UHD

Contents

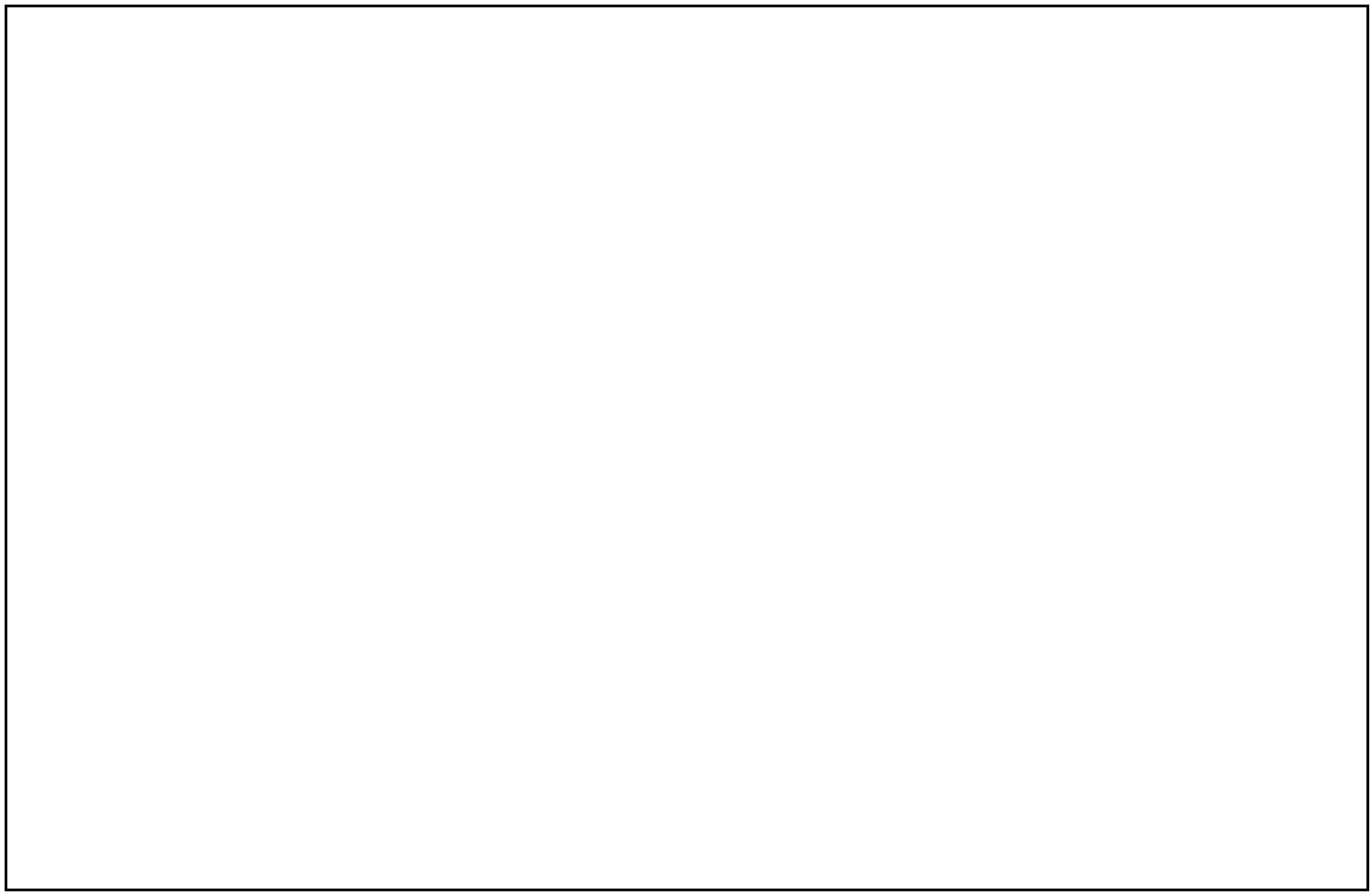
- 1 Introduction
- 2 Open Source
- 3 Release Cycle
- 4 Code Portability
- 5 Dependencies
  - 5.1 Python
  - 5.2 Linux Notes
  - 5.3 Mac OS X Notes
  - 5.4 Windows Notes
  - 5.5 Git
  - 5.6 Build Dependencies
- 6 Sample streaming
- 7 RF Frequency Fine Tuning
- 8 Sample Rate
- 9 IQ Corrections
- 10 Multiple USRPs
- 11 GPIO ? General Purpose Input Output
- 12 RF Front End Expansion
- 13 GNU Radio
- 14 Licensing
- 15 Conclusion
- 16 Additional Resources

The Ettus Research USRP? family of Software Defined Radios (SDRs) are versatile devices that allow users to transmit and receive many different and custom waveforms at various frequencies and settings on a common hardware platform. Commercial, academic, and military customers employ the flexible and reusable USRP hardware for research, wireless exploration, and field deployments.

The USRP comes in many form factors and configurations. The main categories of USRPs are:

- Bus (B) Series ? Connected to a host computer via a USB connection
- Network(N) Series ? Connected to a host computer via an Ethernet connection
- High Performance (X) Series ? Connection can be Ethernet or a x4 PCI-Express connection
- Embedded (E) Series ? Meant to run stand-alone (without a host computer).

The USRP Hardware Driver (UHD) is a user-space library that runs on a general purpose processor (GPP) and communicates with and controls all of the USRP device family. The B, N and X series USRPs send and receive samples from a host computer as illustrated in figure 1. And since our embedded series USRPs house an internal GPP you can run these radios without the need of a host computer (stand-alone mode).



USRPs are transceivers, meaning that they can both transmit and receive RF signals. UHD provides the necessary control used to transport user

waveform samples to and from USRP hardware as well as control various parameters (e.g. sampling rate, center frequency, gains, etc) of the radio.

UHD GPP driver and firmware code is written in C/C++ while the code developed for the FPGA (Field Programmable Gate Array) is written in Verilog. There is a C/C++ API that can interface to other software frameworks, as in the case of GNU Radio, or a user can simply build custom signal processing applications directly on top of the UHD C/C++ API. Figure 2 illustrates this concept:



Figure 2. UHD components

The majority of the UHD code base is open source, including code that executes on the host, as well as code targeted to the USRP hardware (FPGA and microcontroller firmware). As dual-licensed software, UHD is available under the open-source GNU Public License version 3 (GPLv3), as well as an alternate, less-restrictive license offered only by Ettus Research.

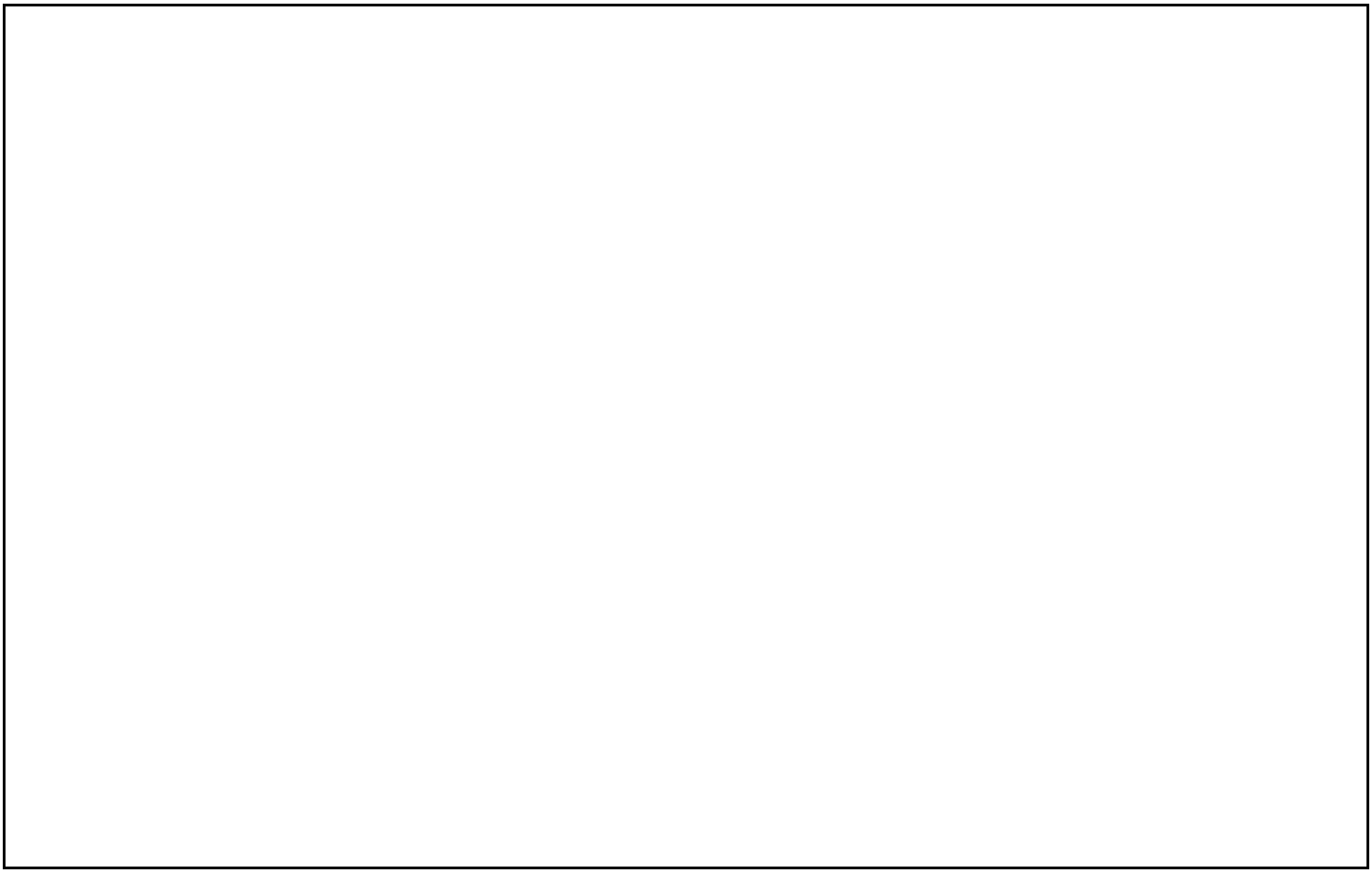
Due to the open source nature of UHD, the entire development process is also open, and it is possible to track UHD?s development through our Git version control system. Users can make a choice if they prefer the latest development code, which is most feature-rich but can be unstable at times, or the more thoroughly tested code that does not include the latest development. Versioned releases (e.g., UHD version 3.9.2) happen approximately every two months, and usually only include bugfixes compared to the previous version (e.g., UHD 3.9.2 has the same feature set as 3.9.1, but is more stable). These bugfix releases are done off of the maint branch, which is where all the bugfix development happens.

New feature development is done on the master branch. This branch should not be considered stable, even though we use continuous integration systems to monitor its state. However, it is possible that APIs or dependency requirements change on the master branch.

At the end of a feature development cycle, the master branch is frozen, and only bugfixes are accepted into the master branch. Once the master branch is considered stable, the maint branch is reset to master, and a new versioned release is produced from the previous master branch. This is indicated by a major version number jump (e.g., when going from 3.8.3 to 3.9.0). Major version releases occur 1-2 times per year, and usually accompany new product releases.

Aside from the maint and master branches, new feature branches are sometimes published (temporarily) to give insight into upcoming features, and earlier access to new development.

The UHD software API supports application development on all USRP SDR products. Using a common software interface is critical, enabling code portability and, allowing applications to transition seamlessly to other USRP SDR systems when development requirements expand or new systems become available. Hence, it enables a significant reduction in development effort by allowing you to preserve and reuse your legacy code. UHD can be installed on Linux, Windows, or a Mac. Installation packages for these platforms as well as instructions for building from source can be found here.



Compatible with Python 2.7 and above (Python 3 is supported). Note Python is required for building, and some utilities that ship with UHD are written in Python. It is not, in general, required for running UHD applications.

This is dependent on the distribution you are using, but most, if not all, of the dependencies should be available in the package repositories for your package manager. See the Linux build instructions for more information.

Install the Xcode app to get the build tools (GCC and Make). Use MacPorts to get the Boost and Mako dependencies. Other dependencies can be downloaded as DMG installers from the web or installed via MacPorts. See the UHD OS X build instructions for more information.

The dependencies can be acquired through installable EXE files. Usually, the Windows installer can be found on the project's website. Some projects do not host Windows installers, and if this is the case, follow the auxiliary download URL for the Windows installer (below). See the Windws build instructions for more information.

- Required to check out the repository (not necessary if building from tarballs).
- On Windows, install Cygwin with Git support to checkout the repository or install msysGit from http://code.google.com/p/msysgit/downloads/list.

UHD Version	GCC	Clang	MS Visual C++	CMake	Boost	LibUSB	Mako	Doxygen	Python	Xilinx Vivado
3.9.X	>= 4.4	>= 3.3	>= 2012 (11.0)	>= 2.8	>= 1.46	>= 1.0	>= 0.5.0	>= 1.8 (recommended)	>= 2.7	== 2014.4
3.10.X	>= 4.8	>= 3.3	>= 2012 (11.0)	>= 2.8	>= 1.53	>= 1.0	>= 0.5.0	>= 1.8 (recommended)	>= 2.7	== 2015.4

Other compilers (or lower versions) may work, but are unsupported.

Note on MSVC: The free Visual Studio Express Edition for Desktop works.

Note on Boost: Older versions of UHD may not work with newer versions of Boost, as the Boost API sometimes changes, and we can't retroactively fix older UHD versions. For example, UHD 3.8.X (and older) won't work with Boost 1.60.

UHD handles the control for transporting I and Q samples (see here for information on I and Q samples) by using standard interface methods such as Ethernet, USB and PCI-Express. Samples can be sent in a continuous stream as in figure 4a or in bursts, figure 4b. In addition the user has the ability to specify when samples are received or transmitted by using built in burst and timed commands.

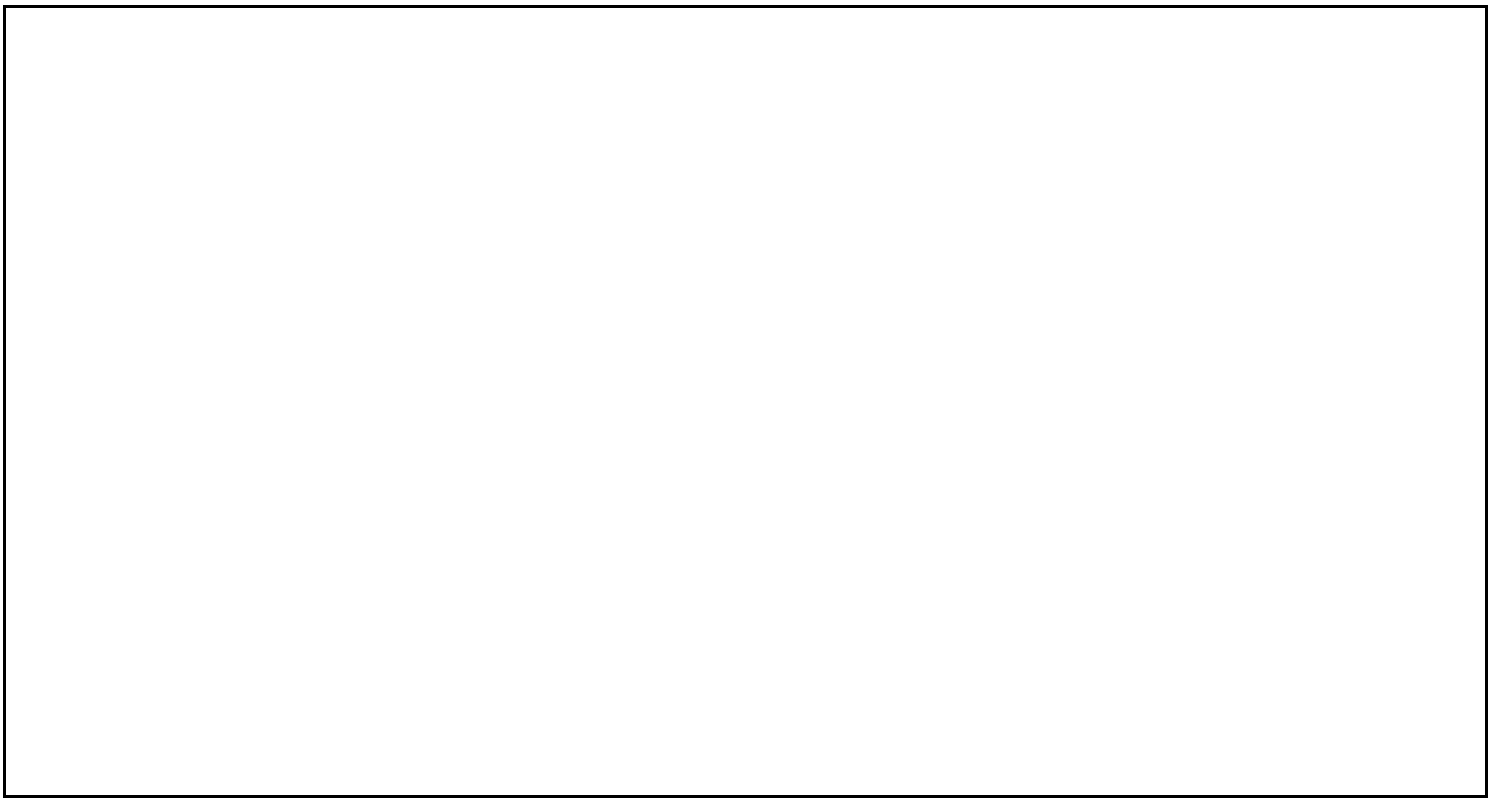
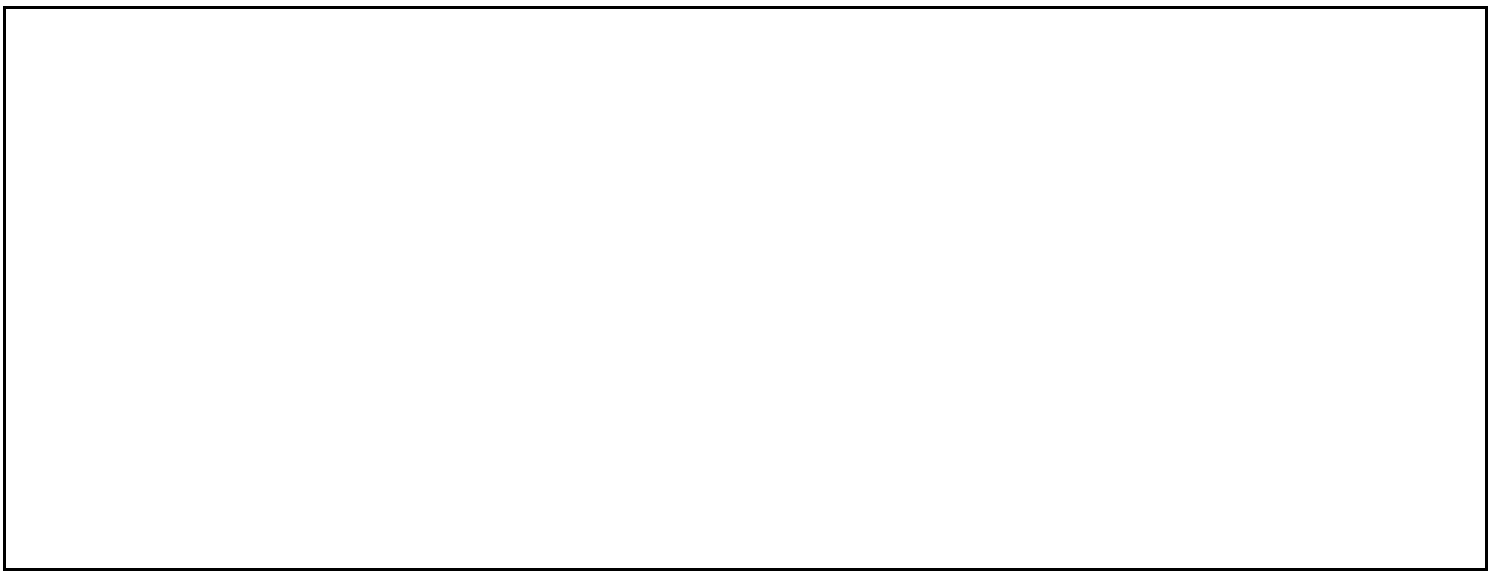


Figure 4. a) Continuous Sample Streaming. b) Burst Sample Mode (SOB = Start of Burst while EOB = End of Burst).

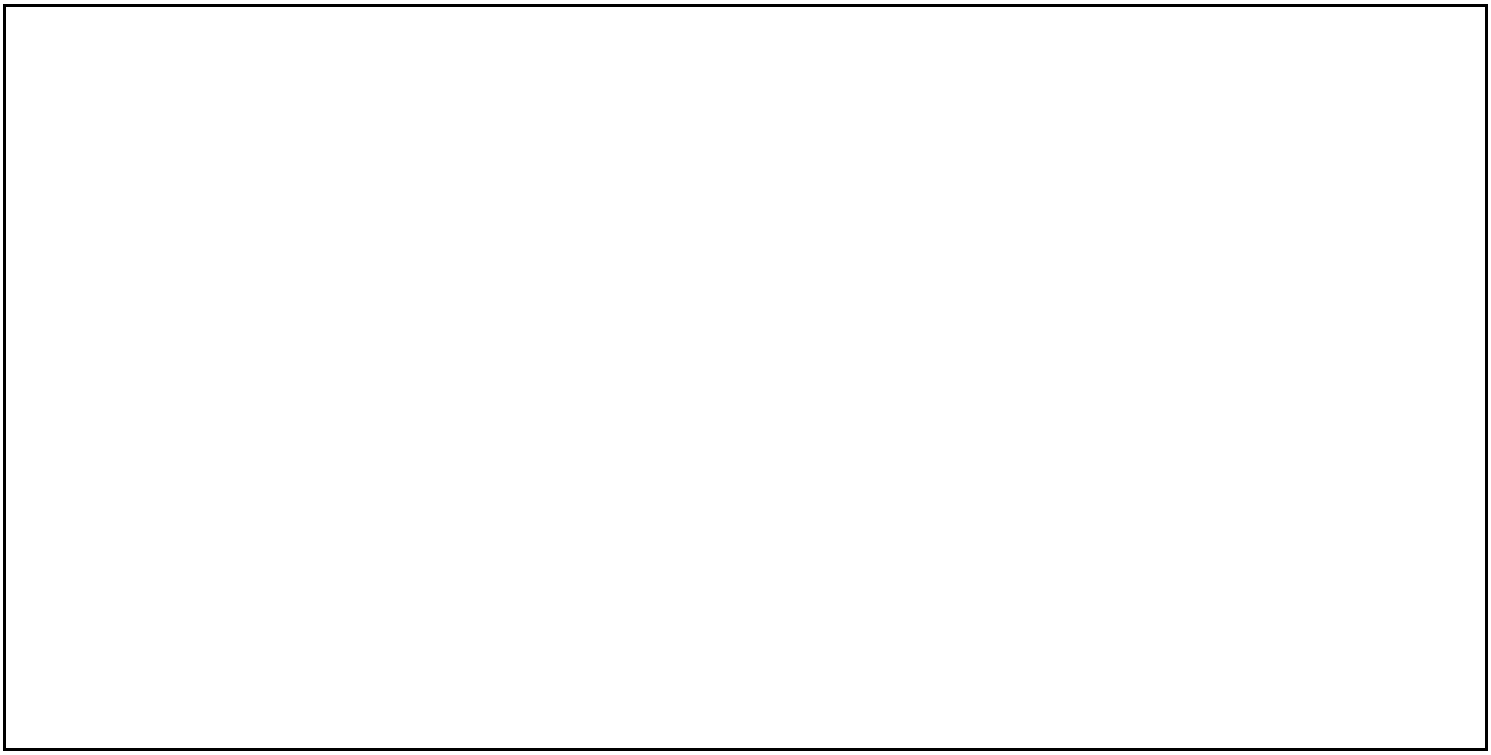
USRP RF front ends may support a certain frequency step size that does not meet all or many of a user?s requirements. For this reason, UHD includes Digital Up-Conversion (DUC) and Digital Down-Conversion (DDC) DSP blocks in the FPGA for fine tuning the RF frequency (see here). This allows users to:

- Have a sub-Hz RF frequency step size
- Mitigate the DC problem that exists on Direct Conversion (Zero IF) hardware (see here).
- Fast tune inside the available bandwidth

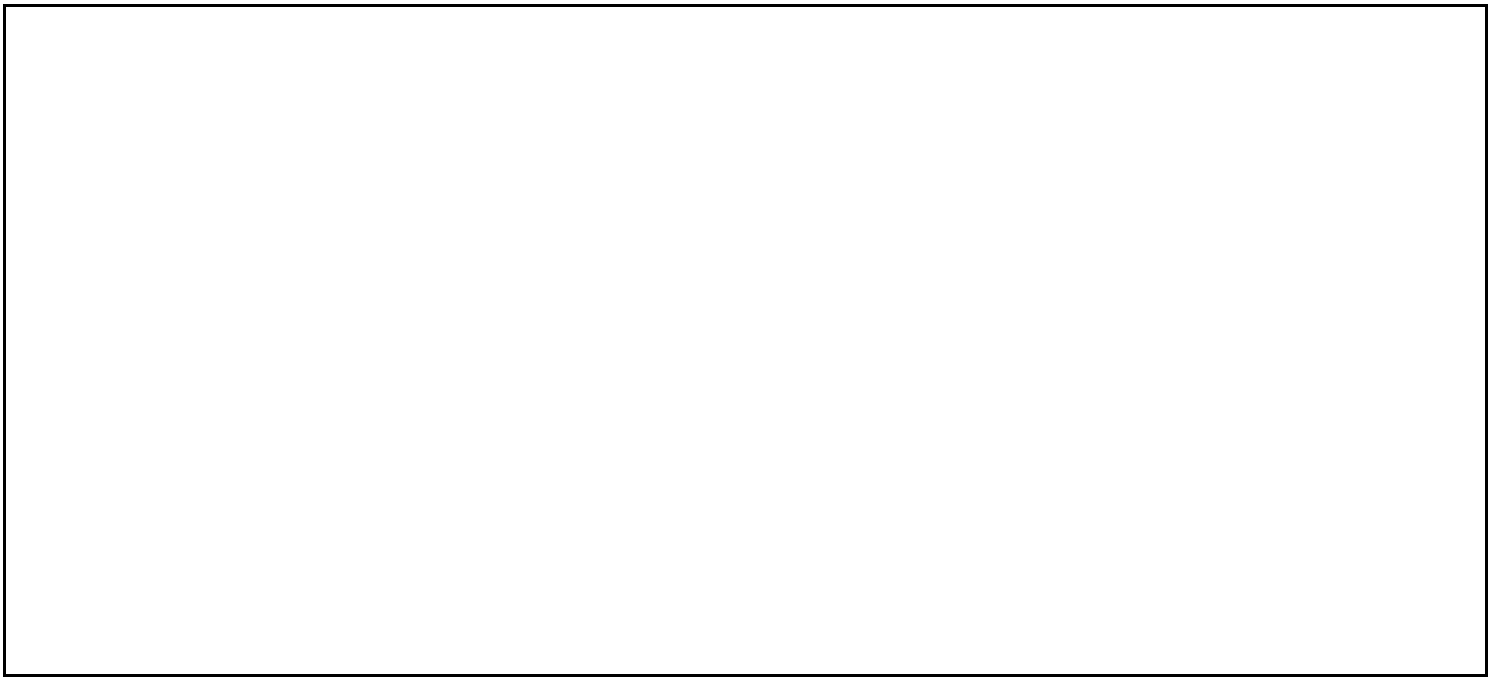


Different SDR configurations, waveforms, and applications require different sample rates. For example, a user may wish to monitor 100 MHz of instantaneous RF bandwidth, but their host PC may only be capable of analyzing 20 MHz of real-time bandwidth. For this and other cases, UHD allows users to set various sample rates to meet their custom application. Within the FPGA, UHD includes decimation and interpolation blocks in order to perform these sample rate translations.

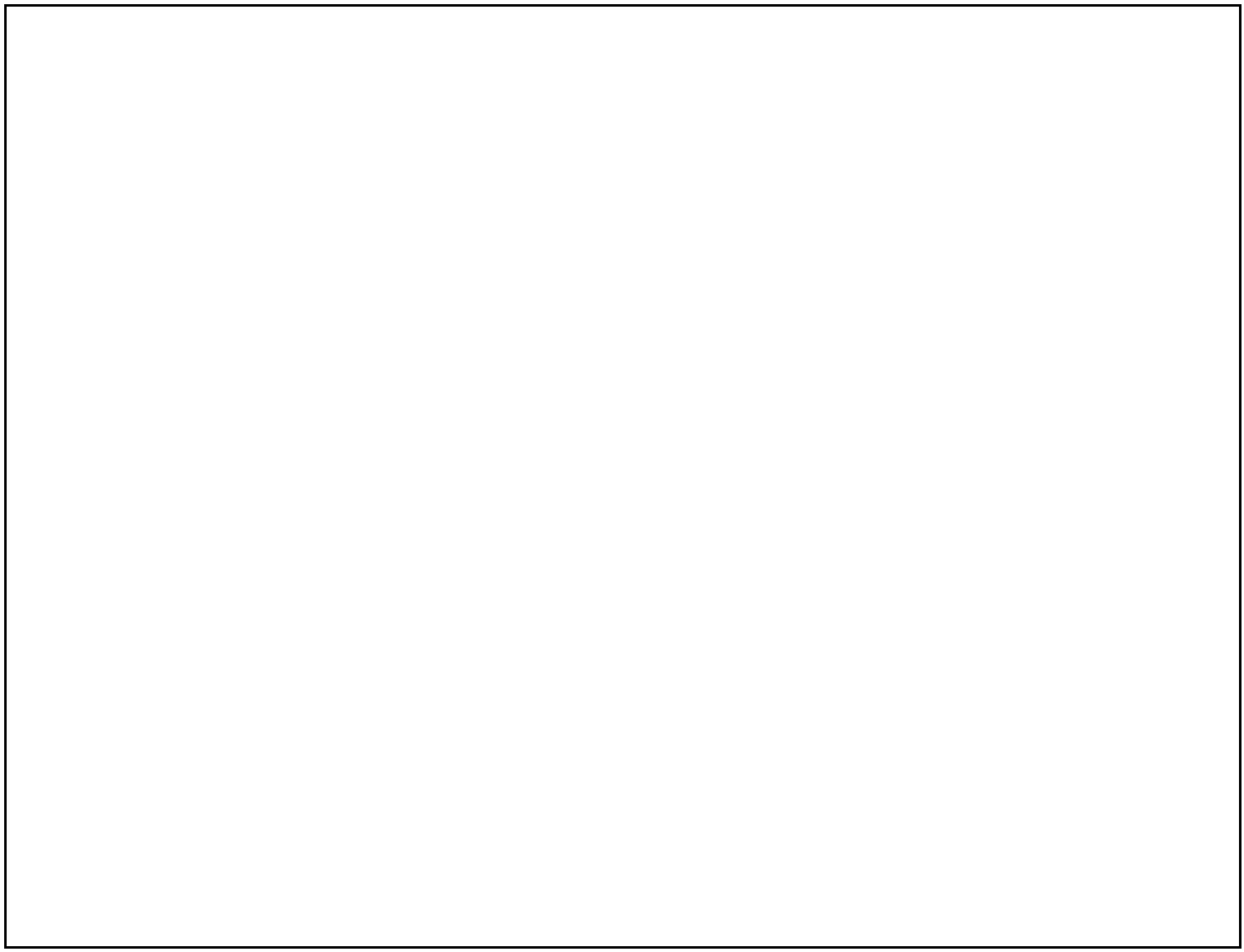
Inherent to all direct conversion (zero IF) RF architectures is the effect of IQ imbalance. A video entitled ?Effects of Quadrature Impairments on 802.11ac EVM?, found here, demonstrates this attribute. Find additional information here. In summary, any given signal on a device using a direct conversion architecture has two paths for TX and two paths for RX. One path is the In-phase or ?I? and the other is Quadrature ?Q?, also referred to as Real and Imaginary.



Because of slight differences due to variances in components, temperature, and other factors, the I and Q signal paths are subject to different conditions, thus altering the original signal that existed at initial capture. When the phase or amplitude of either the I or Q get altered the results show up as signals not actually present in the original signal. For example, note the following IQ impairment simulation in this GNU Radio Flowgraph created in GNU Radio Companion.



By using the ?HW Impairments? block, you can simulate what happens when the I and Q signal paths are disrupted. In figure 8a there are no impairments added. However, when the amplitude of the I path is increased slightly, you get the artifact as show in figure 8b



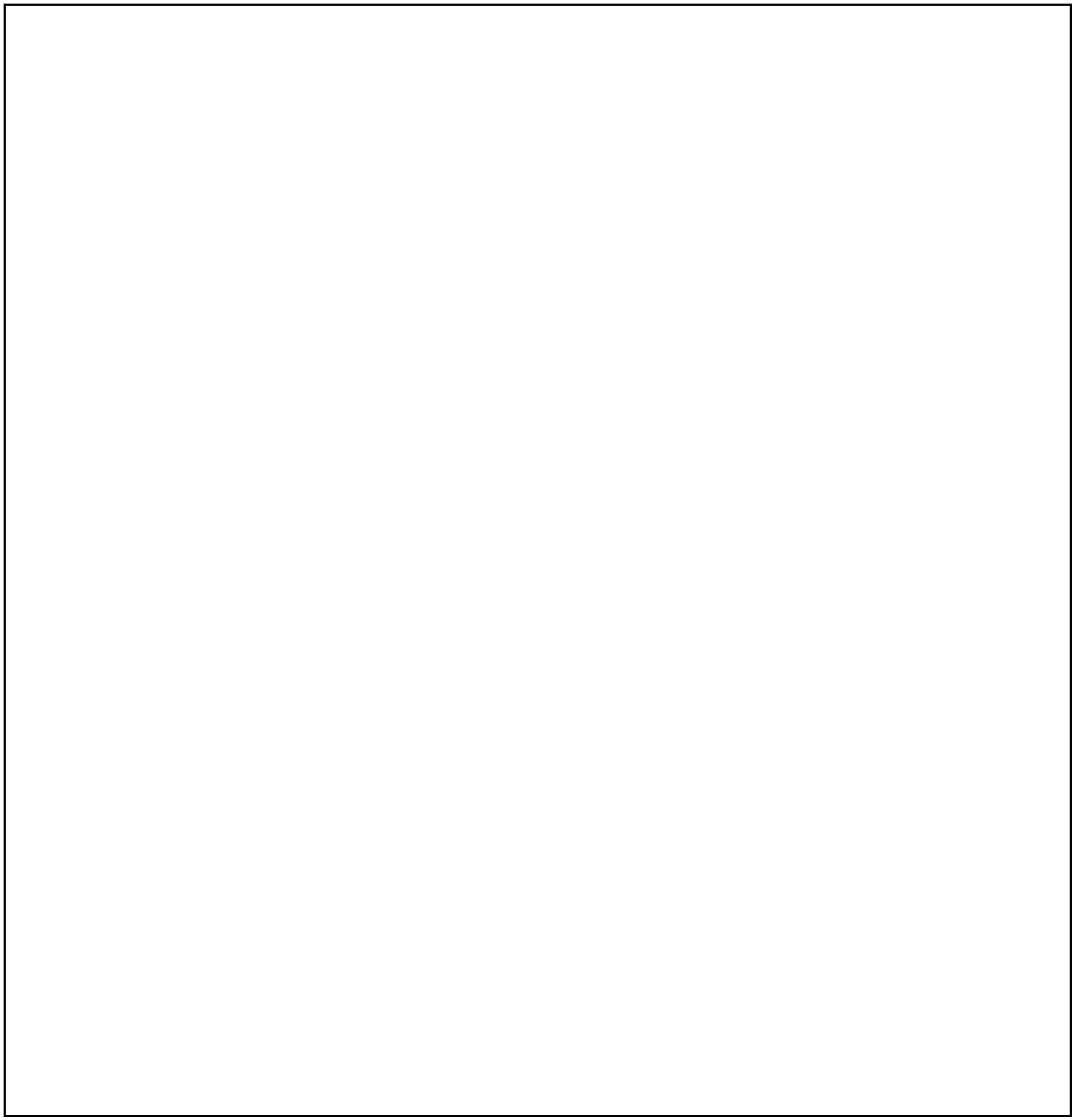
It is possible to mitigate some of these unwanted effects in the digital domain. UHD contains function blocks within the FPGA to compensate for IQ impairments; these blocks appeared in figure 1 as IQC blocks. A user can customize the parameters of these blocks based on empirical measurement or allow UHD to perform an automatic analysis and provide parameters based on a built-in IQ correction algorithm. See the following functions in the UHD manual for more information:

- uhd_cal_rx_iq_balance: - mimimizes RX IQ imbalance vs. LO frequency
- uhd_cal_tx_dc_offset: - mimimizes TX DC offset vs. LO frequency
- uhd_cal_tx_iq_balance: - mimimizes TX IQ imbalance vs. LO frequency

UHD makes scaling the number of channels on a USRP system simple by treating them all as one composite device, see figure 9 below. In the case of the X300/X310, when USRPs are used in this multi-USRP configuration users can use an external clocking source (having both a 10 MHz clock reference and a PPS signal) to synchronize all the devices together.



As an example, in an 8 channel receive system the ?USRP Source? block in GNU Radio Companion would look like figure 10 below



General Purpose Input Output (GPIO) pins can be controlled manually through UHD or can be set from UHD to be automatically triggered when events such as TX or RX occur. An example of where or how to use this automatic triggering: when a user connects an RF amplifier to the TX or RX port of the USRP, the amplifier can be powered on only when the USRP is transmitting. You can find out more about the aux GPIO on the E3x0/X3x0 here.



While mostly a function of HW, the USRP can interface with different RF front ends from 3rd parties. By using the BasicRX and BasicTX daughterboards, a user can send or receive a Baseband or IF (Intermediate Frequency) analog signal to 3rd party front ends.

In addition to leading development of the gr-uhd module that integrates the USRP family (UHD) into GNU Radio, Ettus Research frequently contributes to the GNU Radio project. Although USRP radios are often associated with the GNU Radio software framework, many USRP applications (such as OpenBTS) run without GNU Radio. For more information on GNU Radio see here.

- Licensing FAQ

UHD aids commercial, academic, military, hobbyists and other SDR users with rapid prototyping or deployment of wireless protocols by providing a flexible feature rich SDR API.

Github: https://github.com/EttusResearch/uhd

Manual: https://files.ettus.com/manual/index.html