1. Upgrading bladeRF FX3 Firmware

Brian Glod edited this page on Aug 25, 2018 · [3 revisions](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware/_history)

This page describes how to update the firmware on the bladeRF.

Pre-built firmware is hosted [on the nuand website](http://www.nuand.com/fx3). The latest firmware is pointed to by: <http://nuand.com/fx3/bladeRF_fw_latest.img>. *Make note of any library or FPGA version dependencies provided on the FX3 firmware download page.*

Below are a few approaches for upgrading the bladeRF firmware. The first method listed here is the easiest method.

However, some firmware versions are incompatible with the current tools. If know or suspect your firmware to be earlier than version 1.5.3 or run into errors while trying to upgrade the firmware, it is recommended that you try the second approach of upgrading via the bootloader.

**"Don't Panic!"** - Unless you're loading custom firmware that incorrectly configures device pins, it's generally not possible to "brick" the device. If a failure occurs during the firmware update process, you should always be able to recover via the FX3 bootloader.

|  |
| --- |
| **Table of Contents**   * [Prerequisites](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Prerequisites) * [Normal upgrade procedure](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Normal_upgrade_procedure) * [Using the firmware upgrade included in the Windows installer](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Using_the_firmware_upgrade_included_in_the_Windows_installer) * [Upgrading using the FX3 bootloader (Recovery Method)](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Upgrading_using_the_FX3_bootloader_Recovery_Method)   + [The bladeRF-cli 'recovery' command](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-The_bladeRFcli_recovery_command)     - [Example Log](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Example_Log)   + [Using the Cypress Control Center](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Using_the_Cypress_Control_Center) * [Flashed Old Firmware on New Device](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#user-content-Flashed_Old_Firmware_on_New_Device) |

## Prerequisites

Before continuing, ensure you've...

1. Installed all libraries and utilities, per the [Getting Started Guides](https://github.com/nuand/bladerf/wiki#guides-and-how-tos).
2. Downloaded the latest FX3 image. In Linux:

$ wget http://nuand.com/fx3/bladeRF\_fw\_latest.img

## Normal upgrade procedure

The bladeRF-cli program may be used in Linux, Mac OSX, and Windows (from cmd.exe) to upgrade firmware.

1. Write the firmware to the bladeRF's SPI flash using one of the two approaches:
   1. From the command line:

$ bladeRF-cli -f bladeRF\_fw\_latest.img

* 1. From within the bladeRF-cli:

bladeRF> load fx3 bladeRF\_fw\_latest.img

1. When the write operation completes, power-cycle the device.
2. The interactive mode command, version should now reflect the new firmware version:

$ bladeRF-cli -i

bladeRF> version

bladeRF-cli version: 1.2.0

libbladeRF version: 1.4.0

Firmware version: 1.9.1

FPGA version: Unknown (FPGA not loaded)

## Using the firmware upgrade included in the Windows installer

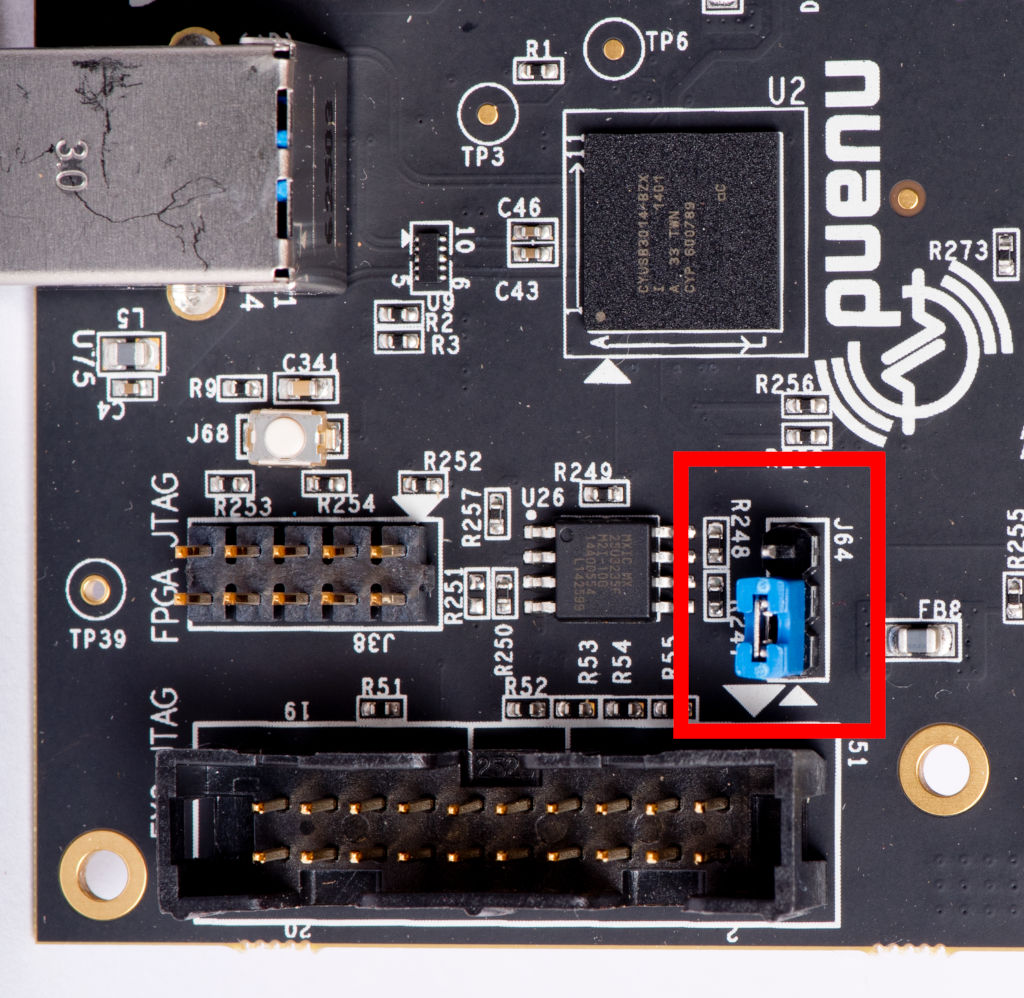
When running the Windows installer (see the [bladeRF Windows Install Guide](https://nuand.com/bladeRF-doc/guides/bladeRF_windows_installer.html)), an option to upgrade the firmware is provided. Under the hood, this runs the bladeRF-cli as described above.

Thus, it is possible to upgrade firmware by re-running the installer, but using the bladeRF-cli remains the simplest approach.

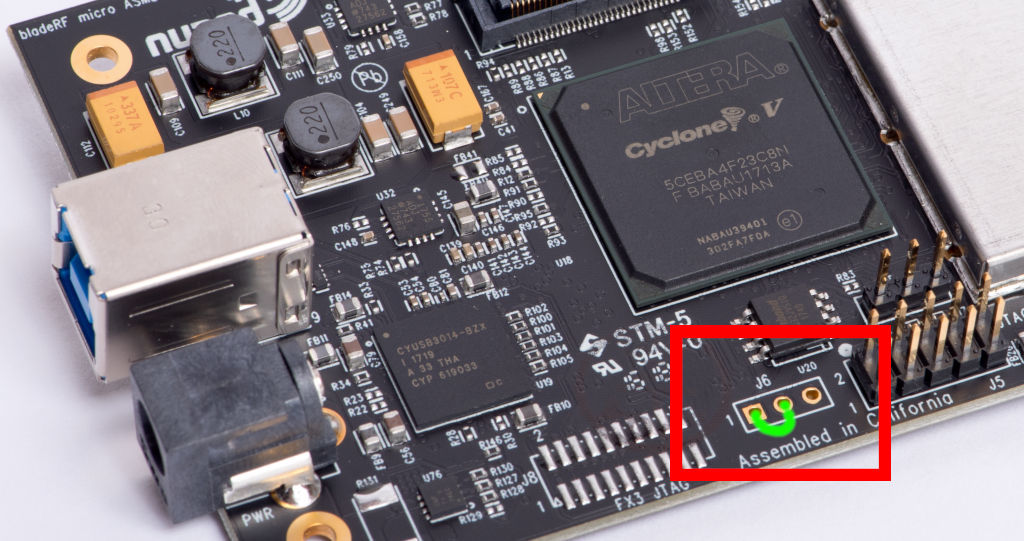
## Upgrading using the FX3 bootloader (Recovery Method)

The bladeRF is configured to fall back to a USB bootloader if valid firmware is not in SPI flash. By placing a jumper across one of the SPI communication pins, it is possible to force the SPI boot to fail and place the device into bootloader mode. From there, one can load and execute bladeRF firmware, and then follow the above procedure to write the firmware to SPI flash.

For **bladeRF Classic**, short pins 1 and 2 of the **J64** header:



For **bladeRF2‑micro**, short pins 1 and 2 of the **J6** header (may not be populated):



### The bladeRF-cli 'recovery' command

1. Unplug all bladeRF devices from your machine.
   1. If you are powering the device(s) from a DC barrel jack, ensure this is disconnected.
2. Short across pins 1 and 2 of either J6 or J64 depending on your bladeRF model, shown in the images above.
   1. This forces the loading of firmware from SPI flash to fail, which results in the device falling back to the USB bootloader.
3. Power on the board.
   1. You should see the bootloader enumerate as 04b4:00f3 Cypress Semiconductor Corp. in dmesg output, or a Cypress WestBridge device in Device Manager.
      1. **Windows Users:** Windows will automatically install the CYUSB3 driver for this bootloader. This will work with the Cypress Control Center utility shipped with the [FX3 SDK](http://www.cypress.com/?rID=57990) , as shown in [this video](https://www.youtube.com/watch?v=oolb9e_9qTc). To recover the device as shown below, you will need to use Zadig (as you did when following the Getting Started Guide) to associate the VID/PID of the Cypress bootloader with a libusb-driver (libusbK or WinUSB) before continuing on.
4. **Remove the jumper or wire that was installed earlier.**
   1. This is very important! If the jumper is not removed, SPI flash writes will not succeed in the following steps.
5. From a shell (or cmd.exe), run bladeRF-cli -i.
   1. You should now be at a *bladeRF>* prompt, with a message indicating that the bootloader detected via a string like: *NOTE: One or more FX3-based devices operating in bootloader mode were detected.*
   2. All of the following commands should be entered in the bladeRF-cli's prompt -- not from your shell.
6. Run recover with no arguments
   1. This will get the *Bus* and *Address* that you'll need for the next step.
7. Run recover <bus> <addr> <path to firmware>
   1. *bus* and *addr* are the values shown as *Bus* and *Address* above, and *path to firmware* is the path to the firmware you want to load.
   2. This command does not write the firmware to flash; it only loads it into RAM. We will write the firmware to the SPI flash in the next two steps.
8. Run open.
   1. This will attach to the first available bladeRF. Specifically, this will attach to the device that you just booted bladeRF firmware on via the *recover* command.
9. Run load fx3 <firmware>.
   1. This will will write the firmware to SPI flash.
10. Unplug and replug the device.
    1. A power cycle is required to boot the new firmware.

**Example Log**

Below is a bladeRF-cli log that shows example output for the above procedure:

jon@example % bladeRF-cli -i

NOTE: One or more FX3-based devices operating in bootloader mode

were detected. Run 'help recover' to view information about

downloading firmware to the device(s).

No device(s) attached.

bladeRF> help recover

Usage: recover [<bus> <address> <firmware file>]

Load firmware onto a device running in bootloader mode, or list all

devices currently in bootloader mode.

With no arguments, this command lists the USB bus and address for

FX3-based devices running in bootloader mode.

When provided a bus, address, and path to a firmware file, the

specified device will be loaded with and begin executing the provided

firmware.

In most cases, after successfully loading firmware into the device's

RAM, users should open the device with the "open" command, and write

the firmware to flash via "load fx3 <firmware file>"

bladeRF> recover

FX3 bootloader devices:

---------------------------------------------------------

Backend: libusb

Bus: 1

Address: 7

Use 'recover <bus> <addr> <firmware>' to download and boot

firmware to the specified device.

bladeRF> echo "Remember to remove the jumper from J64 before continuing"

Remember to remove the jumper from J64 before continuing

bladeRF> echo "If you forget to remove the jumper, the following 'open' will fail"

If you forget to remove the jumper, the following 'open' will fail

bladeRF> recover 1 7 ~/.config/Nuand/bladeRF/bladeRF\_fw\_v1.8.0.img

Success! Use "open" to switch to this device.

Note that a "load fx3 <firmware>" is required to write the firmware to flash.

bladeRF> open

bladeRF> info

Serial #: f12ce1037830a1b27f3ceeba1f521413

VCTCXO DAC calibration: 0x9f50

FPGA size: 40 KLE

FPGA loaded: yes

USB bus: 2

USB address: 8

USB speed: SuperSpeed

Backend: libusb

Instance: 0

bladeRF> version

bladeRF-cli version: 1.2.0

libbladeRF version: 1.4.0

Firmware version: 1.8.0

FPGA version: 0.3.1

bladeRF> load fx3 ~/.config/Nuand/bladeRF/bladeRF\_fw\_v1.8.0.img

Flashing firmware from /home/jon/.config/Nuand/bladeRF/bladeRF\_fw\_v1.8.0.img...

[INFO @ usb.c:498] Erasing 3 blocks starting at block 0

[INFO @ usb.c:503] Erased block 2

[INFO @ usb.c:511] Done erasing 3 blocks

[INFO @ usb.c:705] Writing 479 pages starting at page 0

[INFO @ usb.c:709] Writing page 478

[INFO @ usb.c:718] Done writing 479 pages

[INFO @ flash.c:110] Verifying 479 pages, starting at page 0

[INFO @ usb.c:603] Reading 479 pages starting at page 0

[INFO @ usb.c:606] Reading page 478

[INFO @ usb.c:617] Done reading 479 pages

Done. Cycle power on the device.

bladeRF> q

jon@example % dmesg

<snip>

[96232.958622] usb 2-1: SerialNumber: f12ce1037830a1b27f3ceeba1f521413

[96235.572759] usb 2-1: reset SuperSpeed USB device number 8 using xhci\_hcd

[96235.589353] usb 2-1: LPM exit latency is zeroed, disabling LPM.

[96269.073285] usb 2-1: USB disconnect, device number 8

[96270.697345] usb 2-1: new SuperSpeed USB device number 9 using xhci\_hcd

[96270.714153] usb 2-1: LPM exit latency is zeroed, disabling LPM.

[96270.715387] usb 2-1: New USB device found, idVendor=1d50, idProduct=6066

[96270.715395] usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3

[96270.715399] usb 2-1: Product: bladeRF

[96270.715402] usb 2-1: Manufacturer: Nuand

[96270.715406] usb 2-1: SerialNumber: f12ce1037830a1b27f3ceeba1f521413

</snip>

jon@example % bladeRF-cli -p

Backend: libusb

Serial: f12ce1037830a1b27f3ceeba1f521413

USB Bus: 2

USB Address: 9

jon@example % bladeRF-cli -i

bladeRF> info

Serial #: f12ce1037830a1b27f3ceeba1f521413

VCTCXO DAC calibration: 0x9f50

FPGA size: 40 KLE

FPGA loaded: yes

USB bus: 2

USB address: 9

USB speed: SuperSpeed

Backend: libusb

Instance: 0

bladeRF> version

bladeRF-cli version: 1.2.0

libbladeRF version: 1.4.0

Firmware version: 1.8.0

FPGA version: 0.3.1

bladeRF> q

jon@example %

**Using the Cypress Control Center**

[This video](http://youtu.be/oolb9e_9qTc) also shows a procedure similar to that outlined above, but using the Cypress Control Center (provided with the FX3 SDK) to load firmware, instead of using the bladeRF-cli.

Note that this Cypress software is only available for Windows.

## Flashed Old Firmware on New Device

While newer FX3 images (v2.2.0 and newer) will work on both bladeRF Classic and bladeRF2‑micro, it is possible to flash an older image onto a bladeRF2‑micro. These older firmware versions only know about the bladeRF Classic and simply report a USB device ID of 0x5246 to the host without first checking the underlying hardware. Newer firmware versions perform this check and report either 0x5246 for bladeRF Classic or 0x5250 for bladeRF2‑micro. Host software older than Git hash [e5fb3e9 (8/21/2018)](https://github.com/Nuand/bladeRF/commit/e5fb3e9c20368dcee85f22e3b9a5c106f1d49c83) will fail to open the device and immediately fall back to the command shell:

$ ./bladeRF-cli -i

[ERROR @ .../bladerf1.c:921] Invalid FPGA size 49.

Failed to open device (first available): An unexpected error occurred

$

This can be confirmed using the lsusb command and filtering for vendor ID 0x2cf0. If a bladeRF2‑micro is connected, and 2cf0:5246 shows up, then old firmware was flashed and it needs to be updated.

$ lsusb -d 2cf0:

Bus 002 Device 008: ID 2cf0:5246

If the firmware is even older (pre-v2.0.0), it is possible for the vendor ID to be reported as OpenMoko, 0x1d50.

$ lsusb -d 1d50:

Bus 002 Device 008: ID 1d50:6066

To correct this, it is recommended to update to the latest host software, which will report a critical warning and allow the user to correct the issue (i.e. flash updated firmware), instead of reporting an error and terminating abruptly.

$ ./bladeRF-cli -i

[CRITICAL @ .../bladerf1.c:889] Device type mismatch! FPGA size 49 is a bladeRF2 characteristic, but the USB PID indicates bladeRF1. Initialization cannot continue.

[INFO @ .../bladerf1.c:892] You must download firmware v2.2.0 or later from https://www.nuand.com/fx3/ and flash it (bladeRF-cli -f /path/to/bladeRF\_fw.img) before using this device.

[WARNING @ .../bladerf1.c:904] Skipping further initialization...

bladeRF> quit

$ ./bladeRF-cli -f bladeRF\_fw\_v2.2.0.img

<snip>

Flashing firmware...

<snip>

Done. A power cycle is required for this to take effect.

$

<power cycle>

$ ./bladeRF-cli -i -l hostedxA4-latest.rbf

Deferring device init until after FPGA load

Loading fpga...

Done.

bladeRF> version

bladeRF-cli version: 1.6.0-git-df07dd94

libbladeRF version: 2.0.0-git-df07dd94

Firmware version: 2.2.0-git-3d38fac2

FPGA version: 0.7.3

bladeRF> quit

If updating the host software is not possible, the only way to correct this is to force the FX3 into bootloader mode and update the firmware using the [recovery procedure](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-FX3-Firmware#Upgrading_using_the_FX3_bootloader_Recovery_Method).

# bladeRF CLI Tips and Tricks

Robert Ghilduta edited this page on Aug 30, 2019 · [51 revisions](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks/_history)

|  |
| --- |
| Table of Contents  * [Analyzing RX'd data](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Analyzing_RXd_data)   + [Receiving samples](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Receiving_samples)   + [Octave/MATLAB](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-OctaveMATLAB)     - [CSV data](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-CSV_data)     - [Binary Data (SC16Q11 format)](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Binary_Data_SC16Q11_format)   + [baudline](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-baudline)     - [Pre-recorded samples](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Prerecorded_samples) * [Live Viewing of Samples](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Live_Viewing_of_Samples)   + [Step 1: Create a FIFO](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Step_1_Create_a_FIFO)   + [Step 2: Start Viewer](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Step_2_Start_Viewer)     - [Baudline](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Baudline)     - [GNU Radio Flowgraph](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-GNU_Radio_Flowgraph)   + [Step 3: Start receiving to FIFO in bladeRF-cli](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Step_3_Start_receiving_to_FIFO_in_bladeRFcli) * [Transmitting Pre-generated Samples](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Transmitting_Pregenerated_Samples)   + [Example: Transmitting Samples Generated in Octave/MATLAB](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Example_Transmitting_Samples_Generated_in_OctaveMATLAB) * [Backing up and restoring calibration data](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Backing_up_and_restoring_calibration_data)   + [Backing up calibration data](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Backing_up_calibration_data)   + [Restoring calibration data](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Restoring_calibration_data)     - [From a file](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-From_a_file)     - [From scratch](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-From_scratch)   + [Wiped your calibration data? All is not lost!](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Wiped_your_calibration_data_All_is_not_lost) * [Using bladeRF-cli in shell scripts](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Using_bladeRFcli_in_shell_scripts)   + [Detecting if devices are present](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Detecting_if_devices_are_present)   + [Reading device properties](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Reading_device_properties)   + [Transmitting or receiving samples](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Transmitting_or_receiving_samples) * [Manipulating Expansion I/O Pins](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Manipulating_Expansion_IO_Pins)   + [Usage Information](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Usage_Information)   + [Printing State](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Printing_State)   + [Examples](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Examples) * [Getting the most out of libtecla](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Getting_the_most_out_of_libtecla) * [Using an External Clock Input](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Using_an_External_Clock_Input)   + [bladeRF Classic](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-bladeRF_Classic)   + [bladeRF2-micro](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-bladeRF2micro) * [Using the External Clock Output](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Using_the_External_Clock_Output)   + [bladeRF Classic](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-bladeRF_Classic-2)   + [bladeRF2-micro](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-bladeRF2micro-2) * [The 'print hardware' Command Group](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-The_print_hardware_Command_Group) * [Open a Device by Serial Number](https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#user-content-Open_a_Device_by_Serial_Number) |

## Analyzing RX'd data

Samples received via the bladeRF-cli, in either CSV or binary format, can easily be imported into tools such as [Octave](https://www.gnu.org/software/octave/), [MATLAB](http://www.mathworks.com/products/matlab/) or [baudline](http://www.baudline.com/).

## Receiving samples

The first step is to open the device and configure it using the bladeRF-cli. The below example configures the device to tune to 1.575.42 MHz, with an 8MHz sample rate and 5 MHz bandwidth. Change these and the gain values as needed.

$ bladeRF-cli -i

bladeRF> set frequency rx 1575.42M

Set RX frequency: 1575420000Hz

bladeRF> set samplerate rx 8M

Setting RX sample rate - req: 8000000 0/1Hz, actual: 8000000 0/1Hz

bladeRF> set bandwidth rx 5M

Set RX bandwidth - req: 5000000 Hz actual: 5000000 Hz

# Automatic gain control is enabled by default with libbladeRF 2.0.0 and later.

# If you wish to disable it and set gains manually:

#

# bladeRF> set agc off

#

# Before libbladeRF 2.0.0, 'set lnagain', 'set rxvga1', and 'set rxvga2' were

# used to individually set the gain stages on the bladeRF x40/x115. With

# libbladeRF 2.0.0 and later, on any bladeRF, 'set gain rx 60' will set maximum

# gain, and values less than 60 will be less than that:

#

# bladeRF> set gain rx 60

bladeRF> print

RX Bandwidth: 5000000 Hz

TX Bandwidth: 28000000 Hz

RX Frequency: 1575420000 Hz

TX Frequency: 1000000000 Hz

GPIO: 0x00000037

LMS Enable: Enabled

LMS RX Enable: Enabled

LMS TX Enable: Enabled

TX Band: Low Band (300M - 1.5GHz)

RX Band: High Band (1.5GHz - 3.8GHz)

RX Source: LMS6002D

Loopback mode: none

RXLNA Gain: 6 dB

RXVGA1 Gain: 30 dB

RXVGA2 Gain: 9 dB

TXVGA1 Gain: -14 dB

TXVGA2 Gain: 0 dB

Sampling: Internal

RX sample rate: 8000000 0/1

TX sample rate: 12000000 0/1

Current VCTCXO trim: 0x9130

Stored VCTCXO trim: 0x9130

Expansion GPIO: 0xf9ff391f

Expansion GPIO direction: 0x00000000 (1=output, 0=input)

Next, we receive samples and write them to a file.

When writing large number of samples, especially at higher sample rates, you should use the binary format to save on disk space. Additionally, consider saving samples to a ramdisk or SSD, rather than an HDD.

Below is an example of receiving (20 \* 1024 \* 1024) samples, using the binary "SC16 Q11" format:

bladeRF> rx config file=my\_samples.sc16q11 format=bin n=20M

bladeRF> rx start;

bladeRF> rx

State: Running

Last error: None

File: my\_samples.sc16q11

File format: SC16 Q11, Binary

# Samples: 20971520

# Buffers: 32

# Samples per buffer: 32768

# Transfers: 16

Timeout (ms): 1000

bladeRF> rx wait

bladeRF> rx

State: Idle

Last error: None

File: my\_samples.sc16q11

File format: SC16 Q11, Binary

# Samples: 20971520

# Buffers: 32

# Samples per buffer: 32768

# Transfers: 16

Timeout (ms): 1000

If you'd like to save samples to a CSV, you can change the format option:

bladeRF> rx config file=my\_samples.csv format=csv n=20M

bladeRF> rx start

bladeRF> rx

State: Running

Last error: None

File: my\_samples.csv

File format: SC16 Q11, CSV

# Samples: 20971520

# Buffers: 32

# Samples per buffer: 32768

# Transfers: 16

Timeout (ms): 1000

bladeRF> rx wait

bladeRF> rx

State: Idle

Last error: None

File: my\_samples.csv

File format: SC16 Q11, CSV

# Samples: 20971520

# Buffers: 32

# Samples per buffer: 32768

# Transfers: 16

Timeout (ms): 1000

## Octave/MATLAB

Routines for loading/saving samples from/to various formats are provided [here in the bladeRF git repository](https://github.com/Nuand/bladeRF/tree/master/host/misc/matlab).

The below examples show how to plot the magnitude of samples received in the above example.

### CSV data

In the CSV format, samples are stored in rows of: I, Q.

To load these samples and reconstruct the complex signal:

% We must first tell MATLAB/Octave where to find our functions

% by pointing it to the directory containing the conversion routines.

% You will need to change this path as needed.

addpath('~/projects/bladeRF/host/misc/matlab/')

% Load the samples

signal = load\_csv('my\_samples.csv');

% Plot the magnitude of the signal in the time domain

plot(abs(signal));

### Binary Data (SC16Q11 format)

Binary data in the SC16Q11 format is little-endian, sign-extended, and right-aligned. To load the samples and reconstruct the complex signal:

% We must first tell MATLAB/Octave where to find our functions

% by pointing it to the directory containing the conversion routines.

% You will need to change this path as needed.

addpath('~/projects/bladeRF/host/misc/matlab/')

% Load the samples

signal = load\_sc16q11('my\_samples.sc16q11');

% Plot the magnitude of the signal in the time domain

plot(abs(signal));

## baudline

Samples must be in a binary format to be used with baudline. This section assumes you've saved them to the binary SC16Q11 format, but other conversions are certainly possible.

### Pre-recorded samples

Here's one way to view binary samples you've saved off using the bladeRF-cli program. Note that you can open a file using these settings from the command-line as well.

1. Open baudline
2. Right-click and select **Input -> Open File**
3. Change **File Format** to **raw**
4. Select your file and click **Open**
5. Set the following parameters in the **raw parameters** dialog:
   1. Decompression: **OFF**
   2. Initial byte offset: **0**
   3. Sample Rate: *Sample rate you recorded the samples at*
   4. Channel: **2**, **quadrature**
   5. Decode Format: **16 bit linear**, **little endian**

## Live Viewing of Samples

When getting your bearings in the bladeRF-cli, it's handy to be able to visualize data while you're dialing in settings. Having external program show the samples RX'd by the bladeRF-cli comes in quite handy.

## Step 1: Create a FIFO

$ mkfifo /tmp/rx\_samples.bin

## Step 2: Start Viewer

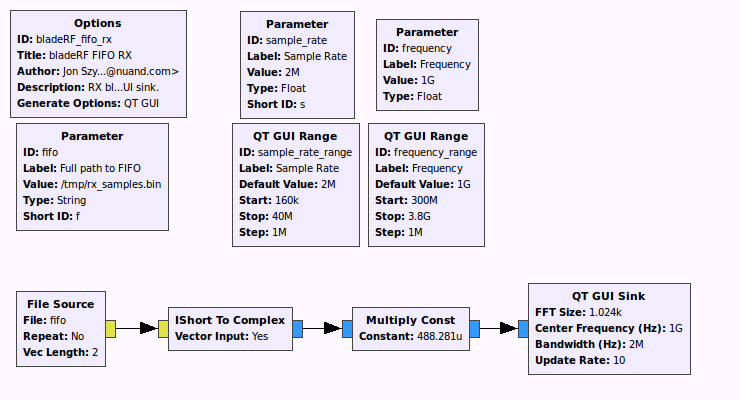
Be sure to adjust samplerate and frequency parameters appropriately to ensure the display matches your hardware settings.

### Baudline

./baudline -reset -format le16 -channels 2 -quadrature -samplerate 2000000 -stdin < /tmp/rx\_samples.bin

### GNU Radio Flowgraph

[This flowgraph](https://www.nuand.com/gnuradio/grc/bladeRF_fifo_rx.grc) ([python](https://www.nuand.com/~jon/gnuradio/grc/bladeRF_fifo_rx.py)) converts bladeRF SC16 Q11 samples to GNU radio complex values and displays them with a Qt GUI sink. Note that the same conversion approach can be taken for transmitting.



./bladeRF\_fifo\_rx.py --sample-rate=2e6 --frequency=433.92e6

## Step 3: Start receiving to FIFO in bladeRF-cli

bladeRF> set frequency 433.92M

bladeRF> set samplerate 2M

bladeRF> set bandwidth 1.5M

bladeRF> rx config file=/tmp/rx\_samples.bin n=0

bladeRF> rx start

bladeRF> # Adjust parameters while viewing the spectrum in baudline!

bladeRF> rx stop

## Transmitting Pre-generated Samples

The bladeRF-cli provides the ability to transmit samples from either a binary or CSV SC16 Q11 file. It is generally very helpful during development/debugging stages to generate and write samples to a file, using one's language of choice, and then transmit them via the bladeRF-cli.

Run help tx from the CLI for a description of available parameters and other important notes.

## Example: Transmitting Samples Generated in Octave/MATLAB

This example shows how to use Octave or MATLAB to create samples for a simple sinusoids at +250 Khz, sampled at 2 Msps, and then transmit these sample with the bladeRF-cli. The same procedure here can be followed for your language of choice, provided you implement a few simple conversion routines.

Octave/MATLAB functions for loading/saving samples from/to the bladeRF SC16 Q11 binary and CSV formats are provided [here in the bladeRF repository](https://github.com/Nuand/bladeRF/tree/master/host/misc/matlab).

You can add these functions to your Octave/MATLAB path as follows:

>> addpath('~/projects/bladeRF/host/misc/matlab/');

help save\_sc16q11

save\_sc16q11 Write a normalized complex signal to a binary file in the

bladeRF "SC16 Q11" format.

[RET] = save\_sc16q11(FILENAME, SIGNAL)

RET is 1 on success, and 0 on failure.

FILENAME is the target filename. The file will be overwritten if it

already exists. The file data is written in little-endian format.

SIGNAL is a complex signal with the real and imaginary components

within the range [-1.0, 1.0).

Next, we create the signal:

% Our samples must be generated at the samplerate we plan to run the device at

SAMPLE\_RATE = 2e6;

% Generate 10 seconds worth of samples. Bear in mind that when using the

% binary SC16 Q11 format, 1 sample consumes 4 bytes of memory/disk space.

% This quickly adds up - 10 seconds @ 2 Msps yields ~ 76.3 MiB. Be careful

% when using higher sample rates!

NUM\_SECONDS = 10;

NUM\_SAMPLES = NUM\_SECONDS \* SAMPLE\_RATE;

% 250 kHz, in radians (ω = F\_Hz \* 2pi)

SIGNAL\_FREQ\_RAD = 250e3 \* 2 \* pi;

% Generate a vector "t" which represents time, in units of samples.

% This starts at t=0, and creates NUM\_SAMPLES samples in steps of 1/SAMPLE\_RATE

t = [ 0 : (1/SAMPLE\_RATE) : (NUM\_SECONDS - 1/SAMPLE\_RATE) ];

% Create a sinusoid (signal = e^(jωt) ) with a magnitude of 0.90

signal = 0.90 \* exp(1j \* SIGNAL\_FREQ\_RAD \* t);

% Plot the FFT of our signal as a quick sanity check.

% The NUM\_SAMPLES denominator is just to normalize this for display purposes.

f = linspace(-0.5 \* SAMPLE\_RATE, 0.5 \* SAMPLE\_RATE, length(signal));

plot(f, 20\*log10(abs(fftshift(fft(signal)))/NUM\_SAMPLES));

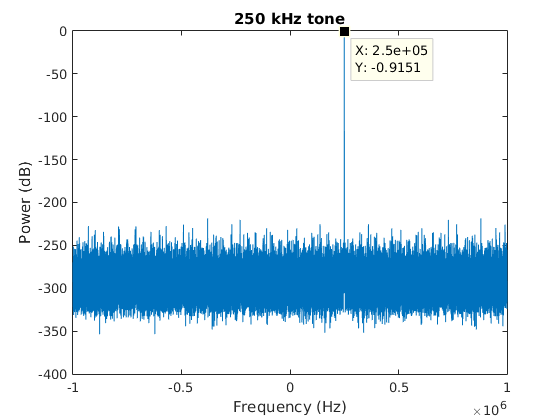
xlabel('Frequency (Hz)');

ylabel('Power (dB)');

title('250 kHz tone');

% Save the signal to a file

save\_sc16q11('/tmp/samples.sc16q11', signal);

Here's an example of the expected plot: 

It's generally recommended to save files to /tmp or a ramdisk location, rather than to a hard disk. For higher sample rates, reading samples from disk may cause underruns.

Next, we'll enter the bladeRF command line interface:

$ bladeRF-cli -i

bladeRF>

First, we configure the device. Ensure the sample rate matches the rate the rate you created your samples with!

Note that the TXVGA1 and TXVGA2 gain values are dependent upon you're operating environment. You should start with setting these to low (e.g., TXVGA=1-20, TXVGA2=0) values if you're connecting directly to test equipment.

bladeRF> set frequency tx 925M

Set TX frequency: 925000000Hz

bladeRF> set samplerate tx 2M

Setting TX sample rate - req: 2000000 0/1Hz, actual: 2000000 0/1Hz

bladeRF> set bandwidth tx 1.5M

Set TX bandwidth - req: 1500000Hz actual: 1500000Hz

bladeRF> set txvga1 -10

bladeRF> set txvga2 0

bladeRF> set bandwidth tx 1.5M

Set TX bandwidth - req: 1500000Hz actual: 1500000Hz

bladeRF> set txvga1 -10

bladeRF> set txvga2 0

Next, we configure the CLI to transmit out samples:

bladeRF> tx config file=/tmp/samples.sc16q11 format=bin

You can also repeat the transmission of a file. For example, if you'd like to repeat the file the file 3 times, with a 1 second delay between each repetition:

bladeRF> tx config repeat=3 delay=1000000

Now we'll transmit the file, and observe that it is running:

bladeRF> tx start

bladeRF> tx

State: Running

Last error: None

File: /tmp/samples.sc16q11

File format: SC16 Q11, Binary

Repetitions: 3

Repetition delay: 1000000 us

# Buffers: 32

# Samples per buffer: 32768

# Transfers: 16

Timeout (ms): 1000

We can wait for the transmission to complete, via the tx wait command. This command will block until either the transmission completes or a specified timeout occurs. You can also interrupt this command with Ctrl-C.

bladeRF> tx wait

bladeRF> tx

State: Idle

Last error: None

File: /tmp/samples.sc16q11

File format: SC16 Q11, Binary

Repetitions: 3

Repetition delay: 1000000 us

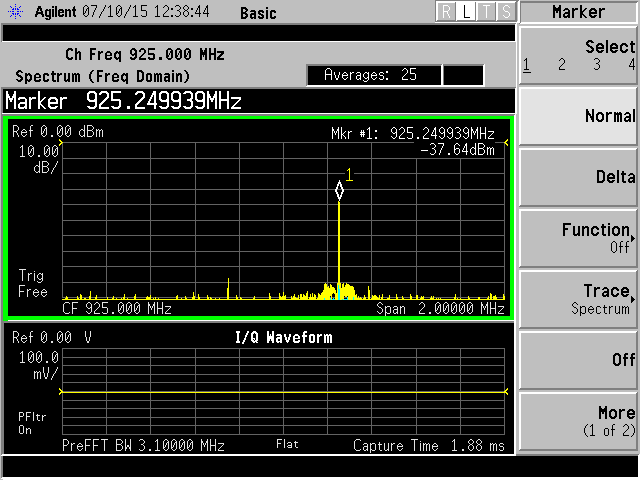
# Buffers: 32

# Samples per buffer: 32768

# Transfers: 16

Timeout (ms): 1000

The following screenshot from a VSA shows this tone being transmitted at the desired 250 kHz offset from the center frequency (925MHz). Note that this is from a device with its DC offset and IQ imbalance compensated for. See [wiki page](https://github.com/Nuand/bladeRF/wiki/DC-offset-and-IQ-Imbalance-Correction#correcting-the-tx-module) describing correction features if you see significant LO leakage at the center frequency, or a significant image at -250 kHz.



## Backing up and restoring calibration data

Currently, the following information is stored in a user-modifiable "calibration region":

* FPGA size (40 or 115)
* [VCTCXO](http://www.abracon.com/Oscillators/ASTX12_ASVTX12.pdf) (Voltage-controlled temperature-compensated oscillator) trim DAC value

Unless you're planning to replace the FPGA (**not** a recommended endeavor), there's no reason to mess with this field. The reason the FPGA size is here rather than in the OTP region is largely historic and pertains to debug usage during early development and testing.

The VCTCXO frequency will drift over time and due to environmental factors. Users will generally want to periodically recalibrate the associated trim DAC value, and write it back to the bladeRF.

## Backing up calibration data

From with the bladeRF-cli's interactive mode, the current calibration values can be viewed via the *info* command:

bladeRF> info

Serial #: 00000000000000000000000000000000

VCTCXO DAC calibration: 0xa0a8

FPGA size: 40 KLE

FPGA loaded: no

USB bus: 4

USB address: 2

USB speed: SuperSpeed

Backend: libusb

Instance: 0

Note the VCTXCO (trim) DAC calibration value above.

To back up this information to a file, which may later be restored through the bladeRF-cli program, the **flash\_backup** command may be used. Type help flash\_backup to see the full help text for this command.

The general usage for saving calibration data is flash\_backup <output\_file> cal:

bladeRF> flash\_backup /home/jon/bladeRF-files/bladerf\_cal\_2014\_03\_18.bin cal

[INFO] Reading 0x00000100 bytes from address 0x00030000.

This command will store the calibration data, along with some additional metadata, in the specified file. To view the associated metadata stored in the flash image, use the **flash\_image** command. This metadata may prove useful if you've accidentally renamed a file, or forgot which device (via serial number) is associated with the file.

bladeRF> flash\_image /home/jon/bladeRF-files/bladerf\_cal\_2014\_03\_18.bin

Checksum: d3ad936733b841cd10f0cb852d2a133e08cf080a60c4a607cca9908b75bd5e59

Image format version: 0.1.0

Timestamp: 2014-03-18 20:25:37

Serial #: 00000000000000000000000000000000

Image type: Calibration data

Address: 0x00030000

Length: 0x00000100

## Restoring calibration data

### From a file

If you've backed up calibration data to a file via the **flash\_backup** command, you can use the **flash\_restore** command to write this data back to the device. Note that the serial number information in the metadata is strictly for informational purposes; it **is not** used to prevent data from being written to another device.

For more information about the **flash\_restore** command, run help flash\_restore.

bladeRF> flash\_restore /home/jon/bladeRF-files/bladerf\_cal\_2014\_03\_18.bin

[INFO] Reading 0x00010000 bytes from address 0x00030000.

[INFO] Erasing 0x00010000 bytes starting at address 0x00030000.

[INFO] Writing 0x00010000 bytes to address 0x00030000.

**Note:** A power cycle will be required for this change to take effect.

### From scratch

This approach useful if you'd like to manually specify a VCTCXO trim value to store in the device.

The **flash\_init\_cal** command generates a calibration region of flash, and can either write it directly to a device, or to a file (for future use with the *flash\_restore* command).

For more information about this command, run help flash\_init\_cal from the bladeRF-cli interactive mode.

For a bladeRF with a 40 kLE FPGA and a desired VCTCXO trim DAC value of 0x9015, you can write this information directly to the device via:

bladeRF> flash\_init\_cal 40 0x9015

[INFO] Reading 0x00010000 bytes from address 0x00030000.

[INFO] Erasing 0x00010000 bytes starting at address 0x00030000.

[INFO] Writing 0x00010000 bytes to address 0x00030000.

**Note:** A power cycle will be required for this change to take effect.

Alternatively, to write this data to a file:

bladeRF> flash\_init\_cal 40 0x9015 /tmp/new\_cal\_data.bin

## Wiped your calibration data? All is not lost!

If you've managed to wipe your calibration region and do not have the necessary tools to identify an appropriate VCTCXO trim value, you may look up that factory default on [this page](https://www.nuand.com/calibration.php).

## Using bladeRF-cli in shell scripts

## Detecting if devices are present

bladeRF-cli -p returns 0 if one or more devices are detected, and a non-zero value if no devices are connected. Below is a snippet you can use to exit a script if no bladeRF devices are detected:

#!/bin/sh

bladeRF-cli -p 1>/dev/null 2>&1

if [ $? -ne 0 ]; then

echo "No bladeRF devices connected." >&2

exit 1

fi

## Reading device properties

The bladeRF-cli info command can be used to fetch a device's serial number or FPGA size, as shown below:

#!/bin/sh

# Fetch device information

DEVICE\_INFO=$(bladeRF-cli -e 'info' 2>/dev/null)

if [ -z "$DEVICE\_INFO" ]; then

echo "Failed to open device and query info." >&2

exit 1

fi

# Read a device's serial number

DEVICE\_SERIAL=$(echo "$DEVICE\_INFO" | grep 'Serial #:' | sed -e 's/.\*Serial #:\s\+//')

# Check if the device has an FPGA loaded

DEVICE\_FPGA\_LOADED=$(echo "$DEVICE\_INFO" | grep "FPGA loaded:" | sed -e 's/.\*FPGA loaded:\s\+//')

# Read the device's FPGA size

DEVICE\_FPGA\_SIZE=$(echo "$DEVICE\_INFO" | grep "FPGA size:" | sed -e 's/.\*FPGA size:\s\+//')

echo ""

echo "Found bladeRF"

echo " Serial #: $DEVICE\_SERIAL"

echo " FPGA type: $DEVICE\_FPGA\_SIZE"

echo " FPGA loaded: $DEVICE\_FPGA\_LOADED"

echo ""

## Transmitting or receiving samples

When transmitting or receiving from a shell script, one must ensure rx wait or tx wait is used to ensure the CLI is not exited prior to the completing the desired reception/transmission.

*Example: RX 10 seconds of data*

bladeRF-cli -e 'rx config file=/tmp/rx\_10s.bin n=0; rx start; rx; rx wait 10s'

*Example: Transmit a file to completion*

bladeRF-cli -e 'tx config file=/tmp/tx\_samples.bin; tx start; tx; tx wait'

## Manipulating Expansion I/O Pins

32 Pins are available for general use on the bladeRF's U74 expansion header. A subset of these pins are exposed for general use on the XB-200 Transverter and XB-100 Expansion boards.

Be careful with interacting with these pins -- they are directly connected to the 1.8V I/O pins on the FPGA. (Although some pins on the XB-200 are connected via a level translator.) Ensure you review the schematics and use appropriate buffers or level converters when attaching devices to these pins!

Support for manipulating these pins from a higher-level interface was introduced on September 1st, 2015. You'll need to be using a version of libbladeRF, the bladeRF-cli, and an FPGA image (>= v0.4.1) from [this commit](https://github.com/Nuand/bladeRF/commit/94fed3d0cde61157097ff05f77cfc07a1c0ac1ef) or later.

## Usage Information

First, run the following commands to view the help and usage information for each.

bladeRF> print xb\_gpio

bladeRF> print xb\_gpio\_dir

bladeRF> set xb\_gpio

bladeRF> set xb\_gpio\_dir

## Printing State

As described above, we can print the state of expansion I/O pins:

bladeRF> print xb\_gpio\_dir all

Expansion GPIO direction register: 0x00000000

GPIO\_1: input

GPIO\_2: input

GPIO\_3: input

GPIO\_4: input

GPIO\_5: input

GPIO\_6: input

GPIO\_7: input

GPIO\_8: input

GPIO\_9: input

GPIO\_10: input

GPIO\_11: input

GPIO\_12: input

GPIO\_13: input

GPIO\_14: input

GPIO\_15: input

GPIO\_16: input

GPIO\_17: input

GPIO\_18: input

GPIO\_19: input

GPIO\_20: input

GPIO\_21: input

GPIO\_22: input

GPIO\_23: input

GPIO\_24: input

GPIO\_25: input

GPIO\_26: input

GPIO\_27: input

GPIO\_28: input

GPIO\_29: input

GPIO\_30: input

GPIO\_31: input

GPIO\_32: input

bladeRF> print xb\_gpio all

Expansion GPIO register: 0xffffffff

GPIO\_1: 1

GPIO\_2: 1

GPIO\_3: 1

GPIO\_4: 1

GPIO\_5: 1

GPIO\_6: 1

GPIO\_7: 1

GPIO\_8: 1

GPIO\_9: 1

GPIO\_10: 1

GPIO\_11: 1

GPIO\_12: 1

GPIO\_13: 1

GPIO\_14: 1

GPIO\_15: 1

GPIO\_16: 1

GPIO\_17: 1

GPIO\_18: 1

GPIO\_19: 1

GPIO\_20: 1

GPIO\_21: 1

GPIO\_22: 1

GPIO\_23: 1

GPIO\_24: 1

GPIO\_25: 1

GPIO\_26: 1

GPIO\_27: 1

GPIO\_28: 1

GPIO\_29: 1

GPIO\_30: 1

GPIO\_31: 1

GPIO\_32: 1

Note that since neither the XB-200 or XB-100 is attached, the pin names are presented as they pertain to the U74 connector on the bladeRF board.

If an expansion board has been enabled via xb 100 enable or xb 200 enable, then only the available I/O pins will be listed. Additionally, the pins names will reflect their associated item on the board.

## Examples

The first example shows how to toggle pin 6 on header J7 of the XB-200 Transverter board.

bladeRF> xb 200 enable

Enabling XB-200 transverter board

XB-200 Transverter board successfully enabled

bladeRF> print xb\_gpio all

Expansion GPIO register: 0xffffcfe9

J7\_1: 1

J7\_2: 1

J7\_6: 1

J13\_1: 1

J13\_2: 1

J16\_1: 1

J16\_2: 1

J16\_3: 1

J16\_4: 1

J16\_5: 1

J16\_6: 1

bladeRF> print xb\_gpio\_dir all

Expansion GPIO direction register: 0x3c00383e

J7\_1: input

J7\_2: input

J7\_6: input

J13\_1: input

J13\_2: input

J16\_1: input

J16\_2: input

J16\_3: input

J16\_4: input

J16\_5: input

J16\_6: input

bladeRF> set xb\_gpio\_dir J7\_6 output

J7\_6: output

bladeRF> print xb\_gpio J7\_6

J7\_6: 1

bladeRF> set xb\_gpio J7\_6 0

J7\_6: 0

bladeRF> set xb\_gpio J7\_6 1

J7\_6: 1

When following the above, you'll see J7 pin 6 change from 1.8V to 0V, and then back to 1.8V.

The next examples shows how to toggle one of the colors in the RGB LED on the XB-100 expansion board. On the current revision of these boards, voltage on the FPGA I/O pins (LED cathode) is not high enough to turn off LEDs on the XB-100. As a workaround, set the LED pin's value to '1' and toggle its direction between 'input' and 'output' to turn the LEDs "off" and "on," respectively.

bladeRF> xb 100 enable

Enabling XB-100 GPIO expansion board

XB-100 GPIO expansion board successfully enabled

bladeRF> print xb\_gpio all

Expansion GPIO register: 0xf9ff01bb

J2\_3: 0

J2\_4: 1

J3\_3: 1

J3\_4: 0

J4\_3: 0

J4\_4: 0

J5\_3: 0

J5\_4: 0

J11\_2: 1

J11\_3: 1

J11\_4: 0

J11\_5: 1

J12\_5: 1

LED\_D1: 1

LED\_D2: 1

LED\_D3: 1

LED\_D4: 1

LED\_D5: 1

LED\_D6: 1

LED\_D7: 1

LED\_D8: 1

TLED\_RED: 1

TLED\_GREEN: 1

TLED\_BLUE: 1

DIP\_SW1: 0

DIP\_SW2: 0

DIP\_SW3: 0

DIP\_SW4: 0

BTN\_J6: 1

BTN\_J7: 1

BTN\_J8: 1

bladeRF> print xb\_gpio\_dir all

Expansion GPIO direction register: 0xf9f80000

J2\_3: input

J2\_4: input

J3\_3: input

J3\_4: input

J4\_3: input

J4\_4: input

J5\_3: input

J5\_4: input

J11\_2: input

J11\_3: input

J11\_4: input

J11\_5: input

J12\_5: input

LED\_D1: output

LED\_D2: output

LED\_D3: output

LED\_D4: output

LED\_D5: output

LED\_D6: output

LED\_D7: output

LED\_D8: output

TLED\_RED: output

TLED\_GREEN: output

TLED\_BLUE: output

DIP\_SW1: input

DIP\_SW2: input

DIP\_SW3: input

DIP\_SW4: input

BTN\_J6: input

BTN\_J7: input

BTN\_J8: input

bladeRF> set xb\_gpio\_dir TLED\_RED input

TLED\_RED: input

bladeRF> set xb\_gpio\_dir TLED\_RED output

TLED\_RED: output

bladeRF> set xb\_gpio\_dir TLED\_RED input

TLED\_RED: input

## Getting the most out of libtecla

If you've built bladeRF-cli with support for [libtecla](http://www.astro.caltech.edu/~mcs/tecla/index.html) (*highly* recommended), you'll have a number of great features available to you, including history, tab-completion (for filenames) and handy key bindings (including emacs and vi bindings).

See the [tecla man page](http://www.astro.caltech.edu/~mcs/tecla/tecla.html) for detailed information about configuring and using libtelca-based programs.

To use vi or emacs key bindings, create a ~/.teclarc file, and add the relevant line:

edit-mode vi

or

edit-mode emacs

## Using an External Clock Input

## bladeRF Classic

WIP, basically the SMB port acts as an input *or* output, but not both simultaneously.

## bladeRF2-micro

The bladeRF-micro features a dual-input clock buffer that can select either the on-board VCTCXO, or an external 38.4 MHz clock to be fanned out to the various components on the board. The on-board VCTCXO is selected by default.

Certain applications (e.g. MIMO, RADAR, etc.) may require multiple bladeRFs to be synchronized using the exact same clock source. For best results, an external clock distribution network with matched cables is used so the frequency and phase differences between each board is as close to zero as possible. The bladeRF2-micro can receive such an external clock using the U.FL port labeled, CLKIN (U93). The external clock must be 38.4 MHz and 3.3 V CMOS logic.

It is important to note that the FX3 requires the clock to be within 150 ppm (5 kHz) of its ideal frequency of 38.4 MHz. While Nuand has been able to deviate by as much as 100 kHz before experiencing stability issues, it is recommended to stay as close to 38.4 MHz as possible due to temperature and process variations.

Using the bladeRF-cli, select the external clock input port:

bladeRF> set clock\_sel external

Clock input: External via CLKIN

To return to the onboard VCTCXO:

bladeRF> set clock\_sel onboard

Clock input: Onboard VCTCXO

To view the clock selection at any time:

bladeRF> print clock\_sel

Clock input: Onboard VCTCXO

## Using the External Clock Output

## bladeRF Classic

WIP, basically the SMB port acts as an input *or* output, but not both simultaneously.

## bladeRF2-micro

One of the outputs of the bladeRF2-micro clock buffer is connected to an external U.FL port labeled CLKOUT (U92). This may be used to supply a 38.4 MHz, 3.3 V CMOS clock to other devices. This output may be enabled or disabled using the bladeRF-cli:

bladeRF> set clock\_out enable

Clock output: Enabled via CLKOUT

bladeRF> set clock\_out disable

Clock output: Disabled

bladeRF> print clock\_out

Clock output: Disabled

## The 'print hardware' Command Group

The print command of bladeRF-cli has been updated to include a hardware convenience parameter that groups what would otherwise be multiple status commands into a single command. Currently, only the bladeRF2-micro has this hardware group, but it will eventually include the bladeRF Classic.

An example usage of this command is shown below for a bladeRF2-micro.

bladeRF> print hardware

Hardware status:

RFIC status:

Temperature: 19.3 °C

CTRL\_OUT: 0xf8 (0x035=0x00, 0x036=0xff)

Power source: USB Bus

Power monitor: 4.78 V, 0.6 A, 2.88 W

RF switch config:

TX1: RFIC 0x0 (TXA ) => SW 0x0 (OPEN )

TX2: RFIC 0x0 (TXA ) => SW 0x0 (OPEN )

RX1: RFIC 0x0 (A\_BAL ) <= SW 0x0 (OPEN )

RX2: RFIC 0x0 (A\_BAL ) <= SW 0x0 (OPEN )

Under the RFIC status section, there is a temperature readout and a status of the CTRL\_OUT pins. The RFIC used on the bladeRF-micro has an on-chip thermocouple allowing it to measure the die temperature. The CTRL\_OUT pins are a set of eight real-time status outputs that can be configured to represent many different options, as dictated by the [AD9361](http://www.analog.com/en/products/ad9361.html) RFIC's register contents at addresses 0x035 and 0x036. To view the available options, review the [AD9361 Reference Manual (UG-570)](http://www.analog.com/en/design-center/landing-pages/001/integrated-rf-agile-transceiver-design-resources.html).

The power source is reported by the on-board automatic power multiplexer. The device above was using USB power at the time of the reading, but DC barrel power would be reported as such.

The bladeRF2-micro also has a power sensor module that reports instantaneous voltage, current, and power on the main 5 V rail.

Finally, the print hardware command will report the RF switch configuration. The RFIC on the bladeRF2-micro has two TX channels and two RX channels (2R2T MIMO). Each RX channel has three band selection options (A, B, and C), and each TX channel has two band selection options (A and B). An external RF switch is used to select which band is active on the SMA port.

## Open a Device by Serial Number

The bladeRF-cli allows the user to specify the device to open by passing a device string with the -d <device\_string> option. The format of device\_string is outlined in the bladerf\_open() section of the [libbladeRF API](http://nuand.com/libbladeRF-doc/v2.0.0/group%5F%5F%5Ff_n%5F%5F%5Fi_n_i_t.html#gab341ac98615f393da9158ea59cdb6a24).

When multiple bladeRFs are used, it is often very useful to specify the serial number of the desired board to use when opening bladeRF-cli. For example, to open serial number 1337...beef using any available backend, the following command may be used. Note that the full serial number is not required; only the first few digits to (reasonably) ensure its uniqueness.

$ bladeRF-cli -d "\*:serial=1337" -e version

bladeRF-cli version: 1.6.1-git-911387a3-dirty

libbladeRF version: 2.0.1-git-911387a3-dirty

Firmware version: 2.2.0-git-4bcdd6ec

FPGA version: Unknown (FPGA not loaded)

# Troubleshooting

Brian Glod edited this page on Jan 11, 2019 · [79 revisions](https://github.com/Nuand/bladeRF/wiki/Troubleshooting/_history)

This page provides some quick tips on troubleshooting problems.

If you run across an issue not listed here and find a solution, please feel free to post it here!

|  |
| --- |
| Table of Contents  * [Configuration and Compilation Issues](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Configuration_and_Compilation_Issues)   + [Build fails due to a warning being treated as an error](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Build_fails_due_to_a_warning_being_treated_as_an_error)   + [libbladeRF](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-libbladeRF)     - [Missing libusb](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Missing_libusb)     - [Outdated libusb](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Outdated_libusb)     - [libusb\_get\_version() is undefined](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-libusb_get_version_is_undefined)   + [Cannot build libtecla under Mac OSX 10.9 due to missing libgcc.a](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Cannot_build_codelibteclacode_under_Mac_OSX_109_due_to_missing_codelibgccacode) * [Runtime Issues](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Runtime_Issues)   + [Failing to see or connect to the device](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Failing_to_see_or_connect_to_the_device)     - [Linux: udev and permissions](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Linux_udev_and_permissions)       * [PyBOMBS](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-PyBOMBS)     - [Device may only be opened by one program](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Device_may_only_be_opened_by_one_program)     - [Device's VID and PID](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Devices_VID_and_PID)     - [Virtual Machines](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Virtual_Machines)     - [Windows](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Windows)     - [Mac OSX](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Mac_OSX)     - [Host or Device-specific Issues](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Host_or_Devicespecific_Issues)       * [Issues with XHCI and Linux](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Issues_with_XHCI_and_Linux)       * [Working configurations](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Working_configurations)       * [Problematic Configurations](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Problematic_Configurations)   + [Outdated firmware warning](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Outdated_firmware_warning)   + [Outdated FPGA warning](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Outdated_FPGA_warning)   + [bladeRF-cli detects device in bootloader mode](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-bladeRFcli_detects_device_in_bootloader_mode)   + [FPGA load failures](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-FPGA_load_failures)     - [Operation Timed Out](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Operation_Timed_Out)   + [File or device I/O failure](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-File_or_device_IO_failure)   + [Linux: Failure to find libbladeRF.so or libbladeRF.so.0](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Linux_Failure_to_find_libbladeRFso_or_libbladeRFso0)   + [[ERROR] Could not write to si5338 (-5): File or device I/O failure](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-ERROR_Could_not_write_to_si5338_5_File_or_device_IO_failure)   + [Mac OSX Superspeed issues](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Mac_OSX_Superspeed_issues) * [Hardware](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Hardware)   + [Power Rails](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Power_Rails)     - [bladeRF Classic](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-bladeRF_Classic)     - [bladeRF 2.0 micro](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-bladeRF_20_micro)   + [Clocks](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Clocks)     - [bladeRF Classic](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-bladeRF_Classic-2)     - [bladeRF 2.0 micro](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-bladeRF_20_micro-2) * [Getting help](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#user-content-Getting_help) |

## Configuration and Compilation Issues

## Build fails due to a warning being treated as an error

By default, builds use *-Werror* or */WX* to treat warnings as errors, with the intent of forcing developers to address any warnings.

Occasionally, we've seen different compiler versions throw errors that were not seen on developers' machines, breaking master for users.

Master is intended to always build. If you suspect a build is broken due to a warning being treated as an error, please report it via IRC or the issue tracker, and run the following from your build directory as a temporary workaround:

cmake -DTREAT\_WARNINGS\_AS\_ERRORS=No ../

## libbladeRF

### Missing libusb

If the libusb is reported to not be found, it may be the case that pkgconfig was not able to locate libusb. (This will certainly be the case for Windows.) Check your PKG\_CONFIG\_PATH variable, and ensure that libusb is installed to a location in your system's library search path.

Users that are in a \*nix environment and have installed libusb via a package manager, should double check that both the library and development packages for libusb-1.0 (sometimes packaged as libusbx-1.0) are installed.

Windows users should use the **LIBUSB\_PATH** CMake variable to specify the location of a the libusb(x) binary distribution.

### Outdated libusb

If you receive a message similar to the following when running CMake, you'll need to obtain a more recent version of libusb:

A newer libusb version is needed by libbladeRF. libusb >= v1.0.10 is required to build. libusb >= v1.0.16 is recommended.

For some distributions, it may be possible to simply pull in a package from an upstream source for a later distro version. However, this is not always possible, as more recent versions of libusb packages may require a newer libc or libudev.

For example, Ubuntu 12.04 users will find that libusb-1.0.9 is provided in the repositories. In this case, it is recommended to upgrade to a more recent distribution (a 3.8+ kernel is also highly desirable). If this is not possible, these users may want to try obtaining a libusb package from the 12.10 repositories (this has not been tested), or attempt to build a slightly newer version from source.

### libusb\_get\_version() is undefined

*Note: This build issue has been resolved as of*[*commit 95a6c1d0e098*](https://github.com/Nuand/bladeRF/commit/95a6c1d0e0988196b57205bff38a47251f0692b4)*. If you see this issue, ensure that you've checkout out the lastest codebase, remove and recreate your build directory, and re-run CMake.*

If you see an error about being unable to find the libusb\_get\_version function, the version of libusb installed on your machine needs to be updated. Updating to >= 1.0.10 for Linux is required, but >= 1.0.16 is **highly** suggested.

There is [an open action item](https://github.com/Nuand/bladeRF/issues/130) to change the CMake configuration to detect and complain about this, prior to compilation.

## Cannot build libtecla under Mac OSX 10.9 due to missing libgcc.a

Please take a look at [Building libtecla](https://github.com/Nuand/bladeRF/wiki/Building-libtecla#configure-and-build) and note the sed script required to remove the link to libgcc.a in the Makefile. This is due to the fact that the configure script for libtecla uses gcc -print-libgcc-file-name, which [does not provide a full-path](http://lists.cs.uiuc.edu/pipermail/llvmbugs/2013-February/027238.html).

Removing all notion of libgcc.a from the Makefile appears to alleviate the problem. Alternatively, one may be able to replace this item with the full path the the clang-provided libgcc.a.

## Runtime Issues

## Failing to see or connect to the device

### Linux: udev and permissions

If you are unable to access the bladeRF in Linux, it may be the case that your user does not have ample permissions to access the device.

As of git commit [0a2a3ac](https://github.com/Nuand/bladeRF/commit/0a2a3ace6bf6bd27d39821ea4ec6ce3ee9fae719) (2015-10-08), libbladeRF will print a warning if it is unable to probe/open a device due to insufficient permissions:

jon@nocontrol % bladeRF-cli -p

[WARNING @ libusb.c:292] Found a bladeRF via VID/PID, but could not open it due to insufficient permissions.

probe: No devices are available. If one is attached, ensure it

is not in use by another program and that the current

user has permission to access it.

jon@nocontrol [1] % bladeRF-cli -i

[WARNING @ libusb.c:428] Found a bladeRF via VID/PID, but could not open it due to insufficient permissions.

No bladeRF device(s) available.

If one is attached, ensure it is not in use by another program

and that the current user has permission to access it.

bladeRF> q

If using an older version, note that using the bladeRF-cli -v verbose will print out more information about error codes, and that setting BLADERF\_LOG\_LEVEL=debug in your environment will provide additional debug output.

In order to access the bladeRF, Linux users must ensure:

* The 88-nuand.rules file has been installed to /etc/udev/rules.d/
  + When building from source, the CMake option INSTALL\_UDEV\_RULES defaults to ON
* The user is in the group specified by the CMake variable BLADERF\_GROUP
  + BLADERF\_GROUP defaults to `plugdev`, because Debian/Ubuntu users are typically in this group when allowed to access removable USB storage devices. You may wish to change this to conform to your security policies or distribution.
* The udev rules have been reloaded since the installation of the aforementioned rules file
  + This can be done via sudo udevadm control --reload-rules and then re-plugging the device.

If you have verified the above items and still cannot see the device:

* Confirm that you see the device in the output of lsusb -v
* Look for any information about the device's connection in the output of dmesg

#### PyBOMBS

Because [PyBOMBS](http://gnuradio.org/redmine/projects/pybombs/wiki) allows users to perform installations as non-root users, it uses -DINSTALL\_UDEV\_RULES=OFF when configuring the bladeRF project with CMake. (Copying a file to /etc/udev/rules.d/ requires root access, which a PyBOMBS user may not have.)

Therefore, if you've installed bladeRF support via PyBOMBS you will need to manually install the udev rules:

1. Copy each [88-nuand-\*.rules.in](https://github.com/Nuand/bladeRF/blob/master/host/misc/udev) to 88-nuand-\*.rules.
2. Edit each 88-nuand-\*.rules file and replace all instances of GROUP="@BLADERF\_GROUP@" to GROUP="plugdev".
   1. This assumes your user is in the plugdev (default) group. Change as needed.
3. Copy this file to the udev rules directory: sudo cp 88-nuand-\*.rules /etc/udev/rules.d/ && sudo chmod 644 /etc/udev/rules.d/88-nuand-\*.rules
4. Reload rules: sudo udevadm control --reload-rules && sudo udevadm trigger
5. Replug the device.

### Device may only be opened by one program

Currently, only one program may have a device open at a time. Any additional calls to bladerf\_open() or bladerf\_open\_with\_devinfo() will fail with BLADERF\_ERR\_NODEV.

Under the hood, this is due to the first caller opening a device claiming the device's interfaces via libusb\_claim\_interface(). Further attempts to claim an interface will fail with LIBUSB\_ERROR\_BUSY.

Setting the libbladeRF log level to DEBUG or VERBOSE will generally make this issue apparent. See the bladeRF-cli -v option, or libbladeRF's bladerf\_log\_set\_verbosity() function for information on setting the log level.

For example, if a first instance of the bladeRF-cli is started:

$ bladeRF-cli -i -v verbose

bladeRF> info

Serial #: 00000000000000000000000000000000

VCTCXO DAC calibration: 0x8100

FPGA size: 40 KLE

FPGA loaded: yes

USB bus: 2

USB address: 6

USB speed: Hi-Speed

Backend: libusb

Instance: 0

If another instance of the CLI is run simultaneously, we'll see the aforementioned libusb error about a device being busy (as its interfaces are already claimed by a different device handle.):

$ bladeRF-cli -i -v verbose ~

[VERBOSE] Found a bladeRF

[DEBUG] FPGA currently does not have a version number.

[DEBUG] Could not claim interface 0 - LIBUSB\_ERROR\_BUSY

No device(s) attached.

bladeRF>

If you're encountering the following error message with GNU Radio programs, but have verified that the device operates in the CLI, double check that no other programs have the device open:

FATAL: bladerf\_source\_c failed to open bladeRF device libusb:instance=0

### Device's VID and PID

Under normal operation, the bladeRF should appear with VID=1d50 and PID=6066.

If you see a Cypress device with VID=04b4 and PID=00f3, your bladeRF's FX3 may be in its bootloader mode. In this case, you may see a message in the bladeRF-cli program that notes a potential bladeRF running in bootloader mode. See the [firmware upgrade](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-firmware) wiki page for information on flashing the device.

### Virtual Machines

If you are using a Virtual Machine, there have been issues reported when changing the interfaces or claiming different interfaces. The level of success has been varied depending on the type of port (XHCI or EHCI) and the VM software used.

It has been reported that some VM software does not fully support USB 3.0. Verify the speed that the VM detects the device at, and try falling back to 2.0 if 3.0 does not work.

Note that you may have to enable USB support in your VM's configuration. For example, in VirtualBox, one must check the "Enable USB Controller" and "Enable USB 2.0 (EHCI) Controller" options under a VM's USB settings.

### Windows

To see the device in Windows, ensure you have the appropriate inf installed so that Windows associated the bladeRF with a libusbK driver. Verify that you've followed [these steps](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Windows#installing-libusbx) in the Getting Started guide for Windows.

Windows driver signing enforcement may cause driver installation problems. Try using the latest Zadig on the computer you want to install the driver on. Zadig will self-sign the driver package and install the corresponding certificate.

In Windows 8, Libusbx may incorrectly report USB 3.0 SuperSpeed connection as Hi-Speed.

### Mac OSX

If you have performed an operating system update, and a previously working bladeRF stops working, you might have a mismatch between the OS and libusb. (For example moving from OSX 10.9.0 to 10.9.1). A sign of this will be that you can see the device via bladeRF-cli -p, but you cannot send or receive any data and instead you will see error messages like [ERROR] Failed to access peripheral. In this case, reinstall libusb:

$ sudo port -f uninstall libusb

$ sudo port install libusb

### Host or Device-specific Issues

As USB 3.0 is still relatively new, we've heard reports of some controller working great, and others not working at all. Ensure you have the latest drivers (and firmware, if needed) for your USB controller.

If possible, try testing the device on both USB 2.0 and USB 3.0 ports, as well as other host machines.

#### Issues with XHCI and Linux

Previously, there was a known problem with Intel XHCI controllers in Linux when killing an application using libbladeRF in a manner that didn't allow libusb\_close() to be called. This caused the device to not open again until a power-on reset was performed. This has been resolved in libbladeRF v1.2.0, via [commit 2c48ba9e](https://github.com/Nuand/bladeRF/commit/2c48ba9e53574c7f4d3f7b8a969d89667b94251c).

Other reported problems with Linux and XHCI drivers range from poor transfer rates to system-wide lockups. Should you encounter these, it is recommended that you try to use a more recent kernel version.

For example, the 3.13 kernel shipping with Ubuntu 14.04 is reported to cause system lockups when using a Renesas uPD720202-based USB 3.0 controller. However, the 3.16 kernel shipping with Ubuntu 14.10 does not appear to be affected. (For those needing to remain on the 14.04 LTS, it is recommended that you [build a more recent kernel](https://wiki.ubuntu.com/KernelTeam/GitKernelBuild))

#### Working configurations

Below is a list of configurations that people have reported success on. Feel free to add to this list. (Please try to keep it reasonably organized.)

|  |  |  |  |
| --- | --- | --- | --- |
| Configuration Name | Device/Motherboard/CPU | Controller or Chipset | Operating System(s) |
| jynik's Desktop | [EVGA x58 FTW3](http://www.evga.com/support/manuals/files/132-GT-E768.pdf), Intel i7 | Onboard NEC/Renesas controller and [2-Port PCIe card](http://www.rocketfishproducts.com/products/computer-accessories/RF-P2USB3.html) NEC/Renesas | Windows 7, XUbuntu 12.10, 13.04, 13.10 (All 64-bit) |
| jynik's Laptop | Thinkpad W530, Intel i7-3630QM | Intel 7 Series/C210 Series xHCI (VID=8086, PID=1e31) | Windows 7, XUbuntu 13.04 (All 64-bit). **Note**: The XHCI issues mentioned above applies to the Linux configuration here. Able to process 40 Msps in osmocom\_fft without overruns. |
| drmpeg's Desktop | Dell T3600, Intel E5-1607 | Onboard NEC uPD720200 xHCI | Ubuntu 13.04 (32-bit just to run old XView stuff). Able to process 36 Msps in osmocom\_fft without overruns. |
| tnt's laptop | Apple Mac Book Pro Late 2008 (MacBookPro5,1) | ExpressCard ASM1042 (VID=1b21, PID=1042) | Gentoo (3.8.10-gentoo x86\_64) |
| Anker PCI-E card | Uspeed USB 3.0 PCI-E Express Card with 4 USB 3.0 Ports | VL805 Chipset (VIA Technologies, Inc. Device 3483 (rev 01)) | Ubuntu 13.04 x86\_64. osmocom\_fft runs without issue at 40Msps |
| bglod's desktop | MSI X99A SLI Plus | VIA VL805 (VID=2109, PID=3431) | Arch Linux 4.10.6-1-ARCH |
| bglod's desktop | MSI X99A SLI Plus | Intel X99 (VID=8087, PID=8002 and PID=800a) | Arch Linux 4.10.6-1-ARCH |
| bglod's desktop | MSI X99A SLI Plus | ASMedia ASM1142 USB 3.1 (PCIe VID=1b21, PID=1242) | Arch Linux 4.10.6-1-ARCH |
| scancapecod's at work desktop | Dell Optiplex 9020 / i5-4570 @ 3.20GHz | Intel 8 Series/C220 Series USB Enhanced Host Controller, Intel USB 3.0 eXtensible Host Controller | Windows 7 Pro 64 bit Service Pack 1. SDR-Radio 30 MHz display |
| HoopyCat's desktop | ASUS Z87-PRO / i7-4770 | Intel Corporation 8 Series/C220 Series Chipset Family USB xHCI (rev 05) | Linux Mint 16 x86\_64 |
| piranha32's desktop | Asus P6T/i7 [920@2.66GHz](mailto:920@2.66GHz) | Syba SD-PEX20139, uPD720201 | Fedora 21 x86\_64. Max RX sample rate measured using bladeRF-cli: 31Msps |
| piranha32's desktop | Asus P6T/i7 [920@2.66GHz](mailto:920@2.66GHz) | IOCrest/Syba SY-PEX20140, Etron EJ188 | Fedora 21 x86\_64. Max RX sample rate measured using bladeRF-cli: 83Msps |

#### Problematic Configurations

Issues have been reported on the following configurations:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Configuration Name | Device/Motherboard/CPU | Controller or Chipset | Operating System(s) | Issues and symptoms |
| steve-m's desktop (with onboard xHCI controller) | ASRock Z68 Pro3 Gen3, Z68, Intel i5-2500K | Etron Technology, Inc. EJ168 USB 3.0 Host Controller (rev 01) | Xubuntu 13.10 x86\_64, Linux 3.11 | "Failed to access peripheral", libusb timeouts, kernel reports "Not enough bandwidth for altsetting 0" and other xHCI related errors |
| Orico PCI-E card | PVU3-4P USB 3.0 PCI-Express Host Controller Card | VIA Technologies, Inc. VL80x xHCI USB 3.0 Controller (rev 03) | Ubuntu 13.04 X86\_64 | Hard lock while loading firmware |
| piranha32's desktop | Asus P6T / i7 [920@2.67GHz](mailto:920@2.67GHz) | SABRENT CP-4PTU USB3 PCIe card with VIA Technologies, Inc. VL80x xHCI USB 3.0 Controller (rev 02) | Fedora 19 x86\_64. | The card worked flawlessly for about a month, but after that time started to hard lock the motherboard on every attempt to stream from bladeRF. |
| [3.5" 4 Port USB 3.0 Bay Hub with All-in-One USB 2.0 Card Reader](http://www.sybausa.com/productInfo.php?iid=1157) | lsusb output: 2109:0811 058f:6364 Alcor Micro Corp. AU6477 Card Reader Controller |  | Arch Linux w/ kernel 3.13.5-1-ARCH | Hard lock when reading samples. Needed to reboot. |

## Outdated firmware warning

This warning comes up if you have a that is considered out-of-date. It may be considered out of date due to libbladeRF requirements, or based upon the FPGA version that was detected.

Please update to the [latest firmware](http://nuand.com/fx3/latest.img), per the instructions in [this wiki page.](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-firmware)

## Outdated FPGA warning

libbladeRF prints a warning if it detects that the device's firmware requires a more recent FPGA version. If you see this message, fetch the associated FPGA image from [the FPGA download page](https://www.nuand.com/fpga.php).

## bladeRF-cli detects device in bootloader mode

In the event that flash is corrupted, the bladeRF falls back to a bootloader mode, courtesy of the Cypress FX3 USB 3.0 peripheral controller. From this bootloader, we can [re-flash the firmware](https://github.com/Nuand/bladeRF/wiki/Upgrading-bladeRF-firmware#upgrading-using-the-fx3-bootloader).

The current bladeRF-cli version should detect and report FX3-based devices found in bootloader mode, as shown below:

$ bladeRF-cli -i

NOTE: One or more FX3-based devices operating in bootloader mode

were detected. Run 'help recover' to view information about

downloading firmware to the device(s).

No device(s) attached.

bladeRF>

In this situation, you should see a USB device enumerated as a Cypress Westbridge device. If the only FX3-based device plugged into your machine is a bladeRF you can continue to reflash it. Otherwise, ensure you're not flashing bladeRF firmware onto some other device.

Per the CLI's instructions, we can get additional information on how to recover from the bootloader:

bladeRF> help recover

Usage: recover [<bus> <address> <firmware file>]

Load firmware onto a device running in bootloader mode, or list all

devices currently in bootloader mode.

With no arguments, this command lists the USB bus and address for

FX3-based devices running in bootloader mode.

When provided a bus, address, and path to a firmware file, the

specified device will be loaded with and begin executing the provided

firmware.

In most cases, after successfully loading firmware into the device's

RAM, users should open the device with the "open" command, and write

the firmware to flash via "load fx3 <firmware file>"

In short, we can run recover to tell us the USB device and address in the bootloader mode, and then we can use this to specify which device to write the firmware to:

bladeRF> recover

FX3 bootloader devices:

---------------------------------------------------------

Backend: libusb

Bus: 1

Address: 5

Use 'recover <bus> <addr> <firmware>' to download and boot

firmware to the specified device.

Assuming we have the firmware, [bladeRF\_fw\_latest.img](http://nuand.com/fx3.php) in our current directory, we see that our next command should be recover 1 5 bladeRF\_fw\_latest.img:

bladeRF> recover 1 5 bladeRF\_fw\_latest.img

Success! Use "open" to switch to this device.

Note that a "load fx3 <firmware>" is required t

This recover command does not yet actually flash anything. It writes to and boots from RAM on the FX3. This is intended to allow developers to quickly experiment with different firmware loads, without having to always write to flash.

Next, we can open the device and write this firmware to flash.

bladeRF> open

bladeRF> version

bladeRF-cli version: 1.1.2

libbladeRF version: 1.2.1

Firmware version: 1.8.0

FPGA version: Unknown (FPGA not loaded)

bladeRF> load fx3 bladeRF\_fw\_latest.img

Flashing firmware from bladeRF\_fw\_latest.img ...

[INFO @ usb.c:611] Erasing 3 blocks starting at block 0

[INFO @ usb.c:616] Erased block 2

[INFO @ usb.c:624] Done erasing 3 blocks

[INFO @ usb.c:818] Writing 479 pages starting at page 0

[INFO @ usb.c:822] Writing page 478

[INFO @ usb.c:831] Done writing 479 pages

[INFO @ flash.c:110] Verifying 479 pages, starting at page 0

[INFO @ usb.c:716] Reading 479 pages starting at page 0

[INFO @ usb.c:719] Reading page 478

[INFO @ usb.c:730] Done reading 479 pages

Done. Cycle power on the device.

bladeRF>

At this point, one can exit the bladeRF-cli and power cycle the device. It should be detected as a bladeRF when it is next plugged in.

## FPGA load failures

### Operation Timed Out

In many cases, load failures have been the result of trying to load the wrong size FPGA image for the board in question. Check to make sure you are using the 40kLE FPGA for x40 and the 115kLE FPGA for x115.

If you're unsure which FPGA size is on your board, identify the the Altera Cyclone IV package on the top of the bladeRF. You should see text on the package that reads *EP4CExxxF23C8N*, where *xxx* is the size -- either 40 or 115.

## File or device I/O failure

If you're unable to load the FPGA using a USB 3.0 port while successfully doing so via USB 2.0, edit /linux/drivers/usb/host/xhci.h by changing TRBS\_PER\_SEGMENT from 64 to 256, and recompiling the kernel.

## Linux: Failure to find libbladeRF.so or libbladeRF.so.0

Ensure the location that you installed libbladeRF to is in your library search path. If you can't recall where this location is:

* Check the *install\_manifest.txt* from your build directory, if you built from source
* Make note of the libbladeRF location in the output of: ldd `which bladeRF-cli`

If running sudo ldconfig -v does not resolve the issue, you may have to add an ldconfig entry to ensure the proper libraty path is searched by the dynamic loader. To do this::

* As root (i.e., use sudo) create an /etc/ld.so.conf.d/bladeRF.conf and open it with your favorite editor
* Add a single line containing /usr/local/lib to the file. If you specified something other than /usr/local for *CMAKE\_INSTALL\_PREFIX* (the default), adjust this line accordingly.
* Run sudo ldconfig and then try running bladeRF-cli --help. The library search issue should now be resolved.

## [ERROR] Could not write to si5338 (-5): File or device I/O failure

One possible reason for this error is a mismatch between the FPGA bistream and the firmware being used. [As of firmware v1.6.0 and FPGA v0.0.1, the UART bridge baudrate has been increased.](http://nuand.com/forums/viewtopic.php?f=2&t=3514) Therefore, a mismatch would result in peripheral communication failures, as the FX3 and the FPGA are unable to communicate.

If you're seeing this issue, ensure that you've updated both your firmware (cycle power after flashing), and FPGA. Ensure you've disabled FPGA autoloading before flashing the new firmware, otherwise the (old) autoloaded FPGA will be attempting to use the lower bridge baudrate.

To disable FPGA autoloading: bladeRF-cli -L X

$ dmesg | grep xhci

xhci\_hcd 0000:03:00.0: Too many fragments 73, max 63

Affected Linux kernel versions: 3.12.6 to 3.12.9.

This has been patched in [3.12.10](https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/drivers/usb/host/xhci.h?h=linux-3.12.y&id=v3.12.10).

## Mac OSX Superspeed issues

If you've used ports to get libusb installed, there is a high likelihood that it is version 1.0.9. Unfortunately, libusb didn't add superspeed support until version 1.0.15 for OSX. libusb v1.0.17+, works on OSX.

## Hardware

## Power Rails

When first plugging in the bladeRF, a single LED should illuminate to indicate power is being delivered to the board. On the bladeRF Classic, this is D1. On the bladeRF 2.0 micro, this is D4. If the power indicator does not turn on and 5 V is being supplied, then it is likely there is something wrong with the power supply rails on the board. However, it is also possible for this LED to be illuminated and still have a power issue.

With the board power connected and a voltmeter (multi-meter set to Volts) in hand, proceed with the following probe points to narrow down the source of the power failure. Remember to pick a solid ground reference for the black probe. A good place is the body of the SMA or SMB ports.

For the purposes of positioning consistency, orient the board so the USB port is on the left and the component side is facing up.

### bladeRF Classic

Open the [bladeRF Classic schematic](https://nuand.com/bladerf.pdf).

Page 1 shows the general power architecture. The main 5 V rail feeds a 1.2 V SMPS and a 3.58 V SMPS. The 1.2 V switcher provides the core voltage to the FX3 and FPGA. The 3.58 V switcher supplies several low-dropout linear regulators that drop the voltage down to 3.3 V, 2.5 V, and 1.8 V for various components including the LMS6.

1. Probe the 5 V rail at J70 pin 2 or 5. Top side, near DC barrel jack. It should read at least 4.5 V.
2. Probe VBUS\_IN at J70 pin 3 or 4. Top side, near DC barrel jack. It should read at least 4.5 V.
3. Probe the 1.2 V switcher output at C368. Top side, top-middle, center-left. It should read ~1.2 V, same with each sub-point below.
   1. Probe each side of FB4 (bottom side, near USB port)
   2. Probe each side of FB5 (bottom side, near USB port)
   3. Probe each side of FB6 (bottom side, near USB port)
   4. Probe each side of FB7 (top side, between USB port and U43/FPGA)
4. Probe the 3.58 V switcher output at C329. Top side, top-middle, center-left. It should read ~3.58 V.
5. U49 VA1P8: top side, top right corner
   1. Pin 1 should read ~3.58 V
   2. Pin 5 should read ~1.8 V
6. U50 VD1P8: top side, bottom center-right
   1. Pin 1 should read ~3.58 V
   2. Pin 5 should read ~1.8 V
7. U66 VA2P5: top side, left of GPIO connector U74
   1. Pin 1 should read ~3.58 V
   2. Pin 5 should read ~2.5 V
8. U55 VD3P0\_CLK: top side, top center board
   1. Pin 1 should read ~3.58 V
   2. Pin 5 should read ~3.0 V
9. U53 V3P3\_RX\_LMS: top side, top right corner
   1. Pin 1 should read ~3.58 V
   2. Pin 4 should read ~3.3 V
10. U54 V3P3\_TX\_LMS: top side, bottom right corner
    1. Pin 1 should read ~3.58 V
    2. Pin 4 should read ~3.3 V
11. U65 VD3P3: top side, bottom center-right
    1. Pin 1 should read ~3.58 V
    2. Pin 5 should read ~3.3 V
12. U37: top side, middle-top, center-left
    1. Pin 5 USB3\_VBUS should read at least ~4.5 V
    2. Pin 4 FLAG should read at least ~4.5 V.

If any of these rails seem lower than nominal, please contact Nuand via e-mail for assistance. Please include a brief description of the problem, expected results, any relevant log information, and the voltage rail(s) that seem suspect.

### bladeRF 2.0 micro

TODO

## Clocks

If the device is not detected by the host and the voltage rails are within their nominal ranges, it is possible there is an issue with the main 38.4 MHz clock or its distribution to the FX3. Armed with an oscilloscope and a fine-tipped probe, proceed with the following debug steps.

### bladeRF Classic

1. U42 is the VCTCXO crystal itself, the raw source of the 38.4 MHz clock. Probe U42 pin 3 with the oscilloscope. A 3.0 V clock at 38.4 MHz should be visible.
2. U67 is a 1:2 clock buffer. Pin A2 (CLK\_OUT1) goes to the FX3. Pin D2 (CLK\_OUT2) goes to U68, the Si5338 clock generator.
   1. Pin A2: the only way to probe this clock output is to scrape off the solder mask on a via along the trace. This can damage your board and may void your warranty. If you suspect this clock output to be problematic, please contact Nuand via e-mail.
   2. Pin D2: probe at U68 pin 3 (IN3). A 3.0 V clock at 38.4 MHz should be visible.
3. U68 is the Si5338 clock generator. It has 7 clock outputs in use. None are critical for initial board detection by the host. For proper operation, all clocks should be functional.
   1. Pin 21 (OUT0B): LMS\_CLK LMS6 PLL clock
   2. Pin 22 (OUT0A): C4\_CLK main FPGA clock
   3. Pin 18 (OUT1A): LMS\_RX\_CLK LMS6 RX data interface clock\*
   4. Pin 14 (OUT2A): LMS\_TX\_CLK LMS6 TX data interface clock\*
   5. Pin 13 (OUT2B): C4\_TX\_CLK FPGA TX data interface clock\*
   6. Pin 10 (OUT3A): SMB\_CAP off-board SMB connector\*
   7. Pin 9 (OUT3B): EXP\_CLK Expansion interface clock\*

\* Clock may not toggle unless it is enabled or in use.

### bladeRF 2.0 micro

TODO

## Getting help

If you're really stuck, join the #bladeRF channel on [Freenode](http://webchat.freenode.net/) or post to the [Nuand Troubleshooting forum](http://nuand.com/forums/viewforum.php?f=4).

Folks in the community are generally very happy to help. However, to make the most of everyone's time, please try to have as much of following information available as you can. Providing this information up front can help you get to the bottom of things much quicker.

* Information about the target host machine
  + Are you in a VM? See the [Virtual Machines](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#Virtual-Machines) section for some known problems.
  + Are you running on a 32-bit or 64-bit processor?
  + What is your USB controller? Given that USB3 is still moderately new, some controllers have found to perform significantly better (or worse) than others.
  + Are you connecting your device to a 2.0 or 3.0 port? If a 3.0 port, is it actually enumerating at SuperSpeed (or at Hi-Speed?)
  + Is this a tablet or smartphone?
* The OS you're using
  + For Linux, please note your distribution (and that distribution's version), as well as kernel version
* Version information for the items you're having issues with
  + When running the bladeRF-cli in interactive mode (bladeRF-cli -i), the version command may be used to display information for both host items and the attached device. This is generally helpful in identifying if a problem you're seeing has been fixed in more recent versions.
* Any relevant error messages or warnings.
  + Run the bladeRF-cli with -v verbose to enable all output
  + (Linux users) Check the output of dmesg and lsusb
  + For GNU Radio and gr-osmosdr issues, note the versions of these you're running.
* Have you followed the steps in the [Getting Started](https://github.com/Nuand/bladeRF/wiki/Home#getting-started-guides) guides?
  + If you're currently in the process of following these guides, which steps have you already performed, and at which step are you encountering problems with?

# Getting Started: XB200 Transverter Board

Robert Ghilduta edited this page on Aug 30, 2019 · [45 revisions](https://github.com/Nuand/bladeRF/wiki/Getting-Started:-XB200-Transverter-Board/_history)

This page provides an overview of the **XB-200** and provides some information on its use.

|  |
| --- |
| Table of Contents  * [Introduction](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Introduction) * [Hardware](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Hardware)   + [Architecture](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Architecture)   + [SMA Connectors](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-SMA_Connectors)     - [Example Connection](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Example_Connection)   + [Important Note: Removal of R62 and R63](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Important_Note_Removal_of_R62_and_R63)   + [GPIO ports](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-GPIO_ports)     - [J7](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-J7)     - [J13](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-J13)     - [J16](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-J16) * [Software Support](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Software_Support)   + [libbladeRF](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-libbladeRF)   + [bladeRF-cli](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-bladeRFcli)   + [gr-osmosdr](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-grosmosdr)     - [Enabling the XB-200 and Selecting Filters](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-Enabling_the_XB200_and_Selecting_Filters)   + [GQRX](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-GQRX)   + [SDR#](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-SDR)   + [SDR-Radio](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB200-Transverter-Board#user-content-SDRRadio) |

## Introduction

The **XB-200 transverter board** is a block up-down converter that expands the bladeRF’s lower frequency range, allowing the bladeRF to be used in HF/VHF applications.

The RX and TX paths each have a set of 3 filters, at the 50 MHz ‑ 54 MHz (6 meter) band, 149 MHz ‑ 159 MHz (2 meter) band, and 206 MHz ‑ 235 MHz (includes 1.25 m) bands. There are also pairs of SMA connectors that will let users plug their own band filters into the RF path.

The XB-200 mates to the top of the bladeRF as follows. (There is only one possible orientation)

* XB-200 **U20** to bladeRF **U74**
* XB-200 **J5** to bladeRF **J61**
* XB-200 **J6** to bladeRF **J60**

For additional information, please see:

* [XB-200 product page](http://www.nuand.com/blog/product/hf-vhf-transverter/)
* [XB-200 schematics](http://nuand.com/xb200.pdf)
* [XB-200 filter designs](http://nuand.com/RF_filters.pdf)

## Hardware

## Architecture

The transverter was meant to extend the range of the bladeRF without impairing the current frequency capabilities. As such, the transverter has a bypass path as well as a mixed path. The bypass path just connects the antenna port to the IF port directly without any modification to the signal.

The mixed path first has a filterbank for selectivity filtering. This filterbank consists of 4 separate paths: 50 MHz filter, 144 MHz filter, 222 MHz filter and a custom filter. The custom filter can be put in line using the filter SMA connections on the transverter.

Each of the filters was designed to try to notch the terrestrial FM band as much as possible. To be able to receive normal FM, the simplest way is to put an SMA jumper in the custom path (i.e., between the RX/TX **FILT** and **FILT-ANT** SMA connectors).

The block converter stage uses an ADF4351 to produce a 1248 MHz high-side injection tone. The mixing frequency was chosen because of it being about 3x higher than the 300 MHz highest frequency we want to use and we can run the ADF4351 in integer-N mode, reducing spurs which may have resulted from a fractional-N mode of operation. The output of the ADF4351 is always divided by 2x, so 38.4 MHz \* 32.5 = 1248 MHz.

Since high side injection is used, the LMS6002D on the bladeRF is tuning to 1248 MHz - (desired frequency). Moreover, there is a spectral inversion that is occurring due to the choice of high-side injection. The LMS6002D is programmed to swap I and Q on the data bus being presented to the FPGA to correct for the flip.

## SMA Connectors

There are 10 SMA connectors on the XB-200. Below are brief descriptions of each.

RX

* **RXIF** (*J3*): RX Intermediate Frequency. Connect this to the bladeRF RX port (*J53*) via an SMA cable.
* **RXANT** (*J12*): RX Input. Connect an antenna here.
* **RXFILT** (*J8*): Mixer-side of the RX custom filter path. Connect this directly to **RXFILT-ANT** to use no filter, or connect this to one end of your custom filter, and the other end of your filter to **RXFILT-ANT**.
* **RXFILT-ANT** (*J9*): Antenna-side of the RX custom filter path.
* **ADC** (*J14*): This SMA exposes the direct ADC sampling path (skipping the LMS6002). This can also be accessed without the XB-200 via *J61* on the bladeRF.

TX

* **TXIF** (*J1*): TX Intermediate Frequency. Connect this to the bladeRF TX port (*J54*) via an SMA cable.
* **TXANT** (*J2*): TX output. Connect an antenna here. When not transmitting, it is recommended to keep a dummy load or attenuator on this port.
* **TXFILT** (*J11*): Mixer-side of the TX custom filter path. Connect this directly to **TXFILT-ANT** to use no filter, or connect this to one end of your custom filter, and the other end of your filter to **TXFILT-ANT**.
* **TXFILT-ANT** (*J10*): Antenna-side of the TX custom filter path.
* **DAC** (*J15*): This SMA exposes the DAC direct sampling path (skipping the LMS6002). This can also be accessed without the XB-200 via *J60* on the bladeRF.

### Example Connection

Note SMA cables on the top are just shorting the custom filter path SMA ports together (i.e. no filtering is being done). Depending on the application, it may be desirable to insert a filter instead. If the custom path is not used, the RXFILT/RXFILT-ANT and/or TXFILT/TXFILT-ANT may be left floating. The DAC and ADC SMA ports on the XB-200 may be left floating.

## Important Note: Removal of R62 and R63

Resistors R62 and R63 on the XB-200 were populated for early production boards.

Unfortunately, this causes unbalanced loading on the analog signals feeding the RF chain and will **severely degrade** performance. It is highly recommended to **remove** R62 and R63 on the XB-200 if they are populated.

These resistors may be found slightly above the J15 DAC SMA connector, and are near the pins of J6 ([image](https://www.nuand.com/images/xb200_R62_R63.png), [schematic](https://www.nuand.com/xb200.pdf)).

## GPIO ports

GPIO pins are exposed on the XB-200 via **J7**, **J13**, and **J16**. libbladeRF API calls to manipulate these pins are a work in progress.

From page 7 of XB200 Schematic:

### J7

|  |  |  |
| --- | --- | --- |
| Pin | Signal | FPGA pin |
| 1 | GPIO3 | EXP\_GPIO\_10 |
| 2 | GPIO4 | EXP\_GPIO\_11 |
| 3 | VCCIO\_R | 1.8 V ref (FPGA right banks) |
| 4 | VCCIO\_R | 1.8 V ref (FPGA right banks) |
| 5 | GPIO1 | EXP\_GPIO\_8 |
| 6 | GPIO2 | EXP\_GPIO\_9 |

### J13

|  |  |  |
| --- | --- | --- |
| Pin | Signal | FPGA pin |
| 1 | GPIO6 | EXP\_GPIO\_17 |
| 2 | GPIO8 | EXP\_GPIO\_18 |
| 3 | Ground |  |
| 4 | Ground |  |
| 5 | VCCIO\_R | 1.8 V ref (FPGA right banks) |
| 6 | GPIO5 | EXP\_SPI\_MOSI |

### J16

|  |  |  |
| --- | --- | --- |
| Pin | Signal | FPGA pin |
| 1 | GPIO9 | EXP\_GPIO\_31 |
| 2 | GPIO10 | EXP\_GPIO\_32 |
| 3 | GPIO11 | EXP\_GPIO\_19 |
| 4 | GPIO12 | EXP\_GPIO\_20 |
| 5 | GPIO13 | EXP\_GPIO\_21 |
| 6 | GPIO14 | EXP\_GPIO\_24 |

## Software Support

This section is intended to provide information regarding support for the XB-200 in various pieces of software. Please keep libbladeRF first, but feel free to add or stub out programs you'd like to see more information for.

## libbladeRF

The general procedure for using the XB-200 programmatically, via libbladeRF is as follows:

* With an open device handle, enable support for the XB-200 via:  
  [bladerf\_expansion\_attach(dev, BLADERF\_XB\_200)](https://github.com/Nuand/bladeRF/blob/15340770ca59d30b82109ad8670aa0fe027ca9fe/host/libraries/libbladeRF/include/libbladeRF.h#L980)
* Select the desired filter bank for the RX and TX paths. For example, to configure the RX module to use the 149 MHz ‑ 159 MHz filter:  
  [bladerf\_xb200\_set\_filterbank(dev, BLADERF\_MODULE\_RX, BLADERF\_XB200\_144M)](https://github.com/Nuand/bladeRF/blob/15340770ca59d30b82109ad8670aa0fe027ca9fe/host/libraries/libbladeRF/include/libbladeRF.h#L1012)
  + See the [bladerf\_xb200\_filter](https://github.com/Nuand/bladeRF/blob/15340770ca59d30b82109ad8670aa0fe027ca9fe/host/libraries/libbladeRF/include/libbladeRF.h#L492) enumeration for other options.
* Specify that the XB-200 mixer path should be used:  
  [bladerf\_xb200\_set\_path(dev, BLADERF\_MODULE\_RX, BLADERF\_XB200\_MIX)](https://github.com/Nuand/bladeRF/blob/15340770ca59d30b82109ad8670aa0fe027ca9fe/host/libraries/libbladeRF/include/libbladeRF.h#L1041)
  + To tune to the "normal" bladeRF ranges, bypass the XB-200 mixer by using the [BLADERF\_XB200\_BYPASS](https://github.com/Nuand/bladeRF/blob/15340770ca59d30b82109ad8670aa0fe027ca9fe/host/libraries/libbladeRF/include/libbladeRF.h#L498) value with the above function.

## bladeRF-cli

To tell the CLI that the XB-200 has been attached, simply run the command xb 200 enable. This will expand the frequency range that the board is able to tune.

To select RX or TX filter banks, use the command xb 200 filter <rx|tx> <50|144|222|custom auto\_1db|auto\_3db.

"50" refers to the 50‑54 MHz (6 meter) band, "144" refers to the 144‑148 MHz (2 meter) band, and "222" refers to the 222‑225 MHz (1.25 meter) band. Technically, the filter associated with the "222" options covers 206‑235 MHz.

The "custom" option enables the custom filter bank path.

The "auto\_1db" and "auto\_3b" options instruct libbladeRF to automatically select the appropriate filterbank based upon the filters' 1 dB and 3 dB points based upon the current device frequency. Outside of these regions, the custom filter path will be selected.

## gr-osmosdr

As of commit [9cb023](http://cgit.osmocom.org/gr-osmosdr/commit/?id=9cb023b00a36d04ad8ea7e0bf08ad0623ab3e95f), gr-osmosdr supports natively the XB-200. Note that the way to enable the board has changed slightly from the previous patched method, so be sure to read the next section very carefully and make appropriate modifications to the device string.

### Enabling the XB-200 and Selecting Filters

To enable the XB-200 in software that utilizes gr-osmosdr, add **xb200** to the source/sink arguments string. By default, the filter bank selected will automatically tune with the tuned frequency at the 1 dB points. Therefore, you must have the **FILT** and **FILT-ANT** connected through a filter, or directly connected together (no filter).

If you wish to select one of the other filter paths, append one of the following to the source/sink arguments string:

* xb200=50M
* xb200=144M
* xb200=222M
* xb200=auto
  + This option will select an appropriate filter path based upon the selected center frequency with 1 dB corners. This is the default if no filter is selected.
* xb200=auto3db
  + This option will select an appropriate file path based upon the selected center frequency with 3 dB corners.
* xb200=custom

Below is an example device string that enables the XB-200 and selects the 144 MHz ‑ 148 MHz filter:

bladerf=0,xb200=144M

## GQRX

As this utilizes gr-osmosdr, you can simply follow the information in the previous section regarding the device arguments.

## SDR#

The bladeRF can be used with SDR# via a [plugin written by Jean-Michel Picod](https://github.com/jmichelp/sdrsharp-bladerf). Follow the instructions on his GitHub page, and feel free to discuss/comment/ask questions about the plugin in Jean-Michel's [forum thread](https://nuand.com/forums/viewtopic.php?f=5&t=3597).

## SDR-Radio

The current beta version (v2.3 build 1990 at the time of writing) of SDR-Radio assumes an XB-200 is attached, and performs the necessary calls to enable it.

If you do not have an XB-200 attached, this behavior causes a clock signal to be driven to the expansion port, which may introduce undesired artifacts. To avoid this, you can [build your own modified bladeRF.dll](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Windows#Building_from_source) with a change that replaces [this line in bladerf.c](https://github.com/Nuand/bladeRF/blob/13dde0f6d53a11042a10cac6535539db3a7b53a4/host/libraries/libbladeRF/src/bladerf.c#L1305) with status = 0; to prevent the expansion board enable from taking place. You may then copy your modified DLL to the SDR-Radio installation directory, replacing the existing bladeRF.dll (which you should first back up).

# Getting Started: XB300 Amplifier Board

Brian Glod edited this page on Aug 28, 2018 · [5 revisions](https://github.com/Nuand/bladeRF/wiki/Getting-Started:-XB300-Amplifier-Board/_history)

|  |
| --- |
| Table of Contents  * [Overview](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-Overview) * [Hardware Configuration](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-Hardware_Configuration)   + [RX](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-RX)   + [TX](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-TX)   + [TRX](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-TRX)   + [Example Connection](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-Example_Connection) * [Usage](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-Usage)   + [libbladeRF](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-libbladeRF)   + [bladeRF-cli](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-bladeRFcli)   + [Recommendations](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-XB300-Amplifier-Board#user-content-Recommendations) |

## Overview

The [Nuand XB‑300](https://www.nuand.com/blog/product/amplifier/) is an amplifier expansion card that greatly increases the range of the bladeRF. On the receive side, the XB‑300 features a low-noise amplifier (LNA) and a combiner for antenna diversity. The transmit side features a Power Amplifier (PA), TRX switch, and a highly accurate ADC for measuring output power of the PA. The schematic for the XB‑300 [is available here](https://nuand.com/xb300.pdf).

The [TSS‑53LNB wideband (0.5-5 GHz) LNA](http://www.minicircuits.com/pdfs/TSS-53LNB+.pdf) found in the RX path provides a low noise figure and a maximum gain of 20 dB.

The [SE2623L PA](http://www.skyworksinc.com/uploads/documents/SE2623L_202397E.pdf) in the TX path offers up to 33 dBm of gain in the 2.4 GHz ISM band. It features an integrated temperature-compensated power detector.

The XB-300 ships in a configuration designed for separate RX and TX paths. Thus, if you are using an XB-300 in its "out of the box" state, follow the **RX** and **TX** instructions below.

The population of resistors R8, R10, and R23 select the input to the RX LNA. Users may adjust the resistor options to instead use a single antenna and the on-board TRX switch, or add an additional RX antenna input to a combiner for diversity.

## Hardware Configuration

## RX

1. Connect the bladeRF **RX** SMA connector (*J53*) to the XB-300 **RXIF** SMA (*J7*) with an SMA cable.
2. Connect a cable or antenna to the XB-300 **RX ANT** SMA connector (*J5*).

## TX

1. Connect the bladeRF **TX** SMA connector (*J54*) to the XB-300 **TXIF** SMA (*J4*) with an SMA cable.
2. Connect a cable or antenna to the XB-300 **TRX ANT** SMA connector (*J3*).

## TRX

Follow the above instructions for **TX**

## Example Connection

The following demonstrates the SMA and antenna connections when using one RX antenna and one TX antenna. This is the default, as-shipped, configuration.



## Usage

The general steps for using the XB-300 are as follows:

1. Enable the board. This configures and initializes the expansion port GPIOs.
2. Enable the LNA and/or PA.

## libbladeRF

Enable the XB-300:

int status = bladerf\_expansion\_attach(device\_handle, BLADERF\_XB\_300);

Enable the the LNA or PA:

bladerf\_xb300\_amplifier amp = BLADERF\_XB300\_AMP\_LNA; /\* Or BLADERF\_XB300\_AMP\_PA \*/

bool enable = true;

int status = bladerf\_xb300\_set\_amplifier\_enable(device\_handle, amp, enable);

For users that have performed the TRX resistor modification, the TRX switch can be configured as follows:

bladerf\_xb300\_trx trx = BLADERF\_XB300\_TRX\_RX; /\* Or BLADERF\_XB300\_TRX\_TX \*/

int status = bladerf\_xb300\_set\_trx(state->dev, trx);

## bladeRF-cli

The below snippet lists the commands needed to enable the XB-300, turn on the TX PA, and to turn on the RX LNA:

bladeRF> xb 300 enable

Enabling XB-300 Amplifier board

XB-300 Amplifier board successfully enabled

bladeRF> xb 300 pa on

PA: Enabled

bladeRF> xb 300 lna on

LNA: Enabled

For users that have performed the TRX resistor modification, the TRX switch can be configure as followed:

# Set switch to TX

bladeRF> xb 300 trx tx

# Set switch to RX

bladeRF> xb 300 trx rx

## Recommendations

An [external DC power supply](https://www.nuand.com/blog/product/power-supply/) is highly recommended when using the XB-300. See [this page](https://github.com/Nuand/bladeRF/wiki/bladeRF-Accessories#bladerf-jumper-configuration) for instructions on how to power the bladeRF from an external DC supply instead of the USB connection.

When using the TX PA, additional external filtering for the band/channel of interest is also **highly** recommended. The connectorized filters provided by [Mini-Circuits](https://www.minicircuits.com/WebStore/Filters.html) are a great option, but there are many other vendors and options available.

# Building libbladeRF for Windows

|  |
| --- |
| **Table of Contents**   * + [A warning](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-A_warning)   + [Tools and Dependencies](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Tools_and_Dependencies)     - [Installing git for Windows](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Installing_git_for_Windows)     - [Installing Visual Studio Community 2017 for Windows Desktop](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Installing_Visual_Studio_Community_2017_for_Windows_Desktop)     - [Installing libusb](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Installing_libusb)     - [Installing and configuring zadig](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Installing_and_configuring_zadig)     - [Installing pthreads-win32](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Installing_pthreadswin32)       * [Using pre-built pthreads-win32](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Using_prebuilt_pthreadswin32)       * [pthreads-win32 patch and recompile](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-pthreadswin32_patch_and_recompile)     - [Installing CMake](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Installing_CMake)   + [Building with CMake and Visual Studio Community 2017](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Building_with_CMake_and_Visual_Studio_Community_2017)     - [Cloning the bladeRF git repository](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Cloning_the_bladeRF_git_repository)     - [Configuring with CMake](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Configuring_with_CMake)     - [Compiling](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Compiling)   + [Verifying basic functionality](https://github.com/Nuand/bladeRF/wiki/Building-libbladeRF-for-Windows#user-content-Verifying_basic_functionality) |

**A warning**

**Be advised that building libbladeRF for Windows is a somewhat difficult process with a lot of manual steps, and this documentation may not be up-to-date with all of the necessary preparations.**

Unless you specifically need to modify libbladeRF itself, it is **strongly recommended** to use our pre-built, signed installer. See [Getting Started: Windows](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Windows) for information on getting and installing this. If you have already installed it, you can proceed to [Getting Started: Verifying Basic Device Operation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Verifying-Basic-Device-Operation).

**Tools and Dependencies**

**Installing git for Windows**

1. Download and install [Git for Windows](https://git-scm.com/) from their [Downloads page](https://git-scm.com/downloads).

**Installing Visual Studio Community 2017 for Windows Desktop**

1. Download Visual Studio Community 2017 for Windows Desktop from Microsoft, which can be found at <https://www.visualstudio.com/downloads/>
2. Follow the installation instructions, including any post-install updates.

*Note:* Visual Studio Community 2017 is also known as Visual Studio 15.2.

*Tip:* You can download time-limited virtual machine images from Microsoft with the latest Windows 10 and Visual Studio builds from <https://developer.microsoft.com/en-us/windows/downloads/virtual-machines>

**Installing libusb**

1. Download the latest Windows binary release of [libusb](http://libusb.info/), which also include development headers. As of this writing, [version 1.0.22](https://github.com/libusb/libusb/releases/tag/v1.0.22) is the latest.
   1. Note: download the libusb-1.0.xx.7z file from the GitHub release
   2. Note: you may need to download and install [7-zip from 7-zip.org](http://www.7-zip.org/) to open this file.
2. Extract the contents to a location of your choice. Make note of this location so that you can later provide it to CMake. The default configuration assumes that files will be in C:/Program Files (x86)/libusb-1.0.22, but if you wish to change the directory, you can change the LIBUSB\_PATH in CMake.

**Installing and configuring zadig**

1. Get the device driver installer (zadig): [http://zadig.akeo.ie](http://zadig.akeo.ie/) (older versions: <http://sourceforge.net/projects/libwdi/files/zadig/>)
2. Open Zadig.
3. If the bladeRF is not already listed in the dropdown box, go to Device->Create New Device, type a device name (i.e., "bladeRF") in the text box, and specify the VID/PID (*2cf0*/*5246* for bladeRF x40/x115, *2cf0*/*5250* for bladeRF xA4/xA9, *04b4*/*00f3* for any bladeRF device in bootloader/recovery mode) in the USB ID fields.
4. In the driver spinbox, select libusbK.
5. Click "Install Driver"
6. Open Device Manager. A new device called *bladeRF* should show up. If there is a yellow bang next to it in device manager:
   1. Right-click on the *bladeRF* entry and select "Update Driver Software...".
      1. Choose "Browse my computer for driver software"
      2. "Let me pick from a list of device drivers on my computer".
      3. Click "Have Disk..." and point it to the location that Zadig installed the driver to (C:\usb\_driver).
      4. Select "bladeRF" and continue through the wizard.
7. Device Manager should now show *bladeRF* under *libusbK USB Devices*.
8. Repeat the Create New Device steps for any other bladeRF VID/PID combinations you expect to encounter.

**Installing pthreads-win32**

**Using pre-built pthreads-win32**

The pthreads library is required to build libbladeRF and its utilities. A few steps are required to install this pthreads implementation. See [the pthreads-win32 website](http://www.sourceware.org/pthreads-win32/) for more information.

**WARNING**: pthreads-win32 version 2.9.1 does not work with newer versions of Windows / Visual Studio. See the "pthreads-win32 patch and recompile" section just below for a workaround.

1. Download the latest release. Currently this is [version 2.9.1](ftp://sourceware.org/pub/pthreads-win32/pthreads-w32-2-9-1-release.zip).
2. Extract the contents of the release zip.
3. Copy the contents of the Pre-built.2 directory to C:\Program Files (x86)\pthreads-win32

**pthreads-win32 patch and recompile**

If you are using a newer version of Windows / Visual Studio and/or are getting error C0211: 'timespec': 'struct' type redefinition when trying to build bladeRF, you will need to rebuild pthreads-win32. The below process works with pthreads-win32 2.9.1.

1. Get full source tree from [the pthreads-win32 website](http://www.sourceware.org/pthreads-win32/), and unpack someplace convenient
2. In Visual Studio, open pthreads-w32-2-9-1-release\pthreads.2\pthread.dsw (if prompted, a one-way upgrade is OK)
3. Right-click the solution, select "Retarget solution", hit OK.
4. Change "Solution Configurations" dropdown to "Release" ("Debug" doesn't work, but that's OK)
5. Edit pthread.h to add #define HAVE\_STRUCT\_TIMESPEC and #define PTW32\_ARCHx64 near the top, just below the include guard
6. Build the solution
7. Copy the contents of the pthreads.2 directory to C:\Program Files (x86)\pthreads-win32.

**Installing CMake**

Download and install CMake for Windows: <http://www.cmake.org/cmake/resources/software.html>

**Building with CMake and Visual Studio Community 2017**

**Cloning the bladeRF git repository**

1. Right click on the Desktop and select "Git GUI Here"
2. Select "Clone Existing Repository"
3. Paste https://github.com/Nuand/bladeRF.git into the Source Location field.
4. Select your preferred directory to clone the repository to, in the Target Directory field. This must be a directory that does not yet exist.
5. Click "Clone" and let it run. When it's done, close the Git Gui window. The bladeRF code is now on your computer.

**Configuring with CMake**

1. Run the CMake GUI utility.
2. Under "Where is the source code", browse to [preferred\_directory]/bladeRF/host.
3. Create a new directory, [preferred\_directory]/bladeRF/host/build.
4. Under "Where to build the binaries", browse to the newly created [preferred\_directory]/bladeRF/host/build. Click the Configure button.
5. Select your appropriate version of Visual Studio. For Visual Studio 2017, select "Visual Studio 15" or "Visual Studio 15 Win64". Select "Use default native compilers", then click "Finish".
6. If the configuration fails, double check the values for LIBUSB\_PATH and LIBPTHREADSWIN32\_PATH, and re-run the configuration.
7. Click on the Generate button.
8. A Visual Studio solution should now be available, host/build/bladeRF.sln.

**Compiling**

1. CMake has created a bladeRF.sln file. Open Visual Studio, and open this file. (You can also click "Open Project" in CMake)
2. A number of projects should show up in the Solution Explorer, including:
   1. bladeRF-cli
   2. bladeRF-fsk
   3. libbladerf\_shared
3. Select Debug or Release build from the drop down menu at the top
4. Run Build -> Build Solution
5. After the build completes, you should see a host/build/output directory in the build directory, containing either a Debug or Release subdirectory. These directories will contain generated executables, libraries, and will contain copies of the required DLLs.

*Note*: If the build fails with error C0211: 'timespec': 'struct' type redefinition, see "pthreads-win32 patch and recompile" above. You may need to delete the [preferred\_directory]/bladeRF/host/build directory and restart from the "Configuring with CMake" step.

**Verifying basic functionality**

Open up a cmd.exe window and navigate to the output directory associated with your build:

C:\Users\jon\Documents\projects\bladeRF\host\build\host\output\Debug>dir

10/19/2013 05:36 PM 122,880 bladeRF-cli.exe

10/19/2013 05:36 PM 139,776 bladeRF.dll

... Snipped out some other items ...

10/19/2013 05:35 PM 92,160 libusb-1.0.dll

10/19/2013 05:35 PM 55,808 pthreadVC2.dll

Note the presence of the libusb-1.0.dll and pthreadVC2.dll in this directory. These are placed here so that the bladeRF-cli and other tests may be run from this directory.

From here, you can continue on with the [Getting Started: Verifying Basic Device Operation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Verifying-Basic-Device-Operation) wiki page. If you encounter any errors or warnings, please see the [Troubleshooting](https://github.com/Nuand/bladeRF/wiki/Troubleshooting) guide.

7. Getting Started: Linux

This guide describes the process of building and installing the bladeRF host libraries and tool for a Linux system, and quickly getting a device up and running using pre-built firmware and FPGA images.

|  |
| --- |
| **Table of Contents**   * + [Easy installation: PyBOMBS](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Easy_installation_PyBOMBS)     - * [PYTHONPATH Errors](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-PYTHONPATH_Errors)     - [PyBOMBS udev rules](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-PyBOMBS_udev_rules)     - [PyBOMBS and gr-fosphor](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-PyBOMBS_and_grfosphor)   + [Easy installation for Ubuntu: The bladeRF PPA](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Easy_installation_for_Ubuntu_The_bladeRF_PPA)   + [Installing dependencies](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Installing_dependencies)     - [Debian-based distros (e.g., Ubuntu)](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Debianbased_distros_eg_Ubuntu)     - [Redhat-based distros (e.g., Fedora, CentOS)](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Redhatbased_distros_eg_Fedora_CentOS)   + [Building bladeRF libraries and tools from source](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Building_bladeRF_libraries_and_tools_from_source)     - [Clone the bladeRF git repository](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Clone_the_bladeRF_git_repository)     - [Configure the build](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Configure_the_build)     - [Perform the build and installation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Perform_the_build_and_installation)     - [Uninstalling later](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Uninstalling_later)   + [Checking basic device operation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Checking_basic_device_operation)   + [Building GNU Radio, gr-iqbal, and gr-osmosdr](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Building_GNU_Radio_griqbal_and_grosmosdr)     - [GNU Radio >= 3.7.7 and VOLK](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-GNU_Radio__377_and_VOLK)     - [Multiple GNU Radio Installations](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Multiple_GNU_Radio_Installations)     - [Using build-gnuradio to fetch dependencies and checkout source via Git](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Using_buildgnuradio_to_fetch_dependencies_and_checkout_source_via_Git)     - [Compile GNU Radio](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Compile_GNU_Radio)     - [Add GNU Radio to linker path and executable path](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Add_GNU_Radio_to_linker_path_and_executable_path)     - [Build and install gr-iqbal](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Build_and_install_griqbal)     - [Build and install gr-osmosdr](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Build_and_install_grosmosdr)     - [Simple test applications to verify device operation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Simple_test_applications_to_verify_device_operation)     - [Build and install GQRX](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Build_and_install_GQRX)       * [Tips](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#user-content-Tips) |

**Easy installation: PyBOMBS**

[PyBOMBS](https://github.com/gnuradio/pybombs) is essentially a package manager and build system for GNU Radio. It handles the fetching, building, and installation of GNU Radio and dependencies, and is largely system independent. The installation steps are listed on the [PyBOMBS GitHub](https://github.com/gnuradio/pybombs) page, but the following is a list of steps and basic packages needed to get up and running with the bladeRF.

Before compiling on Ubuntu systems, ensure the following packages are installed.

$ sudo apt-get install python3-six python3-mako python3-lxml python3-lxml python3-numpy python3-numpy python3-pip git python3-pybind11 libsndfile1-dev

NB: If the pybombs command is not found after pip install ensure that ~/.local/bin is part of your path. If it is not, run (and also add the following command to your .bashrc):

export PATH=$PATH:$HOME/.local/bin

Install PyBOMBS (using pip, but other methods are listed on the PyBOMBS README.md), apply the default configuration, and add the default recipes:

$ pip3 install --upgrade git+https://github.com/gnuradio/pybombs.git

$ pybombs auto-config

$ pybombs recipes add-defaults

To use Nuand's experimental gr-osmo, consider running the following:

$ sed -i 's/gitea.osmocom.org.sdr/github.com\/Nuand/g' ~/.pybombs/recipes/gr-recipes/gr-osmosdr.lwr

$ sed -i '/- gr.\{7,\}/d' ~/.pybombs/recipes/gr-recipes/gr-osmosdr.lwr

$ sed -i '/- gr.iqbal/d' ~/.pybombs/recipes/gr-recipes/gr-osmosdr.lwr

Optionally to enable gr-video-sdl, try:

$ sudo apt-get install libsdl1.2-dev

$ sed -i 's/-DENABLE\_VOLK=ON/-DENABLE\_VOLK=ON -DENABLE\_GR\_VIDEO\_SDL=ON/g' ~/.pybombs/recipes/gr-recipes/gnuradio.lwr ~/.pybombs/recipes/gr-recipes/gnuradio38.lwr

Now, let's create a prefix, or a directory in which to install all the GNU Radio components. In this case, we set the prefix directory to ~/pybombs/bladeRF. We also specify the alias, bladeRF, for easy reference later, and install the default GNU Radio setup.

$ mkdir ~/pybombs/

**Optionally**, to set GNURadio to dev branch 3.9 run the following two commands:

$ pybombs config --package gnuradio gitrev maint-3.9

$ pybombs config --package gr-osmosdr gitrev dev-gr-3.9

Then continue compiling GNURadio.

$ pybombs prefix init ~/pybombs/bladeRF -a bladeRF -R gnuradio-default

At this point, GNU Radio has been installed into ~/pybombs/bladeRF. Now, install useful applications such as bladeRF and gqrx:

$ pybombs -p bladeRF install bladeRF gr-iqbal gr-osmosdr gqrx

Once each component has been successfully built, they may be run using pybombs -p bladeRF run <command>, some examples follow:

$ pybombs -p bladeRF run bladeRF-cli -- -i

$ pybombs -p bladeRF run gqrx

Alternatively, the GNU Radio environment may be "sourced" into the active console, allowing applications to be run directly.

$ source ~/pybombs/bladeRF/setup\_env.sh

$ bladeRF-cli -i

$ gqrx

**PYTHONPATH Errors**

If any PYTHONPATH warning or errors appear while launching gnuradio-companion, ensure all of the Python package library paths in ~/pybombs/bladeRF/lib/ are present in $PYTHONPATH.

To have setup\_env.sh try to fix missing PYTHONPATH paths, first backup the setup\_env.sh then try:

$ echo "export PYTHONPATH=$(echo ~/pybombs/bladeRF/lib/\*/\*-packages | sed 's/ /:/g')":'$PYTHONPATH' >> ~/pybombs/bladeRF/setup\_env.sh

If everything went correctly, something similar should appear:

$ tail -n 1 ~/pybombs/bladeRF/setup\_env.sh

export PYTHONPATH=/home/user/pybombs/bladeRF/lib/python3.8/site-packages:/home/user/pybombs/bladeRF/lib/python3/dist-packages:$PYTHONPATH

Note: the paths should be absolute paths, and they should appear on the very last line of the setup\_env.sh file. If things match, try sourcing the file again source ~/pybombs/bladeRF/setup\_env.sh

If these outputs do not appear or do not match, it might be good to restore the backup of the setup\_env.sh file and manually add the missing Python package paths to the file.

**PyBOMBS udev rules**

You may get a permissions error when trying to access the bladeRF. This is typically from lack of, or incorrect, udev rule(s) and may be remedied by following the [udev installation instructions](https://github.com/Nuand/bladeRF/wiki/Troubleshooting#pybombs).

**PyBOMBS and gr-fosphor**

On recent Ubuntu systems, try the following steps to install gr-fosphor. This assumes the appropriate GPU drivers are already installed.

$ sudo apt install ocl-icd-opencl-dev

$ sudo apt-get install libfreetype6 libfreetype6-dev

$ pybombs -p bladeRF install bladeRF gr-fosphor

**Easy installation for Ubuntu: The bladeRF PPA**

If you're running **Ubuntu 20.04 (focal) or later**, a PPA is available at [ppa:nuand/bladerf](https://launchpad.net/~nuandllc/+archive/bladerf/+packages). New builds will appear occasionally as bladeRF releases (and release candidates) are tagged.

To activate the PPA, simply:

$ sudo add-apt-repository ppa:nuandllc/bladerf

$ sudo apt-get update

$ sudo apt-get install bladerf

If you plan to build gnuradio, gr-osmosdr, etc, you will also need the header files:

$ sudo apt-get install libbladerf-dev

Firmware and FPGA images can be installed from this PPA as well. Firmware should be manually updated using bladeRF-cli --flash-firmware /usr/share/Nuand/bladeRF/bladeRF\_fw.img, but the FPGA image will be automatically loaded by libbladeRF when you open your device.

$ sudo apt-get install bladerf-firmware-fx3 # firmware for all models of bladeRF

$ sudo apt-get install bladerf-fpga-hostedx40 # for bladeRF x40

$ sudo apt-get install bladerf-fpga-hostedx115 # for bladeRF x115

$ sudo apt-get install bladerf-fpga-hostedxa4 # for bladeRF 2.0 Micro A4

$ sudo apt-get install bladerf-fpga-hostedxa9 # for bladeRF 2.0 Micro A9

Note: All of the FPGA images can be installed simultaneously without issue.

Then, skip down to the [Checking basic device operation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux#wiki-Checking_basic_device_operation) section, bypassing the compilation steps.

Note: Ubuntu's repositories have an older version of the bladeRF libraries and tools, with the bladeRF-cli tool provided by a package named bladerf-host, instead of bladerf. If you've previously installed the Ubuntu repository version, the apt-get install bladerf step will remove/upgrade the older packages as appropriate.

**Skip down to "Checking Basic Device Operation" after performing these steps.**

**Installing dependencies**

**If you already installed the bladeRF software using apt-get above, you can skip this section.**

**NOTE:** If your distribution provides libusb-1.0 packages earlier than 1.0.16, you may want to consider to manually installing libusb >= 1.0.16, either from source or from upstream packages. Issues have been reported with earlier versions.

**Debian-based distros (e.g., Ubuntu)**

Install dependent packages for the build:

$ sudo apt-get install libusb-1.0-0-dev libusb-1.0-0 build-essential cmake libncurses5-dev libtecla1 libtecla-dev pkg-config git wget

You can check the libusb and libusb-dev versions installed on your system via:

$ dpkg -s libusb-1.0-0 libusb-1.0-0-dev

*Optional:* If you wish to build libbladeRF documentation and the bladeRF-cli man page, you'll need the following:

$ sudo apt-get install doxygen help2man pandoc

**Redhat-based distros (e.g., Fedora, CentOS)**

$ sudo yum groupinstall "Development Tools" "Development Libraries"

$ sudo yum install libusbx libusbx-devel cmake wget gcc-c++ libedit libedit-devel

*Optional:* If you wish to build libbladeRF documentation and the bladeRF-cli man page, you'll need the following:

$ sudo yum install doxygen help2man pandoc

**Building bladeRF libraries and tools from source**

**If you already installed the bladeRF software using apt-get above, you can skip this section.**

**Clone the bladeRF git repository**

To obtain the latest source code for the first time, clone the Nuand git repository via:

$ git clone https://github.com/Nuand/bladeRF.git ./bladeRF

$ cd ./bladeRF

$ ls

The directory contents will look something like this:

CHANGELOG CONTRIBUTORS debian fx3\_firmware host README.md

CMakeLists.txt COPYING firmware\_common hdl legal

In the future, you can update the repository via:

$ git pull

**Configure the build**

First enter the directory containing the host source. Then create and enter a directory to perform the build in. By working out of the a 'build' directory, it's easy to later clean up, by simply removing 'build/'.

$ cd host/

$ mkdir build

$ cd build

Next, configure the build. In the below example, we:

* Set up a Release build
  + Change to Debug for a debug build, or RelWithDebInfo for a release build with debug symbols
* Enable the installation of udev rules.
  + By default, this grants members of the *plugdev* group read-write access to bladeRF devices. This group can can be changed via -DBLADERF\_GROUP=desired\_group.
* Specify that files should be installed into /usr/local
  + You can install into a different location via the -DCMAKE\_INSTALL\_PREFIX=/some/desired/path option. Note that you'll need to need to configure binary and library search paths if you install elsewhere. If you're unsure how to do this, simply follow the steps below to use the default install location.

$ cmake -DCMAKE\_BUILD\_TYPE=Release -DCMAKE\_INSTALL\_PREFIX=/usr/local -DINSTALL\_UDEV\_RULES=ON ../

Ensure that your user is in the group specified by BLADERF\_GROUP. Check the output of the *groups* command:

$ groups

jon adm cdrom sudo dip plugdev lpadmin sambashare

In the above case, the user is already in the plugdev group, so we can skip ahead to the next subsection.

However, on other systems, the user may not already be in the required group. Furthermore, some distributions may not use the plugdev group. In these cases, you can either add a user to a more appropriate group or create a new group (and configure the build with -DBLADERF\_GROUP=desired\_group. For example, to create a *bladerf* group and add the user *jon* to it:

$ groups

jon wheel

$ sudo groupadd bladerf

$ sudo usermod -a -G bladerf jon

# Now log out and log back in...

$ groups

jon wheel bladerf

**Perform the build and installation**

The following commands:

1. Perform the build
2. Install files to /usr/local or the location specified by CMAKE\_INSTALL\_PREFIX
3. Updated share library paths, so that libbladeRF can be found

$ make && sudo make install && sudo ldconfig

**Uninstalling later**

Note that from this same directory, you can run the following to uninstall the files place on your system in the previous step.

sudo make uninstall

The *install\_manifest.txt* file, created after running the install step successfully, lists all the files installed. It is a good idea to back up this file if you plan on removing this build directory later.

**Checking basic device operation**

Before continuing, see the [Getting Started: Verifying Basic Device Operation](https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Verifying-Basic-Device-Operation) guide to test basic functionality.

If you encounter any issues or warnings, be sure to check the [Troubleshooting](https://github.com/Nuand/bladeRF/wiki/Troubleshooting) guide before continuing on with this article.

**Building GNU Radio, gr-iqbal, and gr-osmosdr**

The following procedure uses build-gnuradio to fetch dependencies and checkout some useful tools. However, it only covers installing GNU Radio and gr-osmosdr. A similar procedure maybe used to install the other tools.

See the [GNU-Radio wiki for more information](https://gnuradio.org/redmine/projects/gnuradio/wiki/InstallingGRFromSource), including the use of [PyBOMBS](http://gnuradio.org/redmine/projects/pybombs/wiki), which is now the installation method preferred by many users.

If you're not comfortable with the instructions presented below, please consider following the [PyBOMBS](https://github.com/gnuradio/pybombs#pybombs) instructions.

**GNU Radio >= 3.7.7 and VOLK**

As of GNU Radio v3.7.7, [VOLK](http://libvolk.org/) is now a separate project pulled in via a git submodule. This implies that one must use the --recursive flag when cloning the repository. (build-gnuradio and PyBOMBS do this for you.) If you are manually cloning the GNU Radio and forget to use the --recursive flag, you'll need to run git submodule init volk followed by git submodule update to initialize the VOLK subrepo.

**Multiple GNU Radio Installations**

Having multiple GNU Radio installations on your system is highly discouraged, unless you're aware of how to avoid mixups between library search paths and binary paths between the different installations.

If you are building GNU Radio from source, as shown below, ensure you do not have GNU Radio installed from your distribution's package manager or from a previous build. If you do, remove it before continuing.

**Using build-gnuradio to fetch dependencies and checkout source via Git**

$ mkdir -p ~/software/gnuradio-build

$ cd ~/software/gnuradio-build

$ wget http://www.sbrac.org/files/build-gnuradio

$ chmod +x ./build-gnuradio

$ ./build-gnuradio -m prereqs gitfetch

The build-gnuradio script will prompt you a few times before proceeding; type 'y' and press enter to continue. Note that this script takes a bit of time, depending on how many dependencies you need to have installed. Run top periodically to see what's currently going on.

**Compile GNU Radio**

$ cd ~/software/gnuradio-build/gnuradio/

$ git checkout -b gnuradio-v3.7.13.4 v3.7.13.4

$ mkdir build

$ cd build

$ cmake -DCMAKE\_INSTALL\_PREFIX=/opt/gnuradio-3.7.13.4 ../

$ make -j8 && sudo make install

**Add GNU Radio to linker path and executable path**

If you would like GNU Radio to be available in PATH for all uses, create a new file called /etc/profile.d/gnuradio.sh:

sudo vi /etc/profile.d/gnuradio.sh

Otherwise, if you would like GNU Radio executables to only be in your path, update your shell's runtime-config file, such as ~/.bashrc.

Put the following into it:

#!/bin/bash

# Add GNU Radio binaries to the search path

GNURADIO\_PATH=/opt/gnuradio-3.7.13.4

export PATH=$PATH:$GNURADIO\_PATH/bin

# Add GNU Radio python libraries to python search path

if [ $PYTHONPATH ]; then

export PYTHONPATH=$PYTHONPATH:$GNURADIO\_PATH/lib/python2.7/dist-packages

else

export PYTHONPATH=$GNURADIO\_PATH/lib/python2.7/dist-packages

fi

**Note:** Be sure to update GNURADIO\_PATH if you deviated from gnuradio-3.7.13.4

**Note:** If you're on a non-Debian based distro, you might need to replace "dist-packages" with "site-packages".

Again, with your favorite text editor, create a new file called `/etc/ld.so.conf.d/gnuradio.conf`:

sudo vi /etc/ld.so.conf.d/gnuradio.conf

Put this in the new file:

/opt/gnuradio-3.7.13.4/lib

**Note:** on some 64 bit systems, you might need a lib64 directory in this file as well.

* Update your library cache again

$ sudo ldconfig -v | grep gnuradio

You should see something akin to the following:

/opt/gnuradio-3.7.13.4/lib:

libgnuradio-analog-3.7.13.4.so.0.0.0 -> libgnuradio-analog.so

libgnuradio-trellis-3.7.13.4.so.0.0.0 -> libgnuradio-trellis.so

libgnuradio-fft-3.7.13.4.so.0.0.0 -> libgnuradio-fft.so

libgnuradio-fec-3.7.13.4.so.0.0.0 -> libgnuradio-fec.so

libgnuradio-qtgui-3.7.13.4.so.0.0.0 -> libgnuradio-qtgui.so

libgnuradio-channels-3.7.13.4.so.0.0.0 -> libgnuradio-channels.so

libgnuradio-wavelet-3.7.13.4.so.0.0.0 -> libgnuradio-wavelet.so

libgnuradio-pmt-3.7.13.4.so.0.0.0 -> libgnuradio-pmt.so

libgnuradio-atsc-3.7.13.4.so.0.0.0 -> libgnuradio-atsc.so

libgnuradio-video-sdl-3.7.13.4.so.0.0.0 -> libgnuradio-video-sdl.so

libgnuradio-vocoder-3.7.13.4.so.0.0.0 -> libgnuradio-vocoder.so

libgnuradio-fcd-3.7.13.4.so.0.0.0 -> libgnuradio-fcd.so

libgnuradio-zeromq-3.7.13.4.so.0.0.0 -> libgnuradio-zeromq.so

libgnuradio-blocks-3.7.13.4.so.0.0.0 -> libgnuradio-blocks.so

libgnuradio-audio-3.7.13.4.so.0.0.0 -> libgnuradio-audio.so

libgnuradio-dtv-3.7.13.4.so.0.0.0 -> libgnuradio-dtv.so

libgnuradio-pager-3.7.13.4.so.0.0.0 -> libgnuradio-pager.so

libgnuradio-noaa-3.7.13.4.so.0.0.0 -> libgnuradio-noaa.so

libgnuradio-filter-3.7.13.4.so.0.0.0 -> libgnuradio-filter.so

libgnuradio-digital-3.7.13.4.so.0.0.0 -> libgnuradio-digital.so

libgnuradio-runtime-3.7.13.4.so.0.0.0 -> libgnuradio-runtime.so

If so, you're all set!

For the changes to your PATH take effect, either:

* Log out and log back in
* Run $ source /etc/profile.d/gnuradio.sh or source ~/.bashrc, depending which file you added the PATH definition to, in order to update the environment in your current terminal

Once you have logged back in to your desktop, you should be able to run gnuradio-companion.

If you get a pop-up with errors about *PYTHONPATH* or *LD\_LIBRARY\_PATH*, check the output of the env command to see how these environment variables are currently set.

**Build and install gr-iqbal**

This step is optional, but will make additional IQ balance functionality available for gr-osmosdr.

$ cd ~/software/gnuradio-build/gr-iqbal

$ mkdir build

$ cd build

$ cmake -DCMAKE\_INSTALL\_PREFIX=/opt/gnuradio-3.7.13.4 ../

$ make -j8 && sudo make install && sudo ldconfig

**Build and install gr-osmosdr**

[gr-osmosdr](http://sdr.osmocom.org/trac/wiki/GrOsmoSDR) was checked out during the build-gnuradio execution. This provides GNU Radio support for a number of devices, including the bladeRF.

$ cd ~/software/gnuradio-build/gr-osmosdr

$ mkdir build

$ cd build

$ cmake -DCMAKE\_INSTALL\_PREFIX=/opt/gnuradio-3.7.13.4 ../

$ make -j8 && sudo make install && sudo ldconfig

**Simple test applications to verify device operation**

The osmocom programs may be used to quickly view an FFT and generate a tone.

Test Receive: (446 MHz)

osmocom\_fft -a bladerf=0 -s 2500000 -f 446000000

Test Transmit: (446 MHz + 25 KHz )

osmocom\_siggen -a bladerf=0 -s 2500000 -f 446000000 -g 4 --sine -x 25000

Note that only one program can have the bladeRF opened at a time.

If you'd like to run a quick test in a full-duplex configuration, consider creating a GRC flowgraph.

*Do not connect the bladeRF TX output to the RX input without an attenuator.* If you do not have an attenuator, consider using the bladeRF's (technically, the LMS6002D's) internal baseband and RF loopback modes, via the [*loopback=*](http://sdr.osmocom.org/trac/wiki/GrOsmoSDR#bladeRFSourceSink) argument to the osmocom sink/source block.

**Build and install GQRX**

[GQRX](http://gqrx.dk/) is a software defined radio receiver powered by GNU Radio and Qt. The bladeRF is supported via gr-osmosdr. The latest and greatest of GQRX can be built from source as follows, assuming you have Qt5 libraries (and development headers) already installed: (Note: On Debian-based distros, you will need to sudo apt-get install libqt5svg5-dev if you don't already have it. On Ubuntu you may need to sudo apt-get install qt5-default for QT5 libraries, if you don't already have them. Ensure that you have qt5-qmake installed.)

$ cd ~/software/gnuradio-build/

$ git clone https://github.com/csete/gqrx.git

$ cd gqrx

$ mkdir build

$ cd build

$ cmake -DCMAKE\_INSTALL\_PREFIX=/opt/gnuradio-3.7.13.4 ../

$ make -j8 && sudo make install && sudo ldconfig

When first running gqrx (or after running gqrx -e), you will be presented with a dialog to configure a device. Below are some suggested settings to start:

**Device String:** bladerf=0,buffers=128,buflen=8192,transfers=32

This selects the first available bladeRF, and adds some additional buffering.

**Sample rate:** 5000000

5Msps is a good starting point, as this should work on both USB 2.0 and USB 3.0 connections.

**Bandwidth:** 3.84 MHz

The bandwidth should always be set less than the sample rate to minimize aliases and prevent noise from folding in. See [Figure 6 in LMS6002D](http://www.limemicro.com/download/LMS6002Dr2-DataSheet-1.2r0.pdf) datasheet to determine how low to set the bandwidth to ensure you get maximal rejection at the bandwidth limits governed by your sample rate. (Note that the plots' X axes are from 0 to Fs/2, not -Fs/2 to Fs/2).

**Tips**

* Note that the *Receive Options* tab allows you to enable various demodulators
* Use the *Squelch* under *Receiver Options* to mute auto below a certain threshold
* The *FFT settings* tab allows you to increase the FFT size and speed up the FFT and waterfall
* If audio is choppy, try reducing your sample rate and bandwidth, the FFT size or rate, or all of these.