dB attenuators must be used on every TX to ensure safe use of the MIMO loopback. Using the system without attenuators will overload the RX and will permanently damage the USRPs.

> **Note** NI recommends keeping all cables in the same tier the same length to reduce skew.

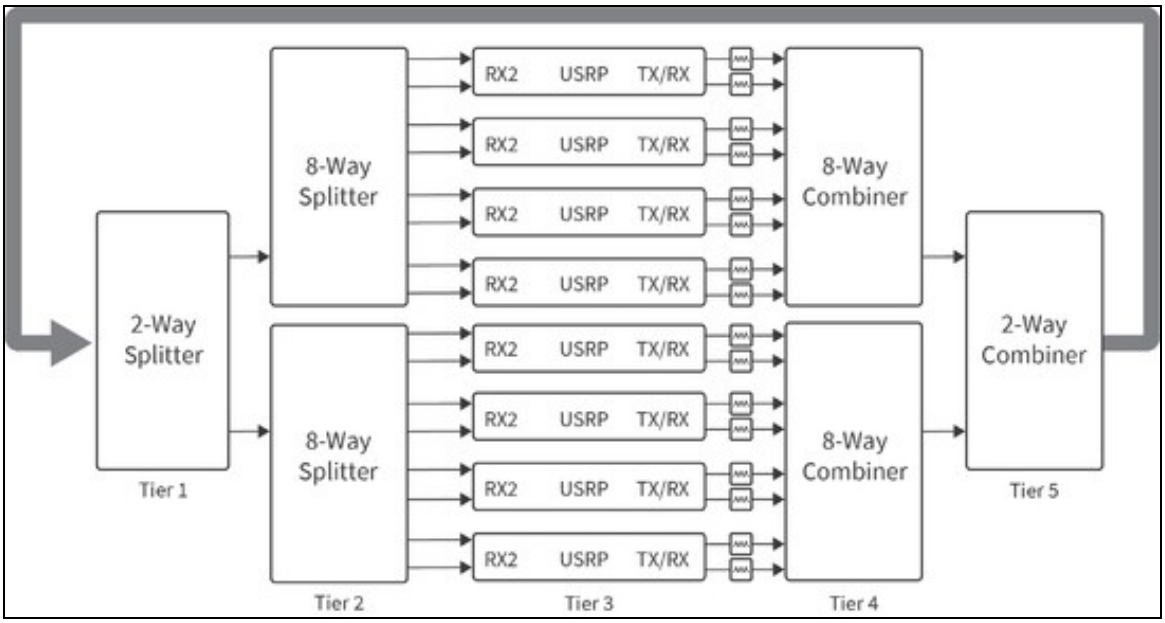
The 8x8 system requires one 8-way splitter and one 8-way combiner. To set up the MIMO loopback, connect the RF 0/1 TX/RX and RX2 connectors on the USRP to an 8-way splitter and an 8-way combiner, as shown in the following figure. All cables connecting between tiers must be the same length and type.

Figure 9. MIMO Loopback (8x8)



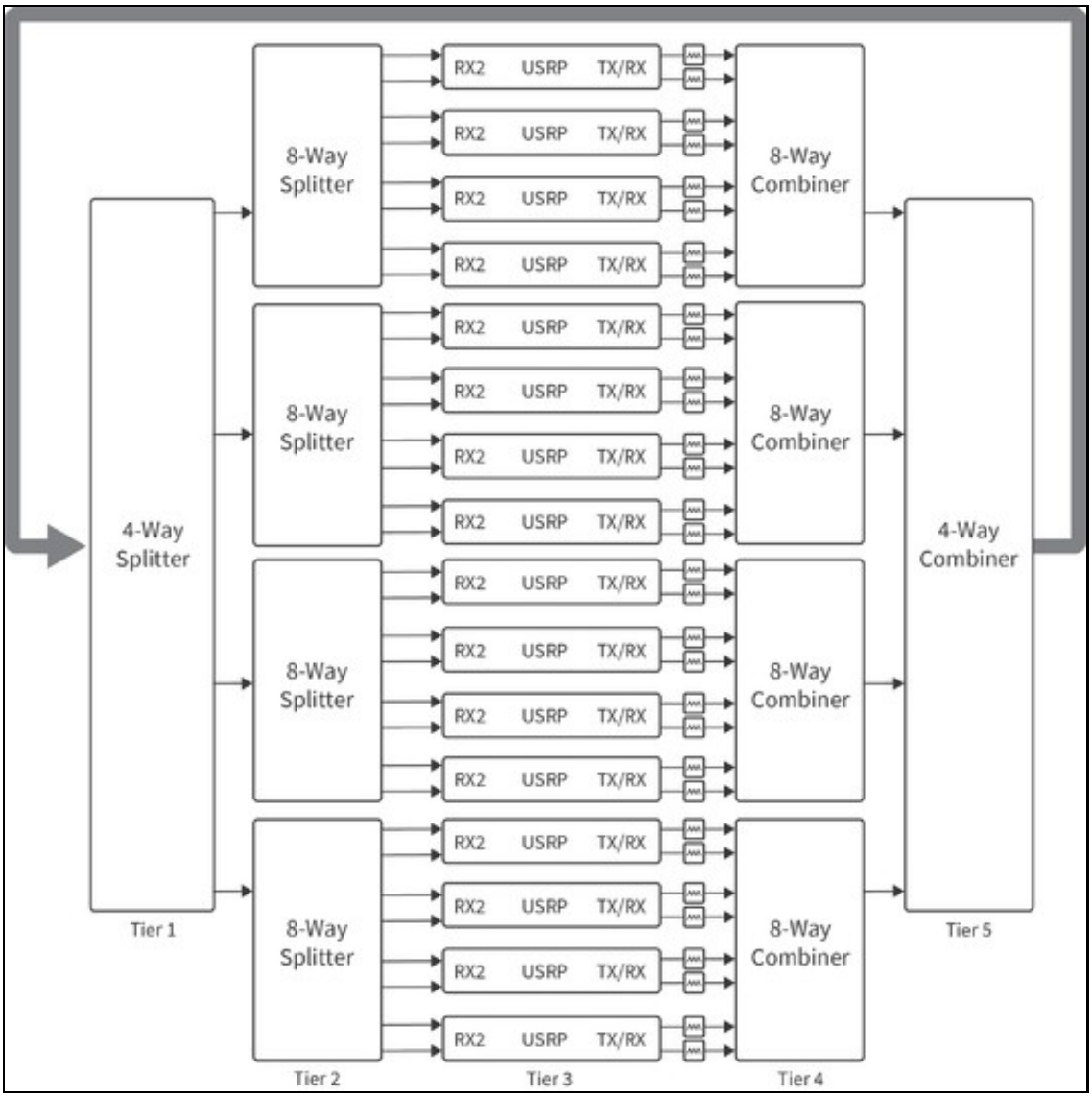
The 16x16 system requires two 8-way splitters, one 2-way splitter, two 8-way combiners, and one 2-way combiner. To set up the MIMO loopback, connect the RF 0/1 TX/RX and RX2 connectors on the USRP to an 8-way combiner and an 8-way splitter. Then connect the two 8-way combiners to a 2-way combiner and the two 8-way splitters to a 2-way splitter. All cables connecting between tiers, shown in the following figure, must be the same length and type.

Figure 10. MIMO Loopback (16x16)



The 32x32 system requires four 8-way splitters, one 4-way splitter, four 8-way combiners, and one 4-way combiner. To set up the MIMO loopback, connect the RF 0/1 TX/RX and RX2 connectors on the USRP to an 8-way combiner and an 8-way splitter. Then connect the four 8-way combiners to a 4-way combiner and the four 8-way splitters to a 4-way splitter. All cables connecting between tiers, shown in the following figure, must be the same length and type.

Figure 11. MIMO Loopback (32x32)

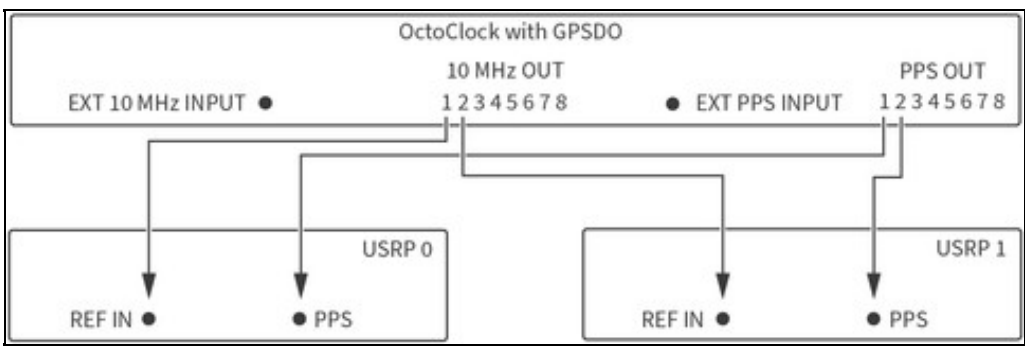


64.2.2

Connect a 10 MHz OUT connector on the OctoClock to the REF IN connector on the USRP. Connect a PPS OUT connector on the OctoClock to the PPS connector on the USRP.

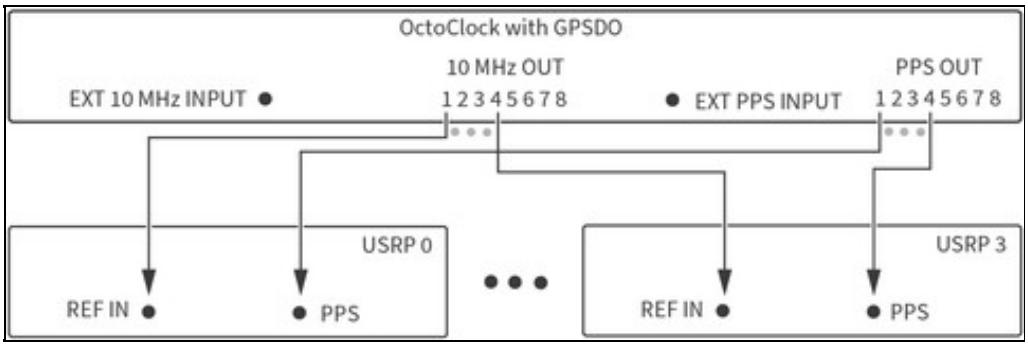
The 4x4 system requires one OctoClock with GPSDO, two USRPs, and uses two 10 MHz and two PPS OUT SMA connectors. Ensure that the source selection switch is set to internal on OctoClock 0 prior to power up.

Figure 12. 10 MHz and PPS Connections (4x4)



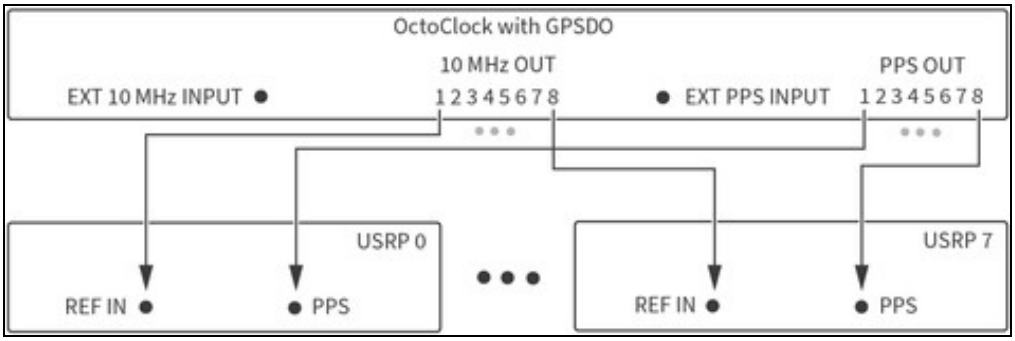
The 8x8 system requires one OctoClock with GPSDO, four USRPs, and uses four 10 MHz and four PPS OUT SMA connectors. Ensure that the source selection switch is set to internal on OctoClock 0 prior to power up.

Figure 13. 10 MHz and PPS Connections (8x8)



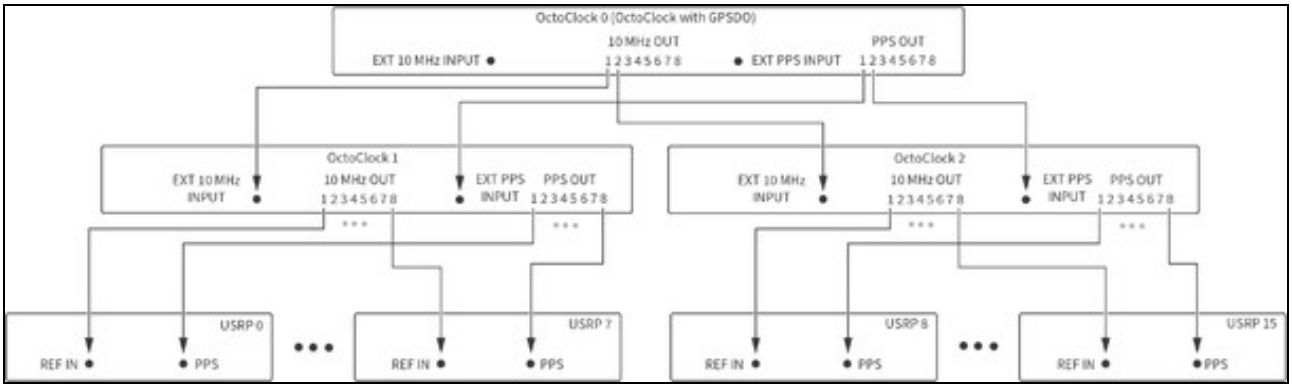
The 16x16 system setup requires one OctoClock with GPSDO, eight USRPs, and uses eight 10 MHz and eight PPS OUT SMA connectors. Ensure that the source selection switch is set to internal on OctoClock 0 prior to power up.

Figure 14. 10 MHz and PPS Connections (16x16)



Scaling to a 32x32 system requires one OctoClock with GPSDO (OctoClock 0, shown in the following figure), two additional OctoClocks (OctoClocks 1 and 2), and 16 USRPs. The source 10 MHz and PPS signals originate from OctoClock 0, which drives the OctoClock 1 and 2 inputs. OctoClock 1 and 2 each distribute all eight output 10 MHz and PPS signals to USRPs, 16 in aggregate. All cables connected from OctoClock 0 must be the same length. All cables connected to the USRPs must be the same length. Ensure that the source selection switch is set to internal on OctoClock 0, and that the source selection switch is set to external on OctoClocks 1 and 2. The selection switches must be set prior to power up.

Figure 15. 10 MHz and PPS Connections (32x32)



64.2.3

Connect the TX LO OUT, RX LO IN, and TX LO IN connectors on the USRPs for LO distribution.

> **Note** This configuration is used to synchronize both the Rx and Tx to the same LO resulting in the same center frequencies for both Tx and Rx. To synchronize different frequencies, connect RX LO OUT to RX LO IN and TX LO OUT to TX LO IN. Refer to *USRP N320/N321 LO Distribution* for connection diagrams.

While connecting the LO distribution, adhere to the following best practices:

- Tx and Rx must share the same LO connection. The LO sharing configuration for USRP N321s and USRP N320s in this system differs slightly from the LO sharing diagram on the Ettus Knowledge Base[1]. Distributing the LO from one channel to all other channels results in phase coherence across all channels, and there will be a constant non-zero phase offset between any two channels. This phase offset can be empirically measured and compensated for in software, thus resulting in all channels being phase aligned. The overall system is phase synchronous under a wide range of conditions; refer to *Synchronization* for more information about measured synchronization performance.
- Configure all USRPs, including the LO Source USRP, to use external LO inputs. *Refer to* `sync.cpp` in the *<REFARCH>*/lib/ folder for information about how the reference software configures the LOs on the USRPs.
- Use tiered LO signal distribution and identical cable lengths between any specific tiers.
- Connect the RX LO OUT and TX LO OUT cabling from Distribution Tier of USRP N321s to the nearest USRP N320s, as shown in the following figures.
- Use minimum identical cable length between tiers.

  The following table lists the tiers for all system configurations.

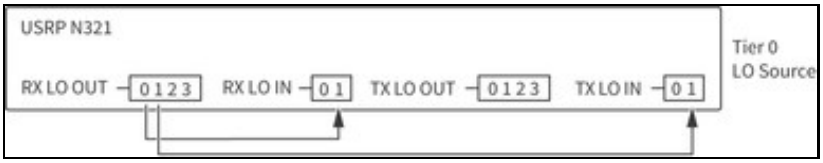
Table 1. LO Source and Tiers

System	Tier(s)	USRP N321 Minimum Quantity	USRP N320 Maximum Quantity
2x2	LO Source (0)	1	?
4x4	LO Source (0), 1	1	1
8x8	LO Source (0), 1, 2	2	2
16x16	LO Source (0), 1, 2	3	5
32x32	LO Source (0), 1, 2, 3	6	10

The following diagrams represent LO distribution signal connections.

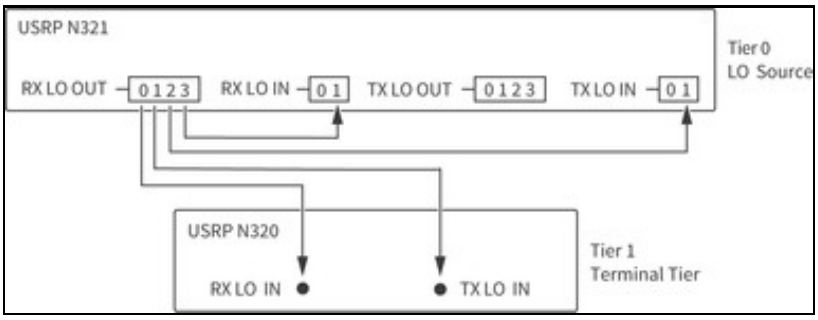
The 2x2 system requires two cables of identical length. Note that you cannot combine LO signals. NI recommends that all LO distribution cables originating from the same tier are the same length.

Figure 16. LO Sharing (2x2)



The 4x4 system requires four cables of identical length. Note that you cannot combine LO signals. NI recommends that all LO distribution cables originating from the same tier are the same length.

Figure 17. LO Sharing (4x4)



For the 8x8 system, both RX LO OUT cables originating from the LO Source USRP N321 must be the same length. All RX LO OUT and TX LO OUT cables originating from the Tier 1 USRP N321 must be the same length. Note that you cannot combine LO signals.

Figure 18. LO Sharing (8x8)

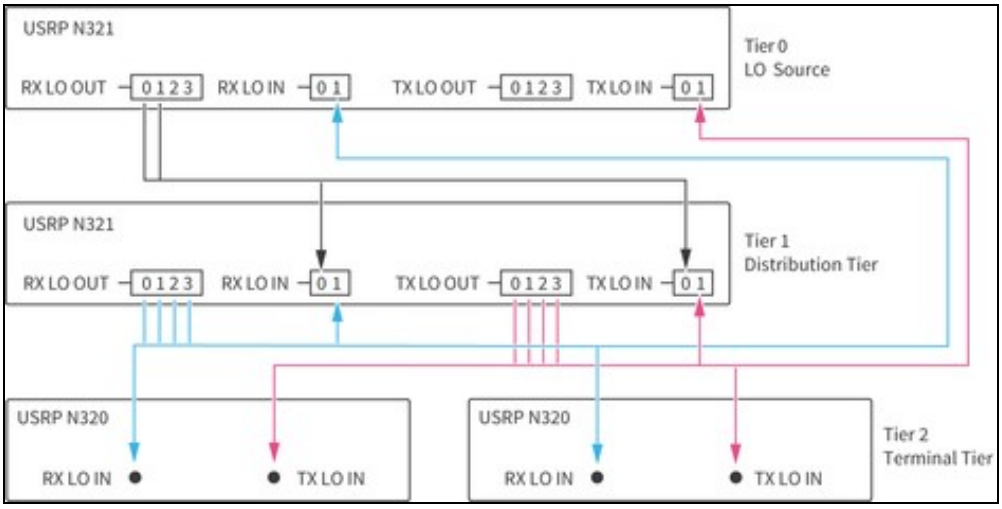


Table 2. LO Sharing Cabling (8x8)

Cable Color in Diagram	Cable Tier	Number of Cables in Tier
	LO Source Tier 0	2
	Distribution Tier 1, RX LO OUT	4
	Distribution Tier 1, TX LO OUT	4

For the 16x16 system, all RX LO OUT cables originating from the LO Source USRP N321 must be the same length. All RX LO OUT and TX LO OUT cables originating from Tier 1 USRP N321s must be the same length. Note that you cannot combine LO signals.

Figure 19. LO Sharing (16x16)

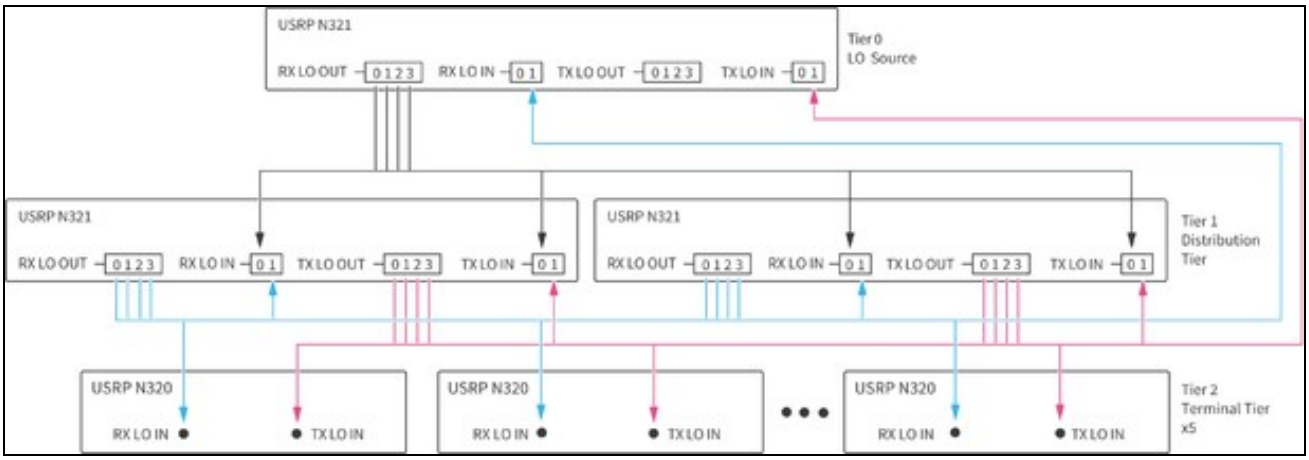


Table 3. LO Sharing Cabling (16x16)

Cable Color in Diagram	Cable Tier	Number of Cables in Tier
	LO Source Tier 0	4
	Distribution Tier, RX LO OUT	8
	Distribution Tier, TX LO OUT	8

For the 32x32 system, both RX LO OUT cables originating from the LO Source USRP N321 must be the same length. All RX LO OUT and TX LO OUT cables originating from the Tier 1 USRP N321 must be the same length. All RX LO OUT and TX LO OUT cables originating from the Tier 2 USRP N321s must be the same length. Note that you cannot combine LO signals.

Figure 20. LO Sharing (32x32)

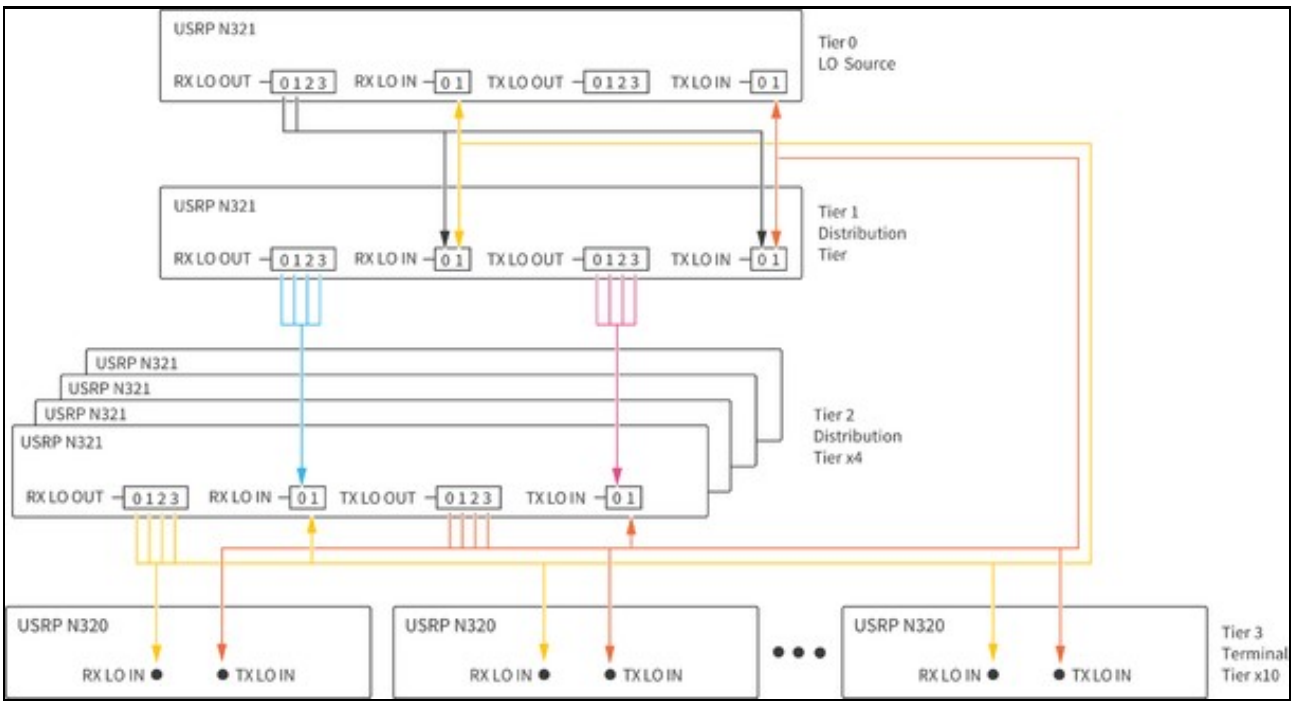


Table 4. LO Sharing Cabling (32x32)

Cable Color in Diagram	Cable Tier	Number of Cables in System
	LO Source Tier 0	2
	Distribution Tier 1, RX LO OUT	4
	Distribution Tier 1, TX LO OUT	4
	Distribution Tier 2, RX LO OUT	16
	Distribution Tier 2, TX LO OUT	16

64.2.4

Make the following RJ45 connections to allow USRP file system configuration and updates later in the process.

> **Note** The network port on the USRPs in this system use DHCP by default. You must either have a DHCP server running on an upstream networked device or set a static IP address for each USRP.

1. Connect an RJ45 cable between the server and the switch.
2. Connect an RJ45 cable between the Management Port on each USRP and a port on one of the switches.
3. (Optional) Connect a cable between the network switch and external internet for network management.

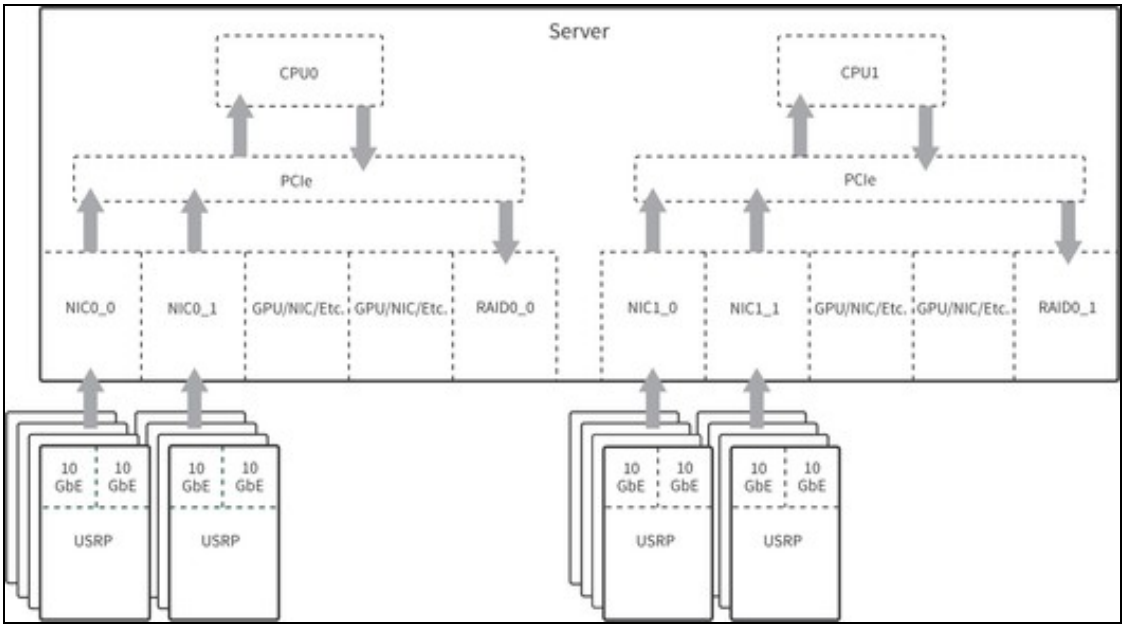
> **Note** NI recommends that you create a document for tracking USRP serial numbers with assigned network address. Visit the *<REFARCH>*/docs/ folder for a customizable template.

64.2.5

The server contains multiple network interface cards (NIC), each supporting multiple SFP+ ports.

Each USRP has two SFP+ ports, SFP+ 0 and SPF+ 1; however, the reference architecture assumes the use of SFP+ 0.

Figure 21. Overview of Dataflow between the USRPs and the Server



Connect an SFP+ cable between the SFP+ 0 port on each USRP and a port on one of the NICs in the server.

Assign each USRP to a particular network address. NI recommends that you organize your network in the following ways:

- Label the NICs and 10 GbE cables to identify each USRP connection.
- Create a document for tracking USRP serial numbers with assigned network address. Visit the *<REFARCH>*/docs/ folder for a customizable template.

> **Note** For maximum streaming rates with DPDK, both SFP+0 and SFP+1 must be configured and connected. Dual port streaming can also be used without DPDK for enhanced streaming capability.

64.2.6

Plug the USRPs, OctoClocks, server, and switch into the system PDU.

64.2.7

If you want to improve streaming performance for your system, you can install Data Plane Development Kit (DPDK) 20.11. Streaming rates to disk are limited by the write speed of the chosen storage medium. Refer to *Streaming with DPDK Simultaneously to/from Disk Rates* for information on the streaming rates.

> **Note** This reference architecture has been tested using DPDK 20.11 with NVIDIA MCX4121A-ACAT ConnectX-4 Lx EN Adapter Cards. The following steps may only apply to the NVIDIA ConnectX-4.

1. Ensure that you install the proper drivers for your NIC prior to installing DPDK and UHD.
2. Install and configure the DPDK software according to the *Data Plane Development Kit Getting Started Guide for Linux.*
3. Configure huge pages by opening the grub configuration file,
   `/etc/default/grub`.
4. Add the following parameters to `GRUB_CMDLINE_LINUX_DEFAULT`:
   ```
   GRUB_CMDLINE_LINUX_DEFAULT="quiet splash
   default_hugepagesz=2048M hugepagesz=2048M hugepages=10000"
   ```
5. Close `/etc/default/grub`.
6. At the command prompt, update your grub configuration with the following command:
   ```
   sudo update-grub
   ```
7. Reboot the host machine for these settings to take effect.
8. Create a UHD configuration file to interface with DPDK.
   1. Run the following command to list the MAC addresses for your NICs:
      ```
      ip a
      ```

2. Use the following commands to create the UHD configuration file at the location `/etc/uhd/`:

```
sudo su
mkdir -p /etc/uhd
nano /etc/uhd/uhd.conf
```

9. Update the following fields of your UHD configuration file.
   - o Update the `dpdk_mac` field to match your NIC MAC address variables
   - o Update the `dpdk_driver` field if the location is different on your system.
   - o Update the `dpdk_corelist` and `dpdk_lcore` fields. In this example, a two-port NIC is used. There should be one core for the main DPDK thread (in this example, core #2), and separate cores assigned to each NIC (in this example, core #3 for the first port on the NIC, core #4 for the second port on the NIC).
   - o Update the `dpdk_ipv4` fields to your IP range.

Refer to the following `uhd.conf` file for an example of how you should update the fields in your configuration file.

```
[use_dpdk=1]
dpdk_mtu=9000
dpdk_driver=/usr/local/lib/dpdk-pmds
dpdk_corelist=2,3,4
dpdk_num_mbufs=8192
dpdk_mbuf_cache_size=64

[dpdk_mac=50:6b:4b:3b:3c:d8]
dpdk_lcore = 3
dpdk_ipv4 = 192.168.50.1/24
dpdk_num_desc = 4096

[dpdk_mac=50:6b:4b:3b:3c:d9]
dpdk_lcore = 4
dpdk_ipv4 = 192.168.51.1/24
dpdk_num_desc = 4096
```

Refer to *Running the Examples* to verify successful installation of DPDK on your system.

Note Refer to *Getting Started with DPDK and UHD* for more information about DPDK and UHD.

64.3

Complete the following steps to install the software and necessary dependencies on your host machine.

1. Navigate to the GitHub repository, located at https://github.com/EttusResearch/refarch-multich, to get the reference software.
2. Clone the repository to your computer using standard GitHub commands or download the repository directly.
3. In a terminal, navigate to the top-level *refarch-multich* directory and issue the following command to make the setup script executable:
   `chmod +x setup_script.sh`
4. Run the script with the following command:
   `sudo ./setup_script.sh`

The `refarch-multich` repository contains the following folders:

- `config`?Contains script files that assist with the software installation process.
- `docs`?Contains system bill of materials and reference architecture template.
- `examples`?Contains reference architecture examples and configuration files.
- `lib`?Contains the library source files.
- `tools`?Contains Python-based data file generator and viewer.

64.4

The following procedure references the *USRP N300/N310/N320/N321 Getting Started Guide*. Follow the https://kb.ettus.com/USRP_N300/N310/N320/N321_Getting_Started_Guide procedure with the specific steps below.

Note When you run the scripts in this section, *all* USRPs connected to the host are updated. Run the `uhd_find_devices` example for a list of USRPs that are connected to the host.

1. Follow the *Connecting to the ARM via SSH* procedure in *Connecting the Device*. You do not need to set up the USRPs for a serial console connection.
2. Follow the procedure in *Updating the Linux File System*.

   Note If the procedure in *Updating the file system with Mender* fails, you must follow the process in *Updating the files system by writing the disk image*. For a script that updates all USRPs connected to the system, refer to `<REFARCH>/config/usrp_filesystem.sh` in the GitHub repository.

3. Follow the procedure in *Updating the Network Configurations*. Set the MTU to `9000`. Every USRP SFP+ ports IP address must have a unique subnet. Change the subnet by replacing *XXX* in the address, for example, `192.168.XXX.2`. If you make any changes to the network configurations, NI recommends that you restart the USRP for those changes to take effect.

   Note For maximum streaming rates with DPDK, dual port streaming must be configured. Follow the procedure in Updating the Network Configurations for configuring both SFP+0 and SFP+1 ports on the USRP. Ensure that the IP addresses are unique.

   Note NI recommends that you create a document for tracking USRP serial numbers with assigned network addresses. Visit the *<REFARCH>/*docs/ folder for a customizable template. For network addresses, NI recommends that you assign static IP addresses for the management port IPs. Refer to the procedure in *Updating the Network Configurations* to update eth0.network file on the USRP.

   For a script to update the SFP ports on all USRPs automatically, refer to *<REFARCH>/*config/usrp_ip_config.sh in the GitHub repository. The script changes the sfp1 port by default but can be modified to change the sfp0 port. You still need to manually configure the host as listed in step 5.

4. Follow the *Network Mode FPGA Image Update* procedure in *Updating the FPGA Image* to update to the XG image. Use the following command to configure both SFP ports to 10 Gb links:
   `uhd_image_loader --args "type=n3xx,addr=<N3xx_IP_ADDR>,fpga=XG"`

where `<N3xx_IP_ADDR>` is the actual IP address of the USRP being updated. You can find this address by correlating the serial number on the device with the IP address returned by the command `uhd_find_devices`.

> **Note** For a script to update the FPGA images on all USRPs connected to the system, refer to *<REFARCH>/*/config/usrp_image_loader.sh on the GitHub repository.

5. Follow the *Dual 10Gb Streaming SFP Ports 0/1* procedure in *Setting Up a Streaming Connection* to configure the host Ethernet adapters for 10 GbE connection to the USRP.

> **Note** Ubuntu Linux uses Netplan to manage IP addresses of network cards, however, other operating systems may configure the network interfaces differently. Visit the *<REFARCH>/config/* folder to find a customizable template `.yaml` file. Each host port and the USRP to which it is paired must have a matching subnet. After the `.yaml` file is configured, copy it to the `/etc/netplan` directory?overwriting `01-network-manager-all.yaml`, if necessary. Then use the following command to apply the network configuration:
> `$ sudo netplan apply`

6. Follow the procedure in *Verifying Device Operation*.
7. (Optional) Follow the procedure in *USRP N3xx Device Specific Operations* to configure USRP settings.

64.5

After setting up the first USRP, follow the procedure in *Setting Up the First USRP* for the remaining USRPs with the specific details listed below.

- For subsequent devices, you may choose to execute the following procedures in parallel:

  o *Updating the Linux File System*
  o *Updating the Network Configurations*?Each USRP port must be configured to have a different subnet.
  o Updating the FPGA Image

- Update the `.yaml` file on the host computer after configuring each remaining USRP.

64.6

Adjust certain settings to optimize the host system. Most of these settings are completed for you through the setup script. NI recommends that you manually configure the following optimizations:

- Increase the ring buffer for each interface by executing the following command in a terminal

  `sudo ethtool -G <interface> tx 4096 rx 4096`
  where `<interface>` is the network interface of the specific USRP. You can find this interface through the Network Manager or by executing the following command in a terminal `ifconfig`.

  > **Note** Increasing the ring buffer of an interface results in more memory reserved. When setting the ring buffer, make sure to monitor your memory usage. A large number of interfaces can result in hundreds of GB of memory usage. You may need to restart the host system to clear the allocated memory.

  > **Note** This command is executed once for each interface being used. For example, a system with 16 USRPs and 16 interfaces runs 16 times and results in around 65 GB of memory usage. A script has been provided to assist with these settings. The user is required to insert the names of the host interfaces. Refer to `nicrb.sh` in the *<REFARCH>/*config/ folder for more information. This script may need to be run after each host reboot. You can reference the interface names in the `.yaml` file configured in *Setting Up the First USRP Device*.

- Enable hyperthreading on the server to improve multi-threaded performance. Refer to the server BIOS settings for information.

65

65.1

Multichannel RF Reference Architecture is built using the RFNoC API that ships with the USRP Hardware Driver (UHD).

- **UHD RFNoC Example Source Code**? NI recommends that users who are new to UHD or the RFNoC API begin with the following UHD RFNoC examples, located in the `uhd/host/examples` directory:

  o `rfnoc_radio_loopback.cpp`?This example uses the RFNoC API to connect an Rx radio to a Tx radio and run a loopback.
  o `rfnoc_replay_samples_from_file.cpp`?This example uses the replay block to replay data from a file.
  o `rfnoc_rx_to_file.cpp`? This example uses the RFNoC API to demonstrate the radio receiving and writing to file.

- **Reference Architecture Example Source Code**? The following examples, located in the *<REFARCH>*/examples/ folder, demonstrate synchronized Tx-Rx operation scalable from 2x2 up to 32x32 systems:

  o `Arch_iterative_loopback.cpp`?This example cycles through each Tx channel to all Rx channels.
  o `Arch_multifreq_loopback.cpp`?This example cycles through different frequencies and calls `rfnoc_txrx_loopback.cpp`.
  o `Arch_rfnoc_txrx_loopback.cpp`?This example demonstrates a single Tx to all Rx loopback file using a multithreaded implementation.
  o `Arch_rfnoc_txrx_loopback_mem.cpp`?This example demonstrates a single Tx to all Rx loopback to memory using a multithreaded implementation.
  o `Arch_rx_to_mem.cpp`?This –example demonstrates receiving on all channels to memory.
  o `Arch_txrx_fullduplex.cpp`?This example demonstrates simultaneously transmitting to and receiving from the host.
  o `Arch_txrx_fullduplex_dpdk_mem.cpp`?This example demonstrates simultaneously transmitting to and receiving from the host memory using DPDK.
  o `Arch_txrx_fullduplex_dpdk.cpp`?This example demonstrates simultaneously transmitting to and receiving from the host storage using DPDK
  o `Arch_dynamic_tx.cpp`?This example is written for a four-channel system. It demonstrates the ability to stream different files and/or have different logic in each transmit thread. Four (potentially) different transmit threads are spawned, one to each USRP channel. Each thread can be customized as needed. You can apply this example to higher channel count systems, as well as receive threads if needed.

  Note The provided examples use the same LO and Tx frequency transmitted across all channels and devices. You must modify this reference architecture if you want to transmit different frequencies across channels.

65.2

The reference architecture examples are compiled when you run the following setup script: `setup_script.sh`.

If you need to make changes to the examples, you must recompile the code.

1. Navigate to *<REFARCH>/build/* to recompile the provided examples.
2. Execute the following commands in the terminal:

```
cmake ..
make ?j
```

If a custom example needs to be compiled, you must modify *<REFARCH>*/CMakeLists.txt to be used with `cmake`. To build a new example titled `new_example.cpp`, complete the following steps.

1. Add the following lines to *<REFARCH>*/CMakeLists.txt:

```
add_executable(new_example {relative File Location}/new_example.cpp)

message(STATUS "New Example")

target_link_libraries(new_example PRIVATE UHD_BOOST Arch_lib)
```

2. Navigate to *<REFARCH>*/build/.
3. Execute the command `make ?j` in the terminal to compile the custom examples.

65.3

The reference architecture examples use configuration files instead of input arguments to configure the application. The configuration files are located in the *<REFARCH>*/examples/ folder.

- Refer to the example `runconfig.cfg` file for a detailed explanation of each argument.
- The reference architecture uses a multiple RAID 0 system to increase the number of parallel writes and, thus, achieve higher data rates. You must configure the `rx-file-location` variable in the `runconfig.cfg` file. Refer to the configuration file for more information.

  Note Ensure that the management address for each USRP is specified in the configuration file as demonstrated in the `runconfig.cfg` example. Not specifying a management address can lead to loss of performance.

65.4

Note If dual port streaming is configured, it must be enabled in the Reference Architecture configuration file. Uncomment `second_addr#` for each USRP and replace with the correct USRP number and SFP+1 IP address as configured in *Setting Up the First USRP*.

1. If you are running a DPDK example, you must alter the configuration file. Add `use_dpdk=1` to the following line in the configuration file:
   `args = type=n3xx,master_clock_rate=250e6,use_dpdk=1`
   Note Ensure that you remove `use_dpdk=1` when running non-DPDK examples.
2. Open a terminal in the build folder.
3. Run an example using the `--cfgFile=runconfig.cfg` command. For example, run `rfnoc_txrx_loopback.cpp` by executing the following command in a terminal:
   `./rfnoc_txrx_loopback --cfgFile=../examples/runconfig.cfg`

65.5

- The replay block works as a playback buffer that uses DRAM to store samples on the USRP. Refer to the importData function in `replaycontrol.cpp` in the *<REFARCH>/lib/* folder to observe how data is sent from the host to store on the USRP DRAM.
- Sample data files should contain sc16 data samples and should be a multiple of two samples (8 bytes) in size because the replay block records and plays back in multiples of 8 bytes. Refer to samples_generator.py in the *<REFARCH>/tools/* folder for an example of a sine wave generation. Refer to the *Using the RFNoC Replay Block* Knowledge Base article for more information.

65.6

Received data files consist of interleaved int16 IQ data (IQIQI...). You can view data and helpful phase information with the python-based `readDatFile.py` example, located in the *<REFARCH>*/tools/dat_analysis folder. Refer to the comments in `readDatFile.py` for more information about the use of the program.

66

NI has validated the performance specifications, described in this section, using a 32x32 test system configured as described in *Building the System*. NI used the reference software to determine the specifications in this document.

66.1

Warranted?specifications describe the performance of the system under stated operating conditions and are covered by the system warranty. Warranted specifications account for measurement uncertainties, temperature drift, and aging. Warranted specifications are ensured by design or verified during production and calibration.

Characteristics describe values that are relevant to the use of the system under stated operating conditions, but are not covered by the model warranty.

Typical?specifications describe the performance met by a majority of systems.

Nominal?specifications describe an attribute that is based on design, conformance testing, or supplemental testing.

Measured?specifications describe the measured performance of a representative system.

Specifications are?*Measured*?unless otherwise noted.

66.2

Transmit channel count	Up to 32
Receive channel count	Up to 32
RF frequency coverage	450 MHz to 6 GHz
RF bandwidth	Up to 200 MHz instantaneous bandwidth per channel
Configuration plane	1 GbE
Data plane	10 GbE
Processing nodes	
On USRP	FPGA
On server	CPU
Data storage and streaming	
Format	Binary
Type	Complex interleaved U16
Acquisition mode	Streaming, finite records

66.3

66.3.1

In the context of this reference architecture, a phase-coherent system is interpreted as a system operating at the same frequency with a consistent phase relationship between any two channels in the system. *Phase coherence* is defined as the attribute of two or more waves where the relative phase between waves is constant during the time of the measurement.

Phase coherence performance of the test system is described using the following metrics:

- *Average channel-to-channel phase skew*?The average phase offset between two channels. Described by how it varies over time or from run to run.
- *Standard deviation of channel-to-channel skew*?One number that describes the variation of phase offset between channels around the average skew over time or from run to run.
- *Rx-Rx*?Channel-to-channel specifically referring to Rx channels.

66.3.2

Skew measurements are made based on IQ data files generated using the reference software provided as part of the Multichannel RF Reference Architecture. The measured system uses the MIMO loopback configuration, described in *Connecting the RF RX-TX Ports for a MIMO Loopback*, to route signals from all Tx ports to all Rx ports with a consistent path length. To measure the phase skew, the Angle-FFT method is used, wherein the IQ data generated from each channel is measured against the relevant reference signal for each measurement. The FFT method is used to extract the frequency and phase components of each signal in the vicinity of the signal?s fundamental frequency. This information is used to determine the magnitude of the phase offset between two signals in each measurement period. To compare signals throughout their entire duration, each signal is split into shorter measurement windows in which the instantaneous phase is calculated using the method described above.

66.3.3

Measurements are valid under the following conditions:

- Configured carrier frequency: 2 GHz (S-band/L-band)
- Configured baseband stimulus: 500 kHz sine wave, continuously transmitted for 1 hour, sampled at 6.25 MSample/s at receiver (250 MSample/s clock rate with decimation rate of 40)
- Measurement window: 4 ms (2,500 samples)
- Configuration:

  o Single device: Stimulus simultaneously received at two Rx channels on the same USRP
  o System: Stimulus simultaneously received at two Rx channels on arbitrary separate USRPs

- Analysis:

  o Where μ_n is average skew between two channels in measurement window *n*
  o μ_n - μ_0 is the average skew relative to t_0

- Range{μ_0, μ_1, ... , μ_N} is the range

Note Rx channels on the same USRP internally share the RX LO input to that device as shown in the following figure.

Figure 22. RF Configuration for Stability Measurements

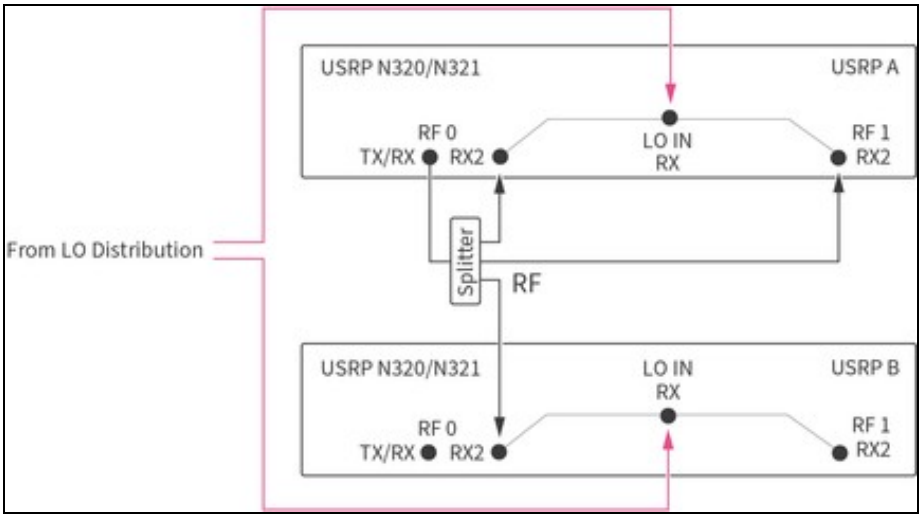
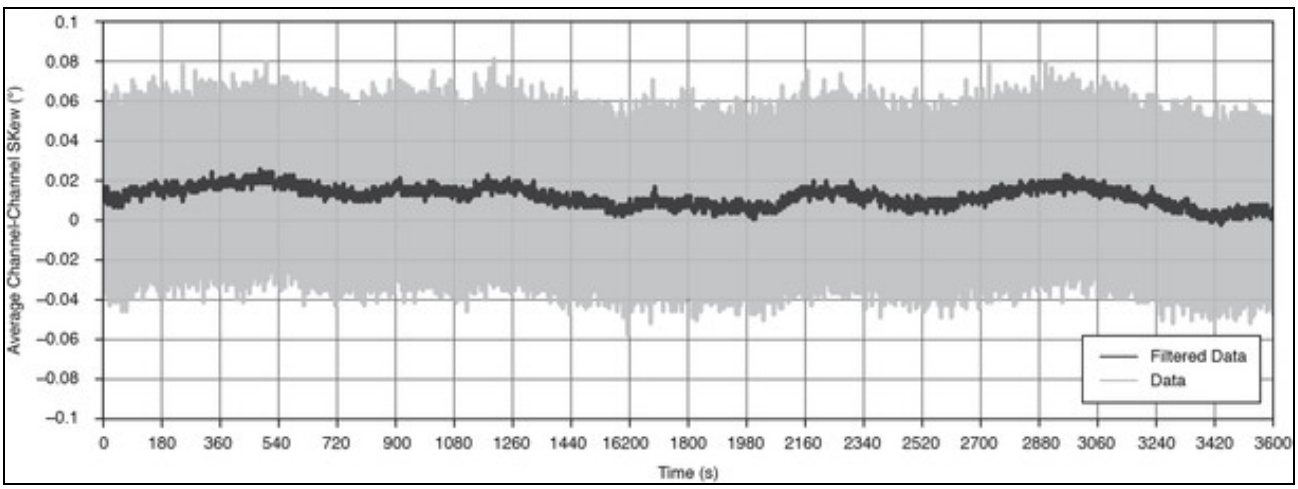
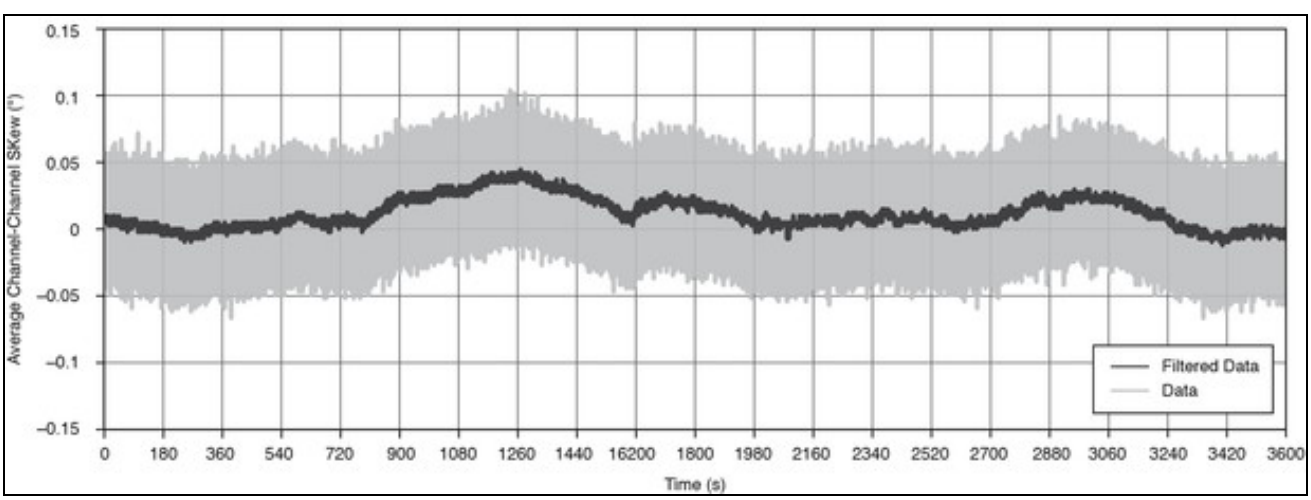


Figure 23. Average Channel-to-Channel Phase Skew between Two Rx Channels on the Same Device over One Hour, Relative to t_0



Range, single device 0.139°

Figure 24. Average Channel-to-Channel Phase Skew between Two Rx Channels on Separate Devices over One Hour, Relative to t_0



Range, system 0.167°

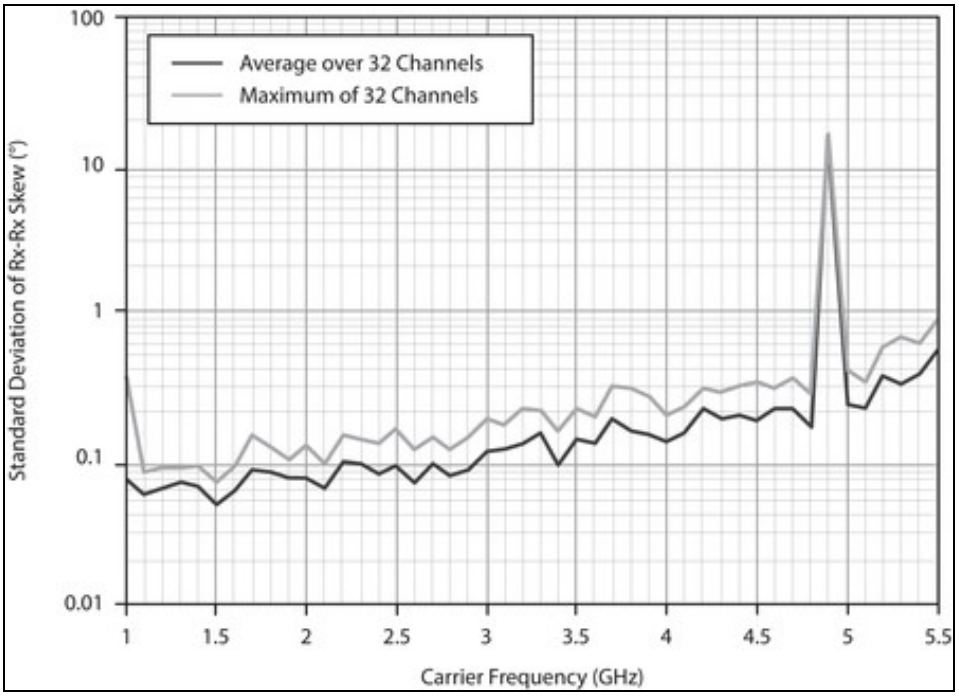
66.3.4

Measurements are valid under the following conditions:

- Sample data generation: 32 Rx receiving simultaneously at each carrier frequency
- Carrier frequency range: 1.0 GHz to 5.5 GHz, steps of 100 MHz
- Configured baseband stimulus: 500 kHz sine wave, continuously transmitted for 56 µs, sampled at 250 MSample/s
- Measurement window: 8 µs (2,000 samples at 250 MSample/s)
- Analysis:

  o Where μ_{nx} is the average skew between channel *n* and reference channel in measurement window x
  o sd_n = ?{μ_{n0}, μ_{n1},..., μ_{n6}} is the standard deviation of average skew for channel n using all measurement windows
  o Average over 32 channels is µ{sd_1...}
  o Maximum of 32 channels is max{sd_1...}

Figure 25. Standard Deviation in Short-Term Rx-Rx Phase Skew as a Function of Carrier Frequency



66.3.5

Measurements are valid under the following conditions:

- 10 runs
- Transmit from one Tx to all 32 Rx, repeat 32 times, once with each Tx
- Configured carrier frequency: 2 GHz
- Configured baseband stimulus: 56 µs, sampled at 250 MSample/s
- Measurement window: 8 µs (2,000: samples)
- Analysis:
- Where m_{nx} is Rx n-to-Rx 0 average skew in run *x*

  o sd_n = ?{m_{n0}, m_{n1},..., m_{n9}} is the standard deviation from run to run of Rx-Rx skew for channel *n*
  o r_n = Max(Abs({m_{n0}, m_{n1},..., m_{n9}} ? m_{n0})) is the range from run to run of Rx-Rx skew for channel *n*
  o µ{sd_1, sd_2,...,sd_{32}} is the system average standard deviation of Rx-Rx skew
  o Maximum{sd_1, sd_2,...,sd_{32}} is the system maximum standard deviation of Rx-Rx skew
  o µ{r_1, r_2,...,r_{32}} is the system average range of Rx-Rx skew
  o Maximum{r_1, r_2,...,r_{32}} is the system maximum range of Rx-Rx skew

66.3.5.1

Run-to-run, standard deviation of channel-to-channel skew (sd_n)	
Average over 32 channels	0.021°
Maximum of 32 channels	0.145°
Range of average channel-to-channel skew over 10 runs (r_n)	
Average of 32 channels	0.070°
Maximum of 32 channels	0.483°

66.3.5.2

Note In each run analyzed here, the carrier frequency is 2.0 GHz. Users must recharacterize channel-to-channel skew when using different carrier frequencies, because average channel-to-channel skew in the system deviates by >>1° per channel.

Run-to-run, standard deviation of channel-to-channel skew (sd_n)	
Average over 32 channels	0.025°
Maximum of 32 channels	0.066°
Range of average channel-to-channel skew over 10 runs (r_n)	
Average of 32 channels	0.067°
Maximum of 32 channels	0.169°

66.3.6

Measurements are valid under the following conditions:

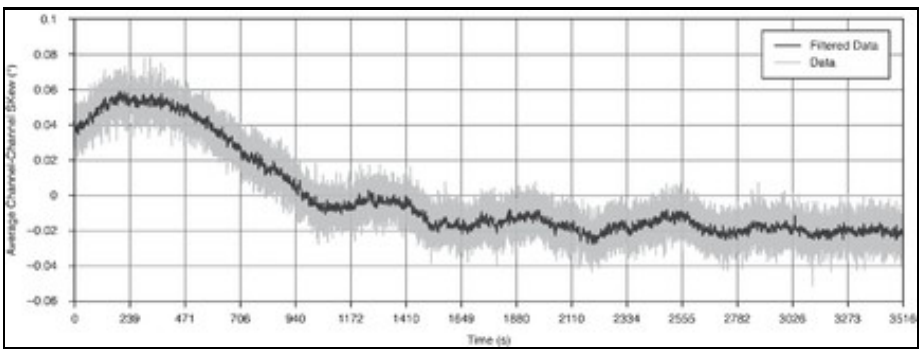
- Configured carrier frequency: 1 GHz
- Configured baseband stimulus: 500 kHz sine wave, continuously transmitted at 250 MSample/s for 1 hour, sampled at 3.2 GSample/s in finite windows
- Measurement window: 35 ms (120,000 samples)
- Configuration:

  o Transmitters: Stimulus simultaneously transmitted from any two channels
  o Receivers: Stimulus simultaneously received at two Rx channels on external acquisition device

- Analysis:

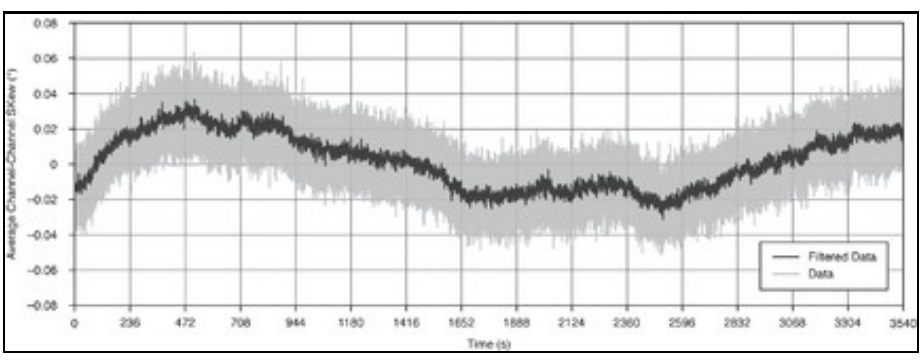
  o Where μ_n is average skew between two channels in measurement window n
  o μ_n - μ_0 is the average skew relative to t_0
  o Range{μ_0, μ_1, ... , μ_N} is the range

Figure 26. Average Channel-to-Channel Phase Skew between Two Tx Channels on the Same Device over One Hour, Relative to t_0



Range, single device	0.128°

Figure 27. Average Channel-to-Channel Phase Skew between Two Tx Channels on Separate Devices over One Hour, Relative to t_0



Range, system	0.115°

66.3.6.1

Measurements are valid under the following conditions:

- 50 runs
- Transmit from 16 Tx
- Configured carrier frequency: 1 GHz
- Configured baseband stimulus: 35 µs, sampled at 3.4 GSample/s
- Measurement window: 35 µs (120,000: samples)
- Analysis:

  o Where m_{nx} is Tx n-to-Tx 0 average skew in run x
  o sd_n = ?{m_{n0}, m_{n1},..., m_{n9}} is the standard deviation from run to run of Tx-Tx skew for channel n
  o r_n = Max(Abs({m_{n0}, m_{n1},..., m_{n9}} - m_{n0})) is the range from run to run of Tx-Tx skew for channel n
  o µ{sd_1, sd_2,...,sd_{32}} is the system average standard deviation of Tx-Tx skew

o Maximum{ sd_1, sd_2,...,sd_{32}} is the system maximum standard deviation of Tx-Tx skew
o μ{r_1, r_2,...,r_{32}} is the system average range of Tx-Tx skew
o Maximum{ r_1, r_2,...,r_{32}} is the system maximum range of Tx-Tx skew

66.3.6.1.1

Run-to-run, standard deviation of channel-to-channel skew (sd_n)	
Average over 16 channels	0.009°
Maximum of 16 channels	0.019°
Range of average channel-to-channel skew over 50 runs (r_n)	
Average of 16 channels	0.034°
Maximum of 16 channels	0.070°

66.3.6.1.1.1

Run-to-run, standard deviation of channel-to-channel skew (sd_n)	
Average over 16 channels	0.008°
Maximum of 16 channels	0.018°
Range of average channel-to-channel skew over 50 runs (r_n)	
Average of 16 channels	0.033°
Maximum of 16 channels	0.070°

66.4

66.4.1

Measurements are valid under the following conditions:

- Configured frequency: 2.5 GHz (S-band)
- Configured stimulus: Single-tone
- Number of channels: Varied
- Duration: Single run, 10 s

Table 5. Streaming to Memory Data Transfer Rate

Streaming Rate (MSample/s per Channel)	Number of Channels	Measurement Time (s)
33.33	32	10
50	16	10
122.88	8	10

Note For more information about supported sampling and master clock rates, refer to the *USRP N300/N310/N320/N321 Getting Started Guide*.

66.4.2

Measurements are valid under the following conditions:

- Configured frequency: 2.5 GHz (S-band)
- Configured stimulus: Single-tone
- Number of channels: Varied

Table 6. Streaming to Disk Data Transfer Rate

Streaming Rate (MSample/s per Channel)	Number of Channels	Measurement Time (s)
33.33	32	10
50	16	10
62.5	8	10

66.4.3

Measurements are valid under the following conditions:

- Configured frequency: 2.5 GHz (S-band)
- Configured stimulus: Single-tone
- Number of channels: Varied

Table 7. Streaming Simultaneously to/from Disk Rate

Streaming Rate (MSample/s per Channel)	Number of Channels	Measurement Time (s)

11.11	32	10
25	16	10
62.5	8	10
62.5	4	10

66.4.4

Measurements are valid under the following conditions:

- Configured frequency: 2.5 GHz (S-band)
- Configured stimulus: Single-tone
- Number of channels: Varied

Table 8. Streaming Simultaneously from Host to Host Memory Rate

Streaming Rate (MSample/s per Channel)	Number of Channels	Measurement Time (s)
250	8	7200

Table 9. Streaming Simultaneously from Host to Host Disk Rate

Streaming Rate (MSample/s per Channel)	Number of Channels	Measurement Time (s)
125	8	10

67

NI tested and configured the Multichannel RF Reference Architecture for a 32x32 system. It is possible to build a 64x64 (or greater) system.

67.1

When the system channel count increases to 64 channels and greater, consider the following:

- The MIMO loopback, 10 MHz and PPS, and LO distribution connections require additional tiers.
- Drift increases for typical measurements.

67.2

Refer to *Connecting the LO Distribution* for information about LO distribution connections and best practices.

> **Note** This configuration is used to synchronize both the Rx and Tx to the same LO resulting in the same center frequencies for both Tx and Rx. To synchronize different frequencies, connect RX LO OUT to RX LO IN and TX LO OUT to TX LO OUT. Refer to *USRP N320/N321 LO Distribution* for connection diagrams.

The following table lists the tiers for the 64x64 system configuration.

Table 8. 64x64 LO Sources and Tiers

System	Tiers	USRP N321 Minimum Quantity	USRP N320 Maximum Quantity
64x64	LO Source (0), 1, 2, 3	11	21

The following diagram represents LO distribution signal connections.

For the 64x64 system, all RX LO OUT cables originating from the LO Source USRP N321 must be the same length. All RX LO OUT and TX LO OUT cables originating from the Tier 1 USRP N321s must be the same length. All RX LO OUT and TX LO IN cables originating from the Tier 2 USRP N321s must be the same length. Note that you cannot combine LO signals.

Figure 28. LO Sharing (64x64)

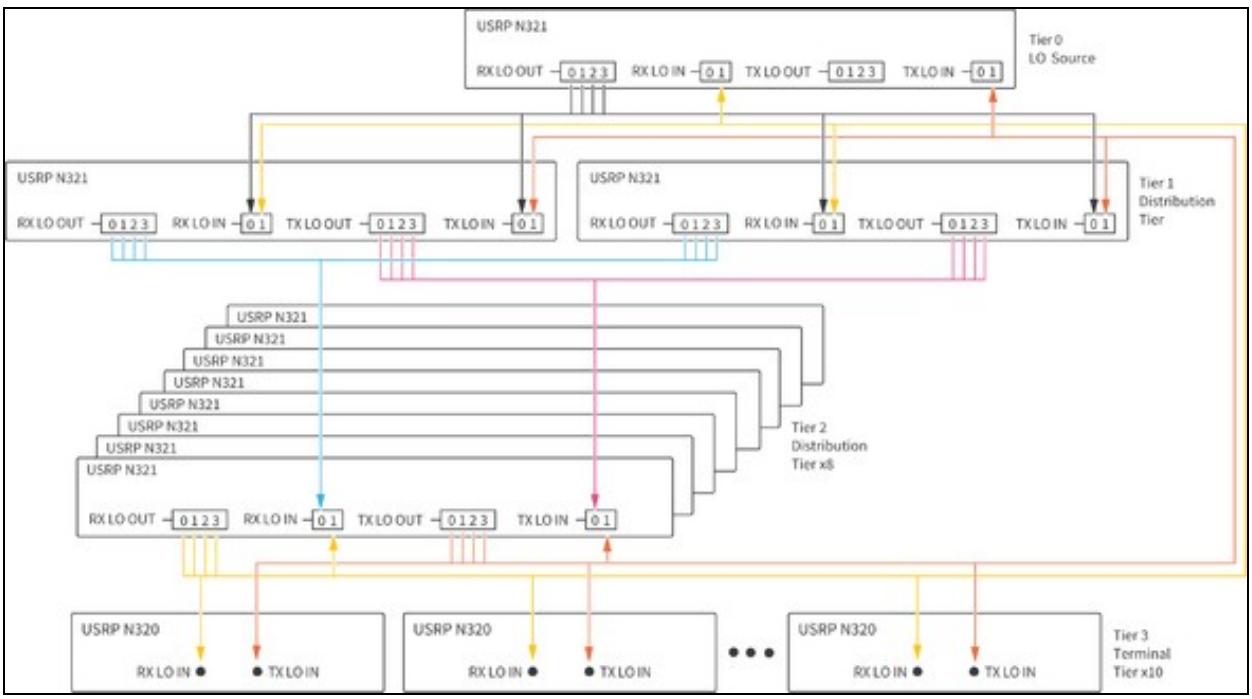


Table 9. LO Sharing Cabling (64x64)

Cable Color in Diagram	Cable Tier	Number of Cables in Tier
	LO Source Tier 0	4
	Distribution Tier 1, RX LO OUT	8
	Distribution Tier 1, TX LO OUT	8
	Distribution Tier 2, RX LO OUT	32
	Distribution Tier 2, TX LO OUT	32

68

68.1

Refer to the following documents to learn more about the Multichannel RF Reference Architecture, software, and hardware components:

- GitHub repository?Contains the system bill of materials and reference software example and library files needed to set up, install, and run your Multichannel RF Reference Architecture system. Visit https://github.com/EttusResearch/refarch-multich.
- *USRP N300/N310/N320/N321 Getting Started Guide*?Describes how to install software, how to set up your USRP, and confirm that your device is operating properly. Visit https://kb.ettus.com/USRP_N300/N310/N320/N321_Getting_Started_Guide.
- *N320/N321* specifications?Lists specifications for the USRP N321 and USRP N320. Visit https://kb.ettus.com/N320/N321.
- *USRP Hardware Driver and USRP Manual?*Describes how to use the USRP N321 and USRP N320 and how to use the API to connect to devices through your own software. Visit https://files.ettus.com/manual/index.html.
- *USRP N320/N321 LO Distribution*?Describes LO distribution connections and commands. Visit https://kb.ettus.com/USRP_N320/N321_LO_Distribution.
- *Using the RFNoC Replay Block*?Describes the basic use of the RFNoC Replay block in UHD and how to run the UHD Replay example. Visit https://kb.ettus.com/Using_the_RFNoC_Replay_Block.
- *Data Plane Development Kit Getting Started Guide for Linux*?Describes how to install and configure DPDK software in a Linux environment. Visit https://doc.dpdk.org/guides-20.11/linux_gsg/intro.html.

68.2

- Using Open-Source Tools with USRP Hardware for SDR Applications?Visit learn.ni.com, select **View Library**, and search for `using open-source tools with USRP`.
- Open-Source SDR Knowledge Base?Visit https://kb.ettus.com/Knowledge_Base.
- SDR-Academy Informal Getting Started Videos?Visit the SDR Academy at www.ettus.com/support/sdr-academy/.
- RFNoC Workshop from GRCon 2020?Visit https://kb.ettus.com/Training_Workshops to find RFNoC video links and workshop materials.

69

1. ? The *USRP N320/N321 LO Distribution* Knowledge Base article does not show one LO signal shared across Tx and Rx.
2. ? *Stability* reports variation in the value of average channel-to-channel phase skew over a long (>>1 s) duration.
3. ? *Short-term* describes variation in channel-to-channel phase skew over a short duration (<100 µs).
4. ? A *USRP session* is a period in which communication links between chips in the USRP are established and not changed. Session reset is accomplished by restarting the reference software for each run.
5. ? A *USRP configuration* is a period in which the digital path within the USRP is configured and not changed. Configuration reset is accomplished by changing carrier frequency from 2.0 GHz to 2.1 GHz and back to 2.0 GHz between runs.
6. ? Measurements were taken with a two-port direct sampling analyzer. The analyzer port 0 was directly connected to USRP 0 Channel 0. Port 1 of the analyzer was connected to the output of a 16-channel RF multiplexer. All other USRP channels were connected to the input of the 16-channel RF multiplexer. USRP 0 Channel 0 was compared to each other channel sequentially.
7. ? *Streaming to memory* is measured as the sustained data transfer rate without any overflows.
8. ? *Streaming to disk* is measured as the sustained data transfer rate without any overflows.
9. ? *Streaming simultaneously to/from disks* is defined as streaming from the host to the radio for transmission, while simultaneously receiving on the radio to a file on the host. The rates are measured as sustained data transfer rates without any overflows.
10. ? *Streaming Simultaneously with DPDK to/from Disk Rates* Streaming simultaneously with DPDK to/from disk is defined as streaming from the host to the radio for transmission, while simultaneously receiving on the radio to a file on the host. The rates are measured as sustained data transfer rates without any overflows using the Trenton Systems 3U BAM.

70 Workshop Tutorial

70.1

There are three training workshops/tutorials offered by Ettus Research, which are listed below.

These training workshops/tutorials can be run as a lecture/seminar, or as a hands-on workshop. They can be run at the NI office in Austin, Texas, USA, or at the location of the customer. These workshops/tutorials are run ad hoc and upon request, and there is no regular schedule for when they are run.

For questions about these workshops/tutorials, please contact us at support@ettus.com.

- **USRP Open-Source Toolchain: UHD and GNU Radio** [NI P/N 910915-21]: This tutorial provides a thorough and practical introduction to the USRP hardware and the open-source software toolchain (UHD and GNU Radio). After exploring the hardware and the architecture of the USRP family of software-defined radios, we will discuss topics such as getting started using a new USRP device, using the UHD device driver, programming the USRP from C++ using the UHD API, using GNU Radio with the USRP, creating and running flowgraphs from both GRC and Python, managing FPGA images, viewing packetized I/Q data with Wireshark, and debugging and triaging common technical problems, such as with network configuration, flow-control errors, and RF issues. Several demos and examples will be shown, such as performing real-time spectrum monitoring, transmitting pre-recorded waveforms, implementing an FM transmitter and receiver, and running an OFDM transceiver. Several additional complementary open-source tools will also be discussed, such as GQRX, Fosphor, Inspectrum, and several GNU Radio Out-of-Tree (OOT) modules. We will explore several demonstrations of wireless systems running on the USRP, such as a record-and-playback system, a spectrum-painting tool, and an LTE network scanner tool. We will include several additional focused topics, such as high-rate 10 Gigabit Ethernet networking and I/Q data streaming, host system performance tuning, USRP device recovery, and various operational best-practices. Attendees should gain a solid foundation and practical understanding of how to configure, program, and operate the USRP to implement a wide range of wireless systems and applications.
    - Workshop_GnuRadio_Materials_20171212.tar.gz (TAR-GZ) (13 MB) (2017-12-12)
    - Workshop_GnuRadio_Slides_20190507.pdf (PDF) (45 MB) (2019-05-07)

- **USRP FPGA Processing Using the RFNoC Framework** [NI P/N 910916-21]: This tutorial provides an in-depth introduction to the RFNoC framework, including a discussion on its design and capabilities, several live demonstrations, and a walk-through of implementing a new user-defined RFNoC block and integrating it into both UHD and GNU Radio. The RFNoC (RF Network-on-Chip) framework is the FPGA architecture used in USRP devices. The RFNoC framework enables users to program the USRP FPGA, and facilitates the integration of custom FPGA-based algorithms into the signal processing chain of the USRP radio. Users can create modular, FPGA-accelerated SDR applications by chaining multiple RFNoC Blocks together and integrating them into both C++ and Python programs using the UHD API, and into GNU Radio flowgraphs. Attendees should gain a practical understanding of how to use the RFNoC framework to implement custom FPGA processing on the USRP radio platform.
    - Workshop_RFNoC_4_GRCon20_Part_1_Slides.pdf (PDF) (2.7 MB) (2020-09-14)
    - Workshop_RFNoC_4_GRCon20_Part_2_Slides.pdf (PDF) (2.7 MB) (2020-09-14)
    - Video Presentation (YouTube URL) (2020-09-14)
    - Application Note for Getting Started with RFNoC in UHD 4.0
    - RFNoC Specification Document

- **USRP Open-Source 5G/NR System Implementation** [NI P/N 910917-21]: This tutorial provides a detailed overview of how to implement a 3GPP-compliant 5G/NR testbed using the USRP radio with the open-source software stacks, srsRAN from Software Radio Systems (SRS), and OpenAirInterface (OAI) from Eurecom, for research, development, and prototyping. We examine both the base station (eNB and gNB) side, as well as the handset (UE) side. We examine three implementations for the UE: an emulated software UE; a commercial handset; and a stand-alone modem module. We discuss in detail how to install, configure, and operate the hardware and software for the base station side and the handset side, as well as for the core network, in order to create a fully functional network. We discuss various aspects of radio and network planning and implementation, discuss network operation monitoring, and discuss performance and throughput measurements. We show a video of a demonstration of the real-time operation of a 5G network. Attendees should gain a practical understanding of how to use USRP devices to implement 5G/NR wireless networks.
    - Application Note for 5G OAI USRP Reference Architecture & Testbed

71 Suggested Reading

71.1

- "Wireless Pi" Blog by Dr Qasim Chaudhari
- "The Scientist and Engineer's Guide to DSP" (free book)
- "Complex to Real" Blog
- "DSP Related" Blog
- "Mathematics of the DFT" (free book)
- "Introduction to Digital Filters" (free book)
- NI Application Note "What is I/Q Data?"
- "Quadrature Signals: Complex, But Not Complicated" by Richard Lyons
- "Quadrature Signals: Complex, But Not Complicated" by Richard Lyons
- Free SDR Book from The MathWorks
- GNU Radio Suggested Reading

71.2

71.2.1

- Guide to Wireless Communications (3rd Edition) by Jorge Olenewa

  https://www.amazon.com/Guide-Wireless-Communications-Jorge-Olenewa/dp/1111307318/

- Wireless Communications: Principles and Practice (2nd Edition) by Theodore S. Rappaport

  https://www.amazon.com/Wireless-Communications-Principles-Practice-2nd/dp/0130422320/

- Wireless Communications from the Ground Up: An SDR Perspective (2nd Edition) by Qasim Chaudhari

  https://wirelesspi.com/
  https://wirelesspi.com/book/
  https://www.amazon.com/gp/product/1729732232/

- Digital Communications: Fundamentals and Applications (2nd Edition) by Bernard Sklar

  https://www.amazon.com/Digital-Communications-Fundamentals-Applications-2nd/dp/0130847887/

- Digital Communications (5th Edition) by John Proakis, Masoud Salehi

  https://www.amazon.com/Digital-Communications-5th-John-Proakis/dp/0072957166/

- Digital and Analog Communication Systems (8th Edition) by Leon W. Couch

  https://www.amazon.com/Digital-Analog-Communication-Systems-8th/dp/0132915383/

- Digital Communication Systems Engineering with Software-Defined Radio by Alexander Wyglinski, Di Pu

  https://www.amazon.com/Digital-Communication-Engineering-Software-Defined-Communications/dp/1608075257/

- Cognitive Radio Communications and Networks: Principles and Practice by Alexander Wyglinski, Maziar Nekovee, Thomas Hou

  https://www.amazon.com/Cognitive-Radio-Communications-Networks-Principles/dp/0123747155/

- Modern Digital and Analog Communication Systems (4th Edition) by B. P. Lathi,? Zhi Ding

  https://www.amazon.com/Digital-Communication-Electrical-Computer-Engineering/dp/0195331451/

- Digital Communication (3rd Edition) by John R. Barry, Edward A. Lee, David G. Messerschmitt

  https://www.amazon.com/Digital-Communication-John-R-Barry/dp/0792375483/

- Digital Communications with Emphasis on Data Modems: Theory, Analysis, Design, Simulation, Testing, and Applications by Richard W. Middlestead

  https://www.amazon.com/Digital-Communications-Emphasis-Data-Modems/dp/0470408529/

- Introduction to Analog and Digital Communications (2nd Edition) by Simon Haykin, Michael Moher

  https://www.amazon.com/Introduction-Analog-Digital-Communications/dp/0471432229/

- Wireless Communications (2nd Edition) by Andreas F. Molisch

  https://www.amazon.com/Wireless-Communications-Andreas-F-Molisch/dp/0470741864/

- Introduction to Wireless Digital Communication: A Signal Processing Perspective by Robert W. Heath Jr.

  https://www.amazon.com/Introduction-Wireless-Digital-Communication-Perspective/dp/0134431790/

- Communications Receivers: Principles and Design (4th Edition) by Ulrich L. Rohde, Jerry C. Whitaker, Hans Zahnd

  https://www.amazon.com/Communications-Receivers-Principles-Design-Fourth/dp/0071843337/

- Synchronization in Digital Communication Systems by Fuyun Ling, John Proakis

  https://www.amazon.com/Synchronization-Digital-Communication-Systems-Fuyun/dp/110711473X/

71.2.2

- Understanding Digital Signal Processing (3rd Edition) by Richard G. Lyons

  https://www.amazon.com/Understanding-Digital-Signal-Processing-3rd/dp/0137027419/

- Discrete-Time Signal Processing (3rd Edition) by Alan V. Oppenheim, Ronald W. Schafer

  https://www.amazon.com/Discrete-Time-Signal-Processing-3rd-Prentice-Hall/dp/0131988425/

- Digital Signal Processing (4th Edition) by John G. Proakis, Dimitris K Manolakis

  https://www.amazon.com/Digital-Signal-Processing-John-Proakis/dp/0131873741/

- Digital Signal Processing in Modern Communication Systems by Andreas Schwarzinger

  https://www.amazon.com/Digital-Signal-Processing-Communication-Systems/dp/0988873508/

- Digital Signal Processing 101: Everything You Need to Know to Get Started (2nd Edition) by Michael Parker

  https://www.amazon.com/Digital-Signal-Processing-101-Second/dp/0128114533/

- Think DSP: Digital Signal Processing in Python by Allen B. Downey

  https://www.amazon.com/Think-DSP-Digital-Signal-Processing/dp/1491938455/

71.2.3

- Software-Defined Radio for Engineers by Travis F. Collins, Robin Getz, Di Pu, and Alexander M. Wyglinski

  https://www.analog.com/en/education/education-library/software-defined-radio-for-engineers.html

- Software Receiver Design: Build your Own Digital Communication System in Five Easy Steps by C. Richard Johnson Jr, William A. Sethares, Andrew G. Klein

  https://www.amazon.com/Software-Receiver-Design-Digital-Communication/dp/0521189446/

- Software Defined Radio using MATLAB & Simulink and the RTL-SDR by Robert W Stewart, Kenneth W Barlee, Dale S W Atkinson

  http://www.desktopsdr.com/
  https://www.amazon.com/Software-Defined-MATLAB-Simulink-RTL-SDR/dp/0992978718/

71.2.4

- RF Microelectronics (2nd Edition) by Behzad Razavi

  https://www.amazon.com/Microelectronics-Communications-Engineering-Technologies-Rappaport/dp/0137134738/

- Wireless Receiver Design for Digital Communications (2nd Edition) by Kevin McClaning

  https://www.amazon.com/Wireless-Receiver-Digital-Communications-Telecommunications/dp/1891121804/

- Microwave and RF Design of Wireless Systems by David M. Pozar

  https://www.amazon.com/Microwave-Rf-Design-Wireless-Systems/dp/0471322822/

71.2.5

- The ARRL 2018 Handbook for Radio Communications

  http://www.arrl.org/arrl-handbook-2018
  https://www.amazon.com/ARRL-Handbook-Radio-Communications-Softcover/dp/1625950713/
  https://www.amazon.com/ARRL-Handbook-Radio-Communications-Hardcover/dp/1625950721/

- Software Defined Radio: for Amateur Radio Operators and Short Wave Listeners by Andrew Barron

  https://www.amazon.com/Software-Defined-Radio-Operators-Listeners/dp/1534992421/

- An Introduction to HF Software Defined Radio: SDR for Amateur Radio Operators by Andrew Barron ZL3DW

  https://www.amazon.com/introduction-Software-Defined-Radio-Operators/dp/1500119938/

71.3

- Spectrum Sensing on LTE Femtocells for GSM Spectrum Re-Farming Using Xilinx FPGAs
- Ship Detection with DVB-T Software Defined Passive Radar
- https://wiki.gnuradio.org/index.php/AcademicPapers
- http://ieeexplore.ieee.org/Xplore/home.jsp

72 Suggested Videos

72.1

- Video tutorial from the GNU Radio Project

  https://wiki.gnuradio.org/index.php/Guided_Tutorials

- Video tutorial from Ettus Research

  http://files.ettus.com/tutorials/

- GNU Radio Tutorials by Balint Seeber (five-part series)

  https://www.youtube.com/playlist?list=PL618122BD66C8B3C4
  https://www.youtube.com/watch?v=N9SLAnGlGQs&list=PL618122BD66C8B3C4

- "Introduction to GNU Radio from the DCC" by Tom Rondeau (four-part series)

  https://www.youtube.com/watch?v=_hGNT1w-jig
  https://www.youtube.com/watch?v=cg3TA3EDx78
  https://www.youtube.com/watch?v=nemfS9QAYHc
  https://www.youtube.com/watch?v=94R2qE7mEc4

72.2

- USRP B200: Exploring the Wireless World by Balint Seeber

  https://www.youtube.com/watch?v=cygDXeZaiOM

- How To Build an FM Receiver with the USRP in Less Than 10 Minutes

  https://www.youtube.com/watch?v=KWeY2yqwVA0

- Going Deeper Into GNU Radio Companion by Hak5

  https://www.youtube.com/watch?v=QhMET2mld14

- OpenBTS - The Well Tempered Hacker Ep 4 by Anders Brownworth

  https://www.youtube.com/watch?v=pTb1_v8M6iA

- Tough Concepts: Understanding How Broadcast Frequency Modulation Works

  https://www.youtube.com/watch?v=IIeQVpW5APk

- ASCII Art FFT with USRP and UHD

  https://www.youtube.com/watch?v=eQy9rvw2hFU

- "Why Doesn't My Signal Look Like the Textbook?" by Matt Ettus at GNU Radio Conference 2016

  https://www.youtube.com/watch?v=PNMOwhEHE6w

- Basics of IQ Signals and IQ modulation & demodulation - A tutorial

  https://www.youtube.com/watch?v=h_7d-m1ehoY

73 CGRAN

The Comprehensive GNU Radio Archive Network (CGRAN) is a free open source repository for 3rd party GNU Radio applications a.k.a Out Of Tree Modules that are not officially supported by the GNU Radio project.

Website: CGRAN

74 SDR Events

- Cyberspectrum
- GNU Radio Conference
- NEWSDR
- FOSDEM
- DEFCON
- Black Hat
- SDRA
- CCC

75 GNU Radio Conference

75.1

The GNU Radio Conference (GRCon) is the annual conference for the GNU Radio project and the community, and has established itself as one of the premier industry events for Software Defined Radio. It is a week-long conference that includes high-quality technical content and valuable networking opportunities. The event attracts over 300 attendees from industry, academia, and government. With an annual program that has broad appeal, GRCon attracts people new to SDR just looking to learn more, folks that want to keep their finger on the pulse and direction of the industry, and seasoned developers ready to show off their latest work.

75.2

Charlotte, North Carolina, USA

14 ? 18 September 2020

75.3

https://pubs.gnuradio.org/index.php/grcon/issue/archive

https://wiki.gnuradio.org/index.php/GNURadioConference

75.4

https://www.gnuradio.org/grcon/grcon19/

75.5

75.5.1

https://www.gnuradio.org/grcon/grcon19/

https://www.youtube.com/playlist?list=PLE7tQUdRKcyaaF1pm1MfQm_g1TF2CP1JZ

https://pubs.gnuradio.org/index.php/grcon

75.5.2

https://www.gnuradio.org/grcon/grcon18/

https://www.youtube.com/playlist?list=PLbBQHMnVMR431egzt2AbEVeeiWD9j64eZ

75.5.3

http://gnuradio.org/grcon-2017/

https://www.gnuradio.org/grcon-2017/program/grcon17-presentations/

https://www.youtube.com/playlist?list=PLbBQHMnVMR43mM18y4r8L7l8bbYMgloFx

https://www.gnuradio.org/blog/2017-09-11-gnu-radio-conference-2017-proceedings-live/

75.5.4

http://gnuradio.org/grcon-2016/

http://gnuradio.org/grcon-2016/talks/

http://lists.gnu.org/archive/html/discuss-gnuradio/2016-11/msg00084.html

https://www.youtube.com/playlist?list=PLbBQHMnVMR43iHZRQ2Uw_UVQb65rOEnow

75.5.5

http://www.trondeau.com/grcon15-presentations/

76 NEWSDR

76.1

The New England Workshop on Software Defined Radio (NEWSDR) has been running for 8 years. The goal of these workshops is to provide a forum through which individuals working on SDR-related projects in the greater Boston area can get together in order to collaborate and introduce SDR concepts to those interested in furthering their knowledge of SDR capabilities and available resources. The event attracts attendees from industry, academia, and government. In 2017, attendance was approximately 110 people. Registration is free.

76.2

Worcester Polytechnic Institute (WPI)

Worcester, Massachusetts, USA

76.3

Thursday 3 May 2018, 17:00 to 21:00

Friday 4 May 2018, 08:00 to 16:00

76.4

http://www.sdr-boston.org/

76.5

```
*********************************************************************
                            NEWSDR 2018

              New England Workshop on Software Defined Radio

                           Eighth Annual

                          Evening Workshops
                           Thursday May 3
                           17:00 to 21:00

                          Day-Long Symposium
                            Friday May 4
                           08:00 to 16:00

                   Worcester Polytechnic Institute (WPI)
                     Worcester, Massachusetts, USA

                        http://www.sdr-boston.org/

        Registration and Poster Submissions Open on February 15

*********************************************************************
Worcester Polytechnic Institute (WPI) will be hosting the 2018 New
England Workshop on Software Defined Radio (NEWSDR) on the evening
of Thursday May 3 and all day on Friday May 4.

As more information about this event becomes available, we will post
information on this mailing list and on our website at:
http://www.sdr-boston.org

Note that registration for this event is free,
and will open on February 15.

Please mark your calendars!
We hope to see you there!

*********************************************************************
                                 EOF
*********************************************************************
```

77 FOSDEM

77.1

FOSDEM (Free and Open Source Software Developers' European Meeting) is a non-commercial, volunteer-organized European event centered on free and open-source software development. It is aimed at developers and anyone interested in the free and open-source software movement. It aims to enable developers to meet and to promote the awareness and use of free and open-source software.

FOSDEM is held annually, usually during the first weekend of February, at the Université Libre de Bruxelles Solbosch campus in the southeast of Brussels, Belgium. Registration is free.

Entrance and participation in the event is entirely free. It is financed by sponsors who accept the non-commercial nature of the event, and by donors amongst the visitors. Donors receive an incentive in form of a token present. Everything is organized and set up by volunteers.

77.2

The SDR devroom has talks about frameworks, research projects, applications, hardware, and tools. This is the place for Open Source SDR developers to meet, exchange ideas, and start working together to improve the state of Open Source and SDR.

77.3

Brussels, Belgium

77.4

1-2 February 2020

77.5

https://fosdem.org/

https://en.wikipedia.org/wiki/FOSDEM

77.6

https://fosdem.org/2020/

https://fosdem.org/2020/schedule/track/free_software_radio/

77.7

77.7.1

https://archive.fosdem.org/2019/

https://archive.fosdem.org/2019/schedule/track/free_software_radio/

77.7.2

https://archive.fosdem.org/2018/

https://archive.fosdem.org/2018/schedule/track/software_defined_radio/

77.7.3

https://archive.fosdem.org/2017/

https://archive.fosdem.org/2017/schedule/track/software_defined_radio/

77.7.4

https://archive.fosdem.org/2016/

https://archive.fosdem.org/2016/schedule/track/software_defined_radio/

77.7.5

https://archive.fosdem.org/2015/

https://archive.fosdem.org/2015/schedule/track/software_defined_radio/

78 Cyberspectrum

78.1

The Bay Area SDR Meetup will serve as a forum to exchange knowledge and ideas related to Software Defined Radio (the software and hardware), and generally aim to get people excited about all the applications that can be realised with the technology. At each meetup, attendees will have the opportunity to present their work/ideas to the group.

Engineers, enthusiasts, hobbyists and people of all experience levels are welcome, no matter what your software/hardware background. Everyone is welcome to submit their ideas/presentations to the pool. For each meetup, a fixed number will be chosen to fit the format. Currently there will be a short show-and-tell section, followed by two 20 minute presentations (e.g. an introduction to an SDR topic and analysis of a mystery signal), and then to conclude a 40 minute in-depth presentation on an application. Suggestions/alterations to this format are welcome of course!

Meetup locations will alternate between the South Bay and San Francisco. Meetup presentations will be recorded and posted online (although you can opt out if you wish).

More Information and Upcoming Events: http://www.meetup.com/Cyberspectrum/

https://www.meetup.com/Cyberspectrum/about/

78.1.1

Live Stream: https://www.youtube.com/watch?v=DUGr_Z04SKs

- Kevin Reid (@switchborg): "A Visual Introduction to DSP for SDR"

A tour of DSP topics relevant to implementation of simple software-defined radios. Focuses on visual explanations of fundamental manipulations of digital signals, including analytic signals, frequency shifting, sampling rates, filtering, and the discrete Fourier transform. http://switchb.org/kpreid/

- @SigBlips: the 'baudline' signal analyser, and some interesting signals

78.1.2

Live Stream: https://www.youtube.com/watch?v=Tdn6LDeAdHo

- Nick Foster @bistromath

Satellite communications with GNU Radio, SDR hardware, homebrew antennas and satellite tracking.

- Josh Myer @xek

A couple months back, some friends and I captured a bunch of RF telemetry from a rocket launch down at Vandenburg AFB, down by LA. I'll talk about how we decided what to capture, what we captured, and how we analyzed the many, many gigabytes of RF that wound up on a hard disk that morning.

- Jonathon Pendlum @SDRJon

RFNoC (RF Network on Chip), a new framework for Ettus third generation devices (X300 & E300) that aims to make FPGA acceleration in SDRs more easily accessible. I will cover some background on FPGAs and their use in SDR, the motivation and design of RFNoC, and conclude with a few live demos.

78.1.3

Live Stream: https://www.youtube.com/watch?v=MFBkX4CNb08

- Julian Arnold (AKA Perpetual Intern of the Month, @broadcrap):

A discussion on creating a simple SDR Doppler RADAR, OR decoding WiFi[masked]a packets - which one will he choose!?

- Derek Kozel (@derekkozel):

"Digital modulation schemes such as Phase Shift Keying (PSK) convert data bits to analog signals for transmission. This mapping is known as a constellation. For simple schemes, optimal constellations can be calculated, but for higher-order modulation schemes with more points, it becomes very difficult to mathematically determine an optimal constellation. Evolutionary algorithms provide a simple and pragmatic way to find good answers without advanced knowledge of communications theory. This talk shows a start-to-finish implementation, including an introduction to digital signals and evolutionary algorithms."

- Alex Ray (@machinaut & Team Lunarnaut):

"We'll be talking about some of the radios, antennas and protocols we're working on as part of the just-started NASA Cube Quest Challenge, a competition to get high data rates to small satellites far away from the Earth.

Our team is still in the very early stages, but we've been building and testing antennas, as well as experimenting with modulation schemes. Along those lines, SDRs are great for out needs because we can be extremely flexible with protocols, bands, antennas, and more!

We'll show off what we've been working on so far, and can talk about what we plan on doing next."

78.1.4

Live Stream: https://www.youtube.com/watch?v=Iq07aQaB8mM

- Julian Arnold (@broadcrap) will talk in depth about building a Doppler RADAR with cantennas and SDR
- Ief Kox (@iefkox) will talk from the other side of the world about MultiPSK and use it to analyse all manner of interesting signals: ACARS (HFDL, VDL), STANAG 4285, ALE, AMTOR Navtex, Wefax, DGPS, SSTV and more!
- Jesus Molina (@verifythentrust) will present on "Wardriving in the age of the Internet of Things with SDR":

In this talk I will present information on how to discover and map radio devices utilizing SDR, and I will present a new concept: Warwatching (yeah, watching IoT devices in real time!). Round 10 years ago we drove around picking up wireless signals from WIFI access points. Tools like Kismet were used to collect relevant information, and the data was then post processed to draw heat maps. Then Google crashed the party with their ever present cars, and with increasingly pervasive dynamic AP (phone hotspots, drones) it doesn?t make much sense anyway.

But SDR opens a new world for us: Wardriving in ANY frequency range. The amount of devices equipped with radio transmitting capabilities have increased, and is time to create new tools for discovery and pentesting in the age of the IoT. With SDR we can detect, listen and interact with several ?static? radio devices (cell towers, FM stations, etc), and also we can ?see? dynamic short range devices (drones, Bluetooth) and even actually watch them using augmented reality!. I will provide a short demonstration and the road ahead.

78.1.5

Live Stream: https://www.youtube.com/watch?v=ZxiphlitrAQ

- Kevin Reid (@switchbord): "An Update to a Visual Introduction to DSP"
- Matt Ettus: "Synchronisation and MIMO demystified"

Beyond a single radio, there are multiple antenna systems, geographically separated systems, and all manner of multi-radio configurations in-between. Matt will talk about what is necessary to make these systems work, and the different levels of timing & synchronisation involved.

- Martin Braun (@braun_noise): "The SDR Mythbusters: Is #cyberspectrum hard to do?"

SDR has a reputation for being very difficult. Is this actually true? GNU Radio developer Martin Braun gives a tour through a typical development cycle and the tools GNU Radio provides to make development as smooth as possible.

This talk is geared mostly towards enthusiasts who are looking start their own GNU Radio development, and want to know exactly which resources are available.

78.1.6

Live Stream: https://www.youtube.com/watch?v=GYFalvzo-nk

- Harvind Samra (CTO and Co-Founder of Range Networks):

"OpenBTS: A Software-Defined Mobile Network"

The OpenBTS software is a Linux application that uses a software-defined radio to present a standard 3GPP air interface to user devices, while simultaneously presenting those devices as SIP endpoints to the Internet. This forms the basis of a new type of wireless network which promises to expand coverage to unserved and underserved markets while unleashing a platform for telecom innovation.

- Jason Abele:

"Software Defined Radio without the Radio, using GNU Radio and a sound card to develop a receiver for atomic time from WWVB"

We will talk a little about DIY VLF/LF antennas, the history of WWVB and atomic clocks, and demonstrate how to use GNU Radio to turn a cheap SDR rig into a very expensive clock.

78.1.7

Live Stream: https://www.youtube.com/watch?v=BoFOt9AUWuE

- Moritz Fischer: A quick show-and-tell preview into decoding DECT using GNU Radio
- Surya Satyavolu: RADAR-Guided Wirelessly-Controlled Automated Driving

Surya will cover the technical problems to be solved to realize automated driving. It will focus on achieving reliable lateral guidance using RADAR and hard-real time longitudinal control using wireless, as well as current advances in RADAR technology that would help in realizing the vision.

- Balint Seeber: Walk-through on creating a simple wireless video streaming system using a webcam & GNU Radio

78.1.8

Live Stream: https://www.youtube.com/watch?v=HsDpMffRafg

- Matt Reilly: "SoDaRadio - A General Purpose Transceiver"

http://sodaradio.sourceforge.net/Site/SoDaRadio.html

- Josh Myer: "Decoding Radio Data System (RDS) from FM broadcast stations"

...is a great introductory in-depth SDR project. Josh will walk through his implementation in IPython, starting with sample capture and ending with decoding some of the RDS protocol frames.

Josh lives in the radio noise mess that is San Francisco, and is currently working on data acquisition and signal processing for a biofeedback product. He?s also in the beginning stages of a few radio capture and direction finding projects. You can find more of his bite-sized radio projects at his SDR Snippets page: http://www.joshisanerd.com/projects/sdr_snippets/

78.1.9

Live Stream: https://www.youtube.com/watch?v=NBfBnPPcuJw

- Martin Braun's to-be-regular update on GNU Radio news
- Tom Tsou: "A Guided Tour of LTE on SDRs"

Tom will speak about LTE fundamentals, the many available stacks that can run with SDRs, what their capabilities are, and what the future holds. He will also do some live demos!

- Moritz Fischer: Decoding DECT with GNU Radio - update

78.1.10

Live Stream: https://www.youtube.com/watch?v=eebEKbdFL-g

- "Using SDR & GNU Radio in Radio Astronomy" by Richard Prestage, NRAO
- Tim O'Shea on some more cool GNU Radio hackery

http://oshearesearch.com/tag/lambda-blocks/

GNU Radio Lambda blocks are a simplification of pure-python blocks for GNU Radio which allow for writing a new block from within GRC with a simple python lambda expression. We'll demonstrate the great signal processing hackery that can achieved with the stream and message versions of this block!

- "PSK Modems in GNU Radio" by Kiran Karra

https://kirankarra.wordpress.com/2015/08/26/qpsk-burst-receiver-synchronization/

PSK Modems in GNU Radio have typically used tracking loops which take time to converge and do not leverage reference signals, through re-thinking the approach to PSK demodulation in a message and burst based context we demonstrate a robust new way to build modems!

- "Hacking an RF Shock Collar" by Tim K

GNU Radio is an awesome tool for reverse engineering, but people seem to get stuck somewhere between "Complex to Mag", Audacity, and MS Paint. It's not as hard to get packets out in "real time" as you might think. In this session, I'll build a transceiver from the ground up for the shock collar from DEFCON 23's Wireless Village.

78.1.11

Live Stream: https://www.youtube.com/watch?v=tG70c3Zadek

- "Etch-A-SDR" by Nate (@devnulling)

In this talk I will be doing a quick show and tell of building the 'Etch-A-SDR'. The 'Etch-A-SDR' is a digital Etch-a-Sketch, that doubles as a fully contained SDR platform.

I am a programmer by day, SDR Enthusiast / Hobbyist, Maker, and Amateur Radio operator by night.

- Spread spectrum SATCOM Hacking: Attacking the GlobalStar Simplex Data Service" by Colby Moore (@colbymoore)

Recently, there have been several highly publicized talks about satellite hacking. However, most only touch on the theoretical rather than demonstrate actual vulnerabilities and real world attack scenarios. This talk will demystify some of the technologies behind satellite communications and do what no one has done before - take the audience step-by-step from reverse engineering to exploitation of the GlobalStar simplex satcom protocol and demonstrate a full blown signals intelligence collection and spoofing capability. I will also demonstrate how an attacker might simulate critical conditions in satellite connected SCADA systems.

In recent years, Globalstar has gained popularity with the introduction of its consumer focused SPOT asset-tracking solutions. During the session, I'll deconstruct the transmitters used in these (and commercial) solutions and reveal design and implementation flaws that result in the ability to intercept, spoof, falsify, and intelligently jam communications. Due to design tradeoffs these vulnerabilities are realistically unpatchable and put millions of devices, critical infrastructure, emergency services, and high value assets at risk.

Colby Moore is Synack's Manager of Special Activities. He works on the oddball and difficult problems that no one else knows how to tackle and strives to embrace the attacker mindset during all engagements. He is a former employee of VRL and has identified countless 0-day vulnerabilities in embedded systems and major applications. In his spare time you will find him focusing on that sweet spot where hardware and software meet, usually resulting in very interesting consequences.

78.1.12

Live Stream: https://www.youtube.com/watch?v=1K6LUAZpaWg

- Marcus Leech from SBRAC: "An integrated proof-of-concept 'all-digital' feed for 21cm radio astronomy"

We show ongoing work in designing and building a proof-of-concept 'all digital' feed for 21cm radio astronomy experiments. While manyprofessional radio astronomy observatories are using "digitize at the feed" techniques, amateur experiments (and successes) in this areaare very close to non-existent.

Digitizing at the feed carries many advantages, including overall system gain stability, and the ability to carry signals over cheap ethernet-over-fiber links. We'll show an example feed arrangement that uses a differential radiometry approach, and does much of the initial processing right atthe feed, including radiometry and spectral calculations, sending summary data to an ordinary PC host over ethernet.

Challenges and pitfalls will be discussed.

- Tobias Zillner from Cognosec: "ZigBee Smart Homes - A Hacker's Open House"

ZigBee is one of the most widespread communication standards used in the Internet of Things and especially in the area of smart homes. If you have for example a smart light bulb at home, the chance is very high that you are actually using ZigBee by yourself. Popular lighting applications such as Philips Hue or Osram Lightify and also popular smart home systems such as SmartThings or Googles OnHub are based on ZigBee. New IoT devices have often very limited processing and energy resources. Therefore they are not capable of implementing well-known communication standards like Wifi. ZigBee is an open, public available alternative that enables wireless communication for such limited devices.

ZigBee provides also security services for key establishment, key transport, frame protection and device management that are based on established cryptographic algorithms. So a ZigBee home automation network with applied security is secure and the smart home communication is protected?

No, definitely not. Due to ?requirements? on interoperability and compatibility as well as the application of ancient security concepts it is possible to compromise ZigBee networks and take over control of all included devices. For example it is easily possible for an external to get control over every smart light bulb that supports the ZigBee Light Link profile. Also the initial key transport is done in an unsecured way. It is even required by the standard to support this weak key transport. On top of that another vulnerability allows third parties to request secret key material without any authentication and therefore takeover the whole network as well as all connected ZigBee devices. Together with shortfalls and limitations in the security caused by the manufacturers itself the risk to this last tier communication standard can be considered as highly critical.

This talk will provide an overview about the actual applied security measures in ZigBee, highlight the included weaknesses and show also practical exploitations of actual product vulnerabilities. Therefore new features in the ZigBee security testing tool SecBee will be demonstrated and made public available.

78.1.13

Live Stream: https://www.youtube.com/watch?v=eEMYA-nzATM

- "GNU Radio Update" (Martin Braun)

Martin is a long-time contributor to the GNU Radio project and the GNURadio community manager. Most of the stuff he touches is SDR-related,and his day job is writing software for Ettus Research, where he's spenta lot of time on RFNoC among other things.

- "A Hands-On Introduction to SDR with GNU Radio & USRP" (Neel Pandeya)

We begin a series of hands-on introductory SDR tutorials using USRP hardware and GNU Radio software on the Linux platform. In this first instalment, we review SDR concepts, explore the USRP hardware, walk through the installation of UHD and GNU Radio on an Ubuntu system, and construct and run a few basic flowgraphs to get familiar with GNU Radio.

Bring your laptop and SDR to follow along!

First installment of a new tutorial series!

In an effort to cater to all skill levels, we'll be running tutorials so everyone can get up to speed with the basics, and new tips and tricks. The goal is for everyone to enjoy SDR regardless of experience/hardware/software/etc...

- "Interrogating Passive, Wireless SAW RFID Sensors with the USRP" (James ?Trip? Humphries)

Passive, wireless sensor design typically dictates many strict performance requirements for the sensor interrogation system in terms of bandwidth, output power, and data capture rate. In the past, this implied that a custom interrogator design would need to be implemented, requiring considerable time and effort as well as being unable to adapt to new sensor requirements. This proves to be particularly challenging in a research environment where sensor specifications may change rapidly. Recent advances in commercial-off-the-shelf (COTS) software defined radio (SDR) platforms have enabled rapid interrogator development while being able to meet the strict requirements of passive, wireless RFID sensor tags.At the University of Central Florida, we have utilized the universal software radio peripheral (USRP) B200, from Ettus Research, to implement a pulsed interrogation system for wideband, wireless surface acoustic wave (SAW) RFID sensors. In this talk, we will discuss the implemented SDR system with consideration given to FPGA modifications, external RF component integration, and post-processing. The system operates at 915MHz with 56MHz bandwidth and has output power of greater than +20dBm. A demonstration of the system will also be given with wireless SAW temperature sensors.

78.1.14

Live Stream: https://www.youtube.com/watch?v=qxPv2bSli6o

- Paul David: "gr-minecraft"

Paul will show off a Minecraft hack where one can stand up Lua "computers" within the game and talk to the outside world through TCP or HTTP. This was used to create a Minecraft frequency display linked to GNU Radio where periodic magnitude values are sent from GNU Radio and reflected on a redstone frequency display within Minecraft.

The gr-minecraft idea was inspired by a similar project to control a Wi-Fi lightbulb from within Minecraft. It's a great illustration of how easy it is to get data in and out of GNU Radio, and a testament of the endless possibilities within Minecraft itself.

- Matt Knight: "Attacking ZigBee Locks with Commodity Wireless Tools"

ZigBee and[masked] are two widely proliferated wireless protocols that are commonly associated with the Internet of Things. This talk will delve into some of the practical aspects of attacking devices that utilize these protocols. In addition to presenting a live demo of a recently developed attack on a ZigBee lock, I will discuss the drawbacks of SDR-based tools and walk through a scenario where using a hardware-defined technology was advantageous.

- Marc Newlin: "MouseJack" (special remote appearance)

Marc will join us from Atlanta to discuss the keystroke injection vulnerability he discovered that affects seven vendors' wireless mouse dongles, allowing an attacker to transmit unencrypted keystrokes into a victim's computer.

78.1.15

Live Stream: https://www.youtube.com/watch?v=-JEv2Yq_sc8

- Martin Braun: "GNU Radio Community Update"

The latest news on all things GNU Radio.

- Kevin Reid: "An Audible Waterfall Plot"

AM receivers are useful for detecting a variety of signals (and noise sources) even when they are not deliberately amplitude modulated. It is possible to implement a non-selective AM receiver, one which receives signals in a very large input bandwidth. This has been done in hardware; the classic crystal radio is an example, but more advanced designs are also made for aviation radio enthusiasts. A further refinement allows the signals to be spatialized, producing a stereo audio output where the sound is panned according to the relative RF frequency of the input, thus creating an ?audible waterfall?.

- Paul David: "Adding Dual 10 Gigabit Ethernet Capability to the USRP X300"

The USRP X300 currently supports a single 1 or 10 Gigabit Ethernet connection to the host computer. In this talk, I will discuss the various design challenges in adding support for two simultaneous 10 Gigabit links, from the perspective of the FPGA, the UHD software driver, and different Ethernet cards, as well as give a brief demonstration showing the real-time monitoring of 400 MHz of instantaneous bandwidth to view WiFi and LTE signals over-the-air.

- Balint: "Couple of interesting experiments"

GNU Radio Spot Jammer FMCW RADAR analysis

78.1.16

Live Stream: https://www.youtube.com/watch?v=mgbepw3G-uo

- "GNU Radio Tutorial Part 2" with Neel Pandeya

The tutorial series will continue! This time we will look at how to construct an FM radio receiver, and decode the RDS digital subcarrier. This will include:

- Explain concepts behind commercial FM and RDS
- Receiving mono FM using a from-scratch flowgraph
- Showing how to build 'gr-rds'
- Demonstrate stereo FM+RDS reception using 'gr-rds'
- Building GQRX
- Demonstrate FM reception using GQRX

Make sure you bring along your laptop and SDR!

- "Understanding the LTE Physical Layer" with Sandor Szilvasi (@sszilvasi)

LTE is an incredible, yet complex, cellular networking standard. Sandor will break it down and explain how a LTE signal is constructed. He will also live demo the demodulation and decoding of local carriers.

- Interactive Install & Setup-fest" with the group

We would like to open up the forum to those who wish to get set up with SDR (hardware and/or software). Bring along your equipment, and as a group we can look at/debug the steps required to get you up and running. This could also include setting up an app, or fixing an Out-Of-Tree module, or even an environment issue on your laptop.

78.1.17

Live Stream: https://www.youtube.com/watch?v=c8MW7N2RxqU

- "The Land Mobile Radio Spectrum: What is out there, how it works, and how you can hear it" with Desmond Crisis (@dcrisis)

Wireless two-way is the technology that keeps the world working in sync. I'll explore the various public safety, private enterprise, and personal communications services from[masked] MHz. We?ll discuss the occupied spectrum, modulation bandwidths, trunked radio schemes and digital transmission modes currently in use on the band as well as what lies ahead. Bring your SDR kit and play along!

- An Academic Look at Interference & Jamming
- Installfest / Hackfest / Debugfest

Bring your laptops, SDRs and signals along, and we'll try to work on getting some cool projects running on your machines!

Have a module that's not quite working? Need help with implementing some code? This is the time to get help from the group!

78.1.18

Live Stream: https://www.youtube.com/watch?v=AL1kcy4e92w

- SlackRadio: Turning your Slack channel into a radio station" with Nate Temple

Slack is a popular real-time messaging system designed for team use. I will demo a small application built with GNU Radio and the Slack API that turns your Slack channel into a real radio station for your office.

- "Pothosware" with Josh Blum

Pothosware: An open-source software stack for the SDR community including the Pothos framework for creating interconnected topologies of processing blocks, Pothos GUI for graphical designing, controlling, and visualizing topologies, and SoapySDR - a SDR abstraction layer. The talk will present and overview of the software, cover the inner workings of the framework, and demonstrations with the GUI.

- FPGA-based ADS-B SDR Receiver with Brian Padalino

Brian will discuss the design and implementation of an ADS-B receiver in the FPGA over the BladeRF.

78.1.19

Live Stream: https://www.youtube.com/watch?v=L5iw_hpKhPE

- SDR Polyamory (Jared Boone, @sharebrained)

Jared discusses non-GNU Radio approaches to doing radio signal reverse engineering, prototyping, testing, and transmitting.

- SDR is hard, but installing GNU Radio is not -- Bootstrapping your bench with PyBOMBS and CGRAN" (Martin Braun, @braun_noise)

Want to get started with GNU Radio and SDR? Let's not worry about anything and use some tools that'll get you set up nearly automatically.I'll show you how to use PyBOMBS to get a setup running on most systems without effort -- even for cross-compile setups!In the second part, we'll talk about some of the more advanced features of PyBOMBS and how YOU can use it to distribute your work.

- "Disposable, Stealthy, Cheap SIGINT" (Chris Kuethe, @kj6gve)

This presentation covers some observations and considerations for using inexpensive and compact ARM boards for signals analysis. Topics may include: power budget, air interface, attributability, performance tuning, lolcats and doges.

- "Osmocom-GMR: Quick introduction to receiving the Thuraya and Mexsat satellite phone systems" (Sylvain 'tnt' Munaut, @tnt)

GMR-1 is an ETSI standard for satellite phones heavily derived fromGSM. The main/only commercial provider using this standard used to beThuraya, mainly used in the Middle-East and Asia and didn't provideany US coverage, but recently the Mexican government deployedMorelos-3 which carries a GMR-1 3G payload and can be received fromthe US. Osmocom-GMR is a free-software project from the Osmocom family toreceive and decode those signals, going all the way from RF to packetsin wireshark or audio files. This talk will be a quick introduction to GMR itself, the project and how to get started using it.

78.1.20

Live Stream: https://www.youtube.com/watch?v=hPiUncCs6Lg

- ?L-Band WX Satellites? (Joe Steinmetz @usa_satcom)

This presentation will cover most aspects of receiving, demodulating and decoding current L-Band Weather Satellite signals (NOAA, MetOp, Meteor, FengYun, GOES). Topics will include hardware, software, de-modulation/decoding techniques, challenges, flows as well as cool sample images and data.

- "Disposable, Stealthy, Cheap SIGINT" (Chris Kuethe, @kj6gve)

This presentation covers some observations and considerations for using inexpensive and compact ARM boards for signals analysis. Topics may include: power budget, air interface, attributability, performance tuning, lolcats and doges.

79 Email

Direct email-based technical support using the `support@ettus.com` email address will be disabled on 1 January 2024.

We are transitioning to the NI Service Request Manager (SRM) to provide direct technical support.

The primary way to obtain technical support for NI/Ettus USRP hardware and software will be through NI SRM starting on 1 January 2024.

We will continue to monitor and respond to emails sent to `support@ettus.com` through 31 December 2023.

To obtain technical support through NI Service Request Manager (SRM), please visit the NI Technical Support Website. From there, you can access NI SRM and submit your service request in three steps:

Step 1: Visit the NI Technical Support Website, and scroll down to "Request Support", and click "Open A Service Request". You will be prompted to log in with your NI account. If you do not yet have an NI account, then you will need to create one, which includes providing valid serial number(s) for your NI/Ettus product(s). For more information about this, please visit Creating and Managing Your ni.com Account.

Step 2: Select the desired service (either "Request Technical Support" or "Repair"). To start an RMA for a USRP device, log in to NI SRM, and select "Repair".

Step 3: Enter your USRP model number(s) under "Supported Hardware Models", and click "Next" to proceed. Someone from the NI/Ettus technical support team will respond to your query within 24 to 48 hours (within two business days). We are often able to respond more quickly than that, depending on our current workload and backlog, holidays, and weekends.

For more details about how to open a service request, please visit Open a Service Request Case Using the NI Service Request Manager (SRM).

The NI/Ettus USRP hardware is entitled to the NI Hardware Warranty Program, which currently includes one year of standard technical support. To learn more about this program, please visit Hardware Service Programs.

NI also offers paid technical support agreement options, which provide more in-depth and customized technical support coverage. For more information about this, please submit a service request asking for further details.

There are also several Mailing Lists which provide a free and informative channel for getting technical support from NI / Ettus Research engineers as well as from many experts in the SDR community.

80 Mailing Lists

80.1

There are several mailing lists which provide a free and informative channel for getting technical support from NI / Ettus Research engineers as well as from many experts in the SDR community.

80.1.1

The focus of the usrp-users mailing list is for questions and discussions about the NI/Ettus USRP hardware as well as the UHD and RFNoC software.

The archives for the usrp-users mailing list can be found here.

80.1.2

The focus of the discuss-gnuradio mailing list is for questions and discussions about the GNU Radio software in general, whether being used with USRP devices or not.

The archives for the discuss-gnuradio can be found here.

More information about the discuss-gnuradio mailing list is available here.

80.1.3

The focus of the srsran-users mailing list is for questions and discussions about the srsRAN software stack for 4G/LTE and for 5G/NR from the srsRAN Project from Software Radio Systems (SRS).

In addition to the srsran-users mailing list, there is also a GitHub Discussions forum for the srsRAN software stack here.

The archives for the srsran-users mailing list can be found here.

80.1.4

The focus of the openair5g-user mailing list is for questions and discussions from users about the OpenAirInterface (OAI) software stack from Eurecom.

Additional information about the OAI mailing lists can be found here and here.

The archives for the openair5g-user mailing list can be found here.

80.1.5

The focus of the openair5g-devel mailing list is for questions and discussions from *developers (not from users)* of the OpenAirInterface (OAI) software stack from Eurecom.

Additional information about the OAI mailing lists can be found here and here.

The archives for the openair5g-devel mailing list can be found here.

80.1.6

The focus of the openair5g-nr mailing list is for questions and discussions from *developers (not from users)* of the OpenAirInterface (OAI) 5G/NR software stack from Eurecom.

Additional information about the OAI mailing lists can be found here and here.

The archives for the openair5g-nr mailing list can be found here.

81 Matrix

81.1

The GNU Radio Project maintains a free public Matrix chat server where the community can have real-time discussions and conversations. This provides a way to live conversations with people from Ettus Research as well as with people in the broader community. Matrix is an open network for secure, decentralized communication that is free and open to everyone. You can access the GNU Radio Matrix chat server from a desktop application, from a web browser such as Firefox, Chrome, Safari, or Edge, and also from the Element app on your Android and Apple iOS smartphone.

Please note that the primary methods of technical support for NI/Ettus USRP hardware and software are NI SRM and the various Mailing Lists, but the GNU Radio Matrix Chat Server provides a different and alternative way to obtain support from both NI/Ettus engineers and from the SDR community.

- GNU Radio Matrix Chat Server (https://chat.gnuradio.org/)
- About the GNU Radio Matrix Chat Server (https://wiki.gnuradio.org/index.php/Chat)
- Matrix Homepage (https://matrix.org/)
- What is Matrix? (https://matrix.org/docs/guides/introduction)
- Matrix Wikipedia page (https://en.wikipedia.org/wiki/Matrix_(protocol))
- Matrix Clients (https://matrix.org/clients/)
- Element Clients (https://element.io/download)
- Element App for Android (https://play.google.com/store/apps/details?id=im.vector.app)
- Element App for iPhone (https://apps.apple.com/us/app/element-messenger/id1083446067)

82 SDR Boston Slack

82.1

The Boston Software Defined Radio User Group, or SDR-Boston, maintains a free public Slack chat server where the community can have real-time discussions and conversations. This provides a way to live conversations with people from Ettus Research as well as with people in the broader community.

SDR-Boston was founded in 2010 with the mission to facilitate the exchange of ideas and to enable greater collaboration within the SDR community via the hosting of technical workshops and gatherings.

The New England Workshop on Software-Defined Radio (NEWSDR) is an event that is run every May/June, and is organized by the SDR-Boston group.

Please note that the primary methods of technical support for NI/Ettus USRP hardware and software are NI SRM and the various Mailing Lists, but the SDR-Boston Slack chat server provides a different and alternative way to obtain support from both NI/Ettus engineers and from the SDR community.

- SDR-Boston Homepage (http://sdr-boston.org/) (https://newsdr.org/)
- SDR-Boston Slack Chat Server (https://newsdr.org/ways-to-connect/)
- The annual NEWSDR event (https://newsdr.org/workshops/)
- Sign up for the SDR-Boston Slack Chat Server (https://join.slack.com/t/sdr-boston/shared_invite/enQtOTQ5MTc1MTIwOTAxLWQ0NmQ4M2ZjNTExMjFhYWE4OTUxYjFiYTI3MzJhYmRjNmNlMjgyZ

83 StackExchange

83.1

StackExchange provides a community-based support forum where questions about USRP, UHD, GNU Radio, and Software Defined Radio (SDR) in general, can be discussed. StackExchange complements the Mailing List, as it provides a web-based forum-style interface to a community which includes many technical experts.

Please note that the primary methods of technical support for NI/Ettus USRP hardware and software are NI SRM and the various Mailing Lists, but StackExchange / StackOverflow provides a different and alternative way to obtain support from both NI/Ettus engineers and from the SDR community.

- StackOverflow Homepage (https://stackoverflow.com/)
- Listing of the StackExchange communities (https://stackexchange.com/sites)
- DSP StackExchange Community (https://dsp.stackexchange.com/)
- Ham Radio StackExchange Community (https://ham.stackexchange.com/)
- List of posts on StackOverflow with the "uhd" tag (http://stackoverflow.com/questions/tagged/uhd)
- List of posts on StackOverflow with the "usrp" tag (http://stackoverflow.com/questions/tagged/usrp)
- List of posts on StackOverflow with the "software defined radio" tag (http://stackoverflow.com/questions/tagged/software-defined-radio)
- List of posts on StackOverflow with the "gnuradio" tag (https://stackoverflow.com/questions/tagged/gnuradio)
- List of posts on StackOverflow with the "gnuradio companion" tag (https://stackoverflow.com/questions/tagged/gnuradio-companion)
- Wikipedia Article for StackExchange (https://en.wikipedia.org/wiki/Stack_Exchange)

84 NI SRM

84.1

Starting on 1 January 2024, the primary way to obtain technical support for USRP devices will be through the NI Service Request Manager (SRM).

To obtain technical support through NI Service Request Manager (SRM), please visit the NI Technical Support Website. From there, you can access NI SRM and submit your service request in three steps:

Step 1: Visit the NI Technical Support Website, and scroll down to "Request Support", and click "Open A Service Request". You will be prompted to log in with your NI account. If you do not yet have an NI account, then you will need to create one, which includes providing valid serial number(s) for your NI/Ettus product(s). For more information about this, please visit Creating and Managing Your ni.com Account.

Step 2: Select the desired service (either "Request Technical Support" or "Repair"). To start an RMA for a USRP device, log in to NI SRM, and select "Repair".

Step 3: Enter your USRP model number(s) under "Supported Hardware Models", and click "Next" to proceed. Someone from the NI/Ettus technical support team will respond to your query within 24 to 48 hours (within two business days). We are often able to respond more quickly than that, depending on our current workload and backlog, holidays, and weekends.

For more details about how to open a service request, please visit Open a Service Request Case Using the NI Service Request Manager (SRM).

The NI/Ettus USRP hardware is entitled to the NI Hardware Warranty Program, which currently includes one year of standard technical support. To learn more about this program, please visit Hardware Service Programs.

There are also paid technical support and consulting options available, which provide more in-depth and customized technical support coverage. For more information about this, please submit a service request asking for further details.

There are also several Mailing Lists which provide a free and informative channel for getting technical support from NI / Ettus Research engineers as well as from many experts in the SDR community.

85 Technical FAQ

85.1

For the UHD Getting Started documentation please refer to the UHD Manual.

For the GNU Radio Getting Started documentation please refer to the GNU Radio wiki.

85.2

Yes, we have antennas for nearly every daughterboard which we sell. Please see the ordering page for pictures and specifications at the following links:

- https://www.ettus.com/product/category/Antennas
- https://kb.ettus.com/Antennas

85.3

The UHD (USRP Hardware Driver) supports all Ettus Research hardware on the following operating systems:

- Linux (any distribution should work, but primarily Ubuntu, Fedora, and CentOS, and specifically the current and prior versions)
- Mac OS X 10.4 or newer, macOS 11 or newer
- Microsoft Windows 7, 8, 10

Primary development for UHD is done on Linux.

85.4

Yes. Schematics for select USRP devices and daughterboards are available. You can find them here:

https://files.ettus.com/schematics/

85.5

The USRP hardware is sold as test equipment. If you choose to use your USRP hardware and daughterboards to transmit using an antenna, it is your responsibility to make sure your use is in compliance with any and all laws for the country, frequency, and power levels in which the device is used. Additionally, some countries regulate reception in certain frequency bands. Again, it is the responsibility of the user to maintain compliance with any and all local laws and regulations.

86 Licensing FAQ

UHD is publicly offered under the GNU General Public License version 3 (GPLv3), and is also available under a less-restrictive Alternative License. For more details, please see the Licenses page on ettus.com.

RFNoC is publicly offered under the GNU Lesser General Public License version 3 (LGPLv3).

This page contains some Frequently Asked Questions about these licenses. Please note that while we can provide guidance about our licensing model, you should make license decisions based on your own legal counsel.

86.1

86.1.1

No! You can sell products & services with GPLv3 software. You must provide the source code for your product to anyone to whom you convey your product. See the question below.

86.1.2

Yes! If you use the GPLv3-licensed version of UHD in your application, your application must then also be licensed under the GPLv3. If you want to create a proprietary product with Ettus Research HW / SW, you need to use the Alternative License, which was created specifically for this purpose.

86.1.3

That's fine! The GPL only obligates sharing source code with an external party when they receive your application.

86.1.4

No, you do not need the Alternative License. The GPLv3-licensed version of UHD is perfect for this. It allows you to make changes to UHD as you wish, as long as you provide the source code for your "fork" of UHD to recipients of your application.

86.1.5

It depends, but if you want to *guarantee* that your source code does not become available publically then way to do this is via the Alternative License. The GPLv3 does not require you to release your source code publicly. It only requires you to provide your source code to anyone to whom you convey your application. Your application must be licensed under the GPLv3, though, so your customers would have the right to share the source code more broadly -- even publically -- if they wished.

86.1.6

If you have not made changes to our source code, then you can simply point your customers to our public software repositories. If you have made changes to our source code, you must either provide your "fork" of our software to your customers or contribute your changes back to us so that they are in the mainline version of the software.

87 USRP1

87.1

Please note that this product is now End-of-Life (EOL), and is no longer available for sale through Ettus Research, and is not recommended for use in new designs or in new projects.

87.2

- 16 MHz of RF bandwidth with 8 bit samples
- 8 MHz of RF bandwidth with 16 bit samples (RX Only)
- USB 2.0 high speed connectivity
- MIMO with a single USRP device the motherboard has two daughterboard slots (2 RX + 2 TX connectors)
- FPGA: Altera Cyclone
- ADCs: 12-bits 64 MS/s
- DACs: 14-bits 128 MS/s

87.3

All Ettus Research products are individually tested before shipment. The USRP1 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP1 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

87.4

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

87.5

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

87.6

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

87.7

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

88 USRP2

88.1

Please note that this product is now End-of-Life (EOL), and is no longer available for sale through Ettus Research, and is not recommended for use in new designs or in new projects.

88.2

- 50 MHz of RF bandwidth with 8 bit samples
- 25 MHz of RF bandwidth with 16 bit samples
- Gigabit Ethernet connectivity
- MIMO capable - requires two or more USRP2 devices as motherboard has one daughterboard slot (1 RX + 1 TX connectors)
- Onboard FPGA processing
- FPGA: Xilinx Spartan XC3S2000
- ADCs: 14-bits 100 MS/s
- DACs: 16-bits 400 MS/s
- Ability to lock to external 5 or 10 MHz clock reference

88.3

88.3.1

The available resources on the FPGA will vary depending on the code written for it. Based on the 27 March 2012 FPGA code build, the following resources are available:

88.3.1.1

- General Logic
- Memory: 3% free
- DSP Resources: The FPGA does not have DSP resources

88.3.2

Not all SD Cards are compatible with the USRP2; therefore we recommend using the Kingston 2GB SD cards with part number KGS SD2GB as we have found they work reliably. For our customers' convenience, we still stock replacement SD cards available for $10. Please contact sales@ettus.com if you require a replacement SD card.

88.4

All Ettus Research products are individually tested before shipment. The USRP2 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP2 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

88.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

88.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

88.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

88.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

89 E110

89.1

Please note that this product is now End-of-Life (EOL), and is no longer available for sale through Ettus Research, and is not recommended for use in new designs or in new projects.

89.2

89.2.1

- Designed for embedded applications (runs a full distribution of Linux)
- 720 MHz OMAP3 (ARM Cortex A8 processor & TI C64x+ DSP)
- 512MB RAM
- 4GB microSD Card
- 100 Mbit Ethernet connectivity
- Motherboard has one RTX daughterboard slot (1 RX + 1 TX connectors)
- Onboard FPGA processing
- FPGA: Xilinx Spartan XC3SD1800A
- ADCs: 12-bits 64 MS/s
- DACs: 14-bits 128 MS/s
- TCXO Frequency Reference (~2.5ppm)
- Flexible clocking from 10 MHz to 64 MHz

89.2.2

- Designed for embedded applications (runs a full distribution of Linux)
- 720 MHz OMAP3 (ARM Cortex A8 processor & TI C64x+ DSP)
- 512MB RAM
- 4GB microSD Card
- 100 Mbit Ethernet connectivity
- Motherboard has one RTX daughterboard slot (1 RX + 1 TX connectors)
- Onboard FPGA processing
- FPGA: Xilinx Spartan XC3SD3400A
- ADCs: 12-bits 64 MS/s
- DACs: 14-bits 128 MS/s
- TCXO Frequency Reference (~2.5ppm)
- Flexible clocking from 10 MHz to 64 MHz

89.3

- E100/E110 - File:Ettus Embedded Series.pdf

89.4

- Utilization statistics are subject to change between UHD releases, current as of UHD 3.9.4

89.4.1

```
Device utilization summary:
---------------------------

Selected Device : 3sd3400acs484-4

 Number of Slices:                    13361  out of  23872    55%
 Number of Slice Flip Flops:          16121  out of  47744    33%
 Number of 4 input LUTs:              23350  out of  47744    48%
    Number used as logic:             20165
    Number used as Shift registers:    3185
 Number of IOs:                         195
 Number of bonded IOBs:                 176  out of    309    56%
    IOB Flip Flops:                     103
 Number of BRAMs:                        74  out of    126    58%
 Number of GCLKs:                         2  out of     24     8%
 Number of DSP48s:                       28  out of    126    22%
```

89.5

89.5.1

The default username and password for the USRP E100 and USRP E110 of the Embedded Series is:

- Username: root
- Password: usrpe

89.5.2

The available resources on the FPGA will vary depending on the code written for it. Based on the 27 March 2012 FPGA code build, the following resources are available:

89.5.2.1

- General Logic:
  - Flip Flops: 69% free
  - LUTs: 78% free
- Memory: 50% free
- DSP Resources: 78% free

89.5.2.2

- General Logic:
  - Flip Flops: 29% free
  - LUTs: 6% free
- Memory: 10% free
- DSP Resources: 13% free

89.6

All Ettus Research products are individually tested before shipment. The USRP E100/E110 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP E100/E110 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

89.7

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

89.8

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

89.9

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

89.10

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

90 B100

90.1

Please note that this product is now End-of-Life (EOL), and is no longer available for sale through Ettus Research, and is not recommended for use in new designs or in new projects.

90.2

90.2.1

- 16 MHz of RF bandwidth with 8 bit samples
- 8 MHz of RF bandwidth with 16 bit samples
- USB 2.0 high speed connectivity
- Motherboard has one RTX daughterboard slot (1 RX + 1 TX connectors)
- Onboard FPGA processing
- FPGA: Xilinx Spartan 3A-1400 FPGA
- ADCs: 12-bits 64 MS/s
- DACs: 14-bits 128 MS/s
- Ability to lock to external 5 or 10 MHz clock reference
- TCXO Frequency Reference (~2.5ppm)
- Flexible clocking from 10 MHz to 64 MHz
- FPGA code can be changed with Xilinx® ISE® WebPACK? tools

90.3

- Utilization statistics are subject to change between UHD releases, current as of UHD 3.9.4

90.3.1

```
Device utilization summary:
---------------------------

Selected Device : 3s1400aft256-4

 Number of Slices:                    10238  out of  11264    90%
 Number of Slice Flip Flops:          12136  out of  22528    53%
 Number of 4 input LUTs:              18184  out of  22528    80%
    Number used as logic:             15003
    Number used as Shift registers:    2605
    Number used as RAMs:                576
 Number of IOs:                         150
 Number of bonded IOBs:                 148  out of    161    91%
    IOB Flip Flops:                     140
 Number of BRAMs:                        31  out of     32    96%
 Number of MULT18X18SIOs:                20  out of     32    62%
 Number of GCLKs:                         2  out of     24     8%
```

90.4

All Ettus Research products are individually tested before shipment. The USRP B100 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the USRP B100 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

90.5

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

90.6

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

90.7

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

90.8

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

91 DBSRX2

91.1

The DBSRX2 is a configurable receiver that can operate from 0.8-2.3 GHz. The DBSRX2 has a programmable 1-60 MHz channel filter that allows users to serve a wide variety of applications while providing the appropriate level of selectivity in the analog path. The DBSRX2 is MIMO capable, and can also power an external LNA or active antenna. Example applications include 902-928 MHz ISM, cellular and PCS, hydrogen and hydroxyl astronomy bands, and DECT.

Note: The maximum operating frequency specification has been reduced from 2.4 to 2.3 GHz.

91.2

All Ettus Research products are individually tested before shipment. The DBSRX2 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the DBSRX2 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

91.3

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

91.4

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

91.5

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

91.6

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

92 TVRX2

92.1

Please note that this product is now End-of-Life (EOL), and is no longer available for sale through Ettus Research, and is not recommended for use in new designs or in new projects.

92.2

The TVRX2 daughterboard includes two independent downverter chains, allowing reception in two different bands, simultaneously. The TVRX2 is ideal for receive-only applications that require access to a number of bands in HF, VHF and UHF frequency ranges. Example applications include dual band receivers for signal collection, whitespace radios, ISM band receivers and broadcast TV reception.

92.3

All Ettus Research products are individually tested before shipment. The TVRX2 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the TVRX2 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

92.4

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

92.5

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

92.6

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

92.7

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale

93 XCVR2450

93.1

Please note that this product is now End-of-Life (EOL), and is no longer available for sale through Ettus Research, and is not recommended for use in new designs or in new projects.

93.2

The XCVR2450 is a high-performance transceiver intended for operation 2.4 GHz and 5.9 GHz range. Filtering on the XCVR2450 provides exceptional selectivity and dynamic range in the intended bands of operation. The typical power output of the XCVR2450 is 100 mW. Example applications include public safety, UNII, ISM, Japanese wireless and UWB development platforms.

The XCVR2450 is a half-duplex transceiver. For full-duplex operation in the 2.4 GHZ range, see the SBX.

The XCVR2450 daughterboard has been revised to improve its specifications and is no longer compatible with the USRP1. For applications requiring an XCVR2450 daughterboard to be used exclusively with the USRP1, please purchase the XCVR2450-U1.

Not Recommended for New Designs. Please consider the CBX, SBX or UBX instead.

93.3

Frequency Range

- Low Band: 2.4-2.5 GHz
- High Band: 4.9-6.0 GHz

93.4

93.4.1

Schematics: File:xcvr2450.pdf

93.5

All Ettus Research products are individually tested before shipment. The XCVR2450 is guaranteed to be functional at the time it is received by the customer. Improper use or handling of the XCVR2450 can easily cause the device to become non-functional. Listed below are some examples of actions which can prevent damage to the unit:

- Never allow metal objects to touch the circuit board while powered.
- Always properly terminate the transmit port with an antenna or 50? load.
- Always handle the board with proper anti-static methods.
- Never allow the board to directly or indirectly come into contact with any voltage spikes.
- Never allow any water, or condensing moisture, to come into contact with the boards.
- Always use caution with FPGA, firmware, or software modifications.

Never apply more than -15 dBm of power into any RF input.

Always use at least 30dB attenuation if operating in loopback configuration

93.6

Technical support for USRP hardware is available through email only. If the product arrived in a non–functional state or you require technical assistance, please contact support@ettus.com. Please allow 24 to 48 hours for response by email, depending on holidays and weekends, although we are often able to reply more quickly than that.

We also recommend that you subscribe to the community mailing lists. The mailing lists have a responsive and knowledgeable community of hundreds of developers and technical users who are located around the world. When you join the community, you will be connected to this group of people who can help you learn about SDR and respond to your technical and specific questions. Often your question can be answered quickly on the mailing lists. Each mailing list also provides an archive of all past conversations and discussions going back many years. Your question or problem may have already been addressed before, and a relevant or helpful solution may already exist in the archive.

Discussions involving the USRP hardware and the UHD software itself are best addressed through the **u?srp–-users** ?mailing list at http://usrp-users.ettus.com.

Discussions involving the use of GNU Radio with USRP hardware and UHD software are best addressed through the **d?iscuss–-gnuradio**? mailing list at https://lists.gnu.org/mailman/listinfo/discuss–gnuradio?.

Discussions involving the use of OpenBTS® with USRP hardware and UHD software are best addressed through the **o?penbts–-discuss?** mailing list at https://lists.sourceforge.net/lists/listinfo/openbts–discuss?.?

The support page on our website is located at https://www.ettus.com/support?. The Knowledge Base is located at ?https://kb.ettus.com?.

93.7

Every country has laws governing the transmission and reception of radio signals. Users are solely responsible for insuring they use their USRP system in compliance with all applicable laws and regulations. Before attempting to transmit and/or receive on any frequency, we recommend that you determine what licenses may be required and what restrictions may apply.

93.8

If you have any non–-technical questions related to your order, then please contact us by email at orders@ettus.com?, or by phone at +1–408–610–6399 (Monday-Friday, 8 AM - 5 PM, Pacific Time). Please be sure to include your order number and the serial number of your USRP.

93.9

Terms and conditions of sale can be accessed online at the following link: http://www.ettus.com/legal/terms-and-conditions-of-sale