Ultra-wideband SDR architecture for AMD RFSoCs and PYNQ based GNU Radio blocks

Marius Šiaučiulis MARIUS.SIAUCIULIS@STRATH.AC.UK
Louise H. Crockett
Robert W. Stewart
Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK

Abstract

The AMD RFSoC (Radio Frequency System on Chip) architecture has gained significant attention within the Software Defined Radio (SDR) community for its integration of Radio Frequency (RF) frontend, FPGA fabric and Linux-capable Arm-based processing system in a single package. Despite its accessibility to researchers via the RFSoC 2x2 and RFSoC 4x2 development board platforms, its adoption within the GNU Radio community has been limited. This work demonstrates the potential of combining RFSoC with GNU Radio by utilizing a bidirectional QSFP network link to transmit and receive a wideband Orthogonal Frequency-Division Multiplexing (OFDM) signal. Using the remote procedure calls we are able to control the Tx/Rx center frequency and RFSoCs Digital Up/Down Converter (DUC/DDC) rates from the host PC to achieve runtime configurable bandwidth. Additional signal inspection and visualisation is implemented using existing GNU Radio GUI widgets and analysis blocks.

1. Introduction

In recent years, advances in Analogue-to-Digital and Digital-to-Analogue converter (ADC and DAC) technology have increased achievable sampling rates to multiple Giga Samples per Second (GSPS). These high sampling sampling rates help to enable diverse applications including radio astronomy [1][2], wireless communications [4], instrumentation [3], and quantum computing research [5][6]. In each case the data converters are interfaced with processing resources suitable for the task, which may include general purpose Central Processing Units (CPUs), and often the parallel processing capabilities of a Field Programmable Gate Array (FPGA), or FPGA-based System on Chip (SoC). In this paper, we specifically consider the AMD Zynq UltraScale+ Radio Frequency SoC (RFSoC), which has the added advantage of combining high speed ADCs and DACs with FPGA logic, a processing system, and other supporting resources on a single integrated device.

Developing systems for SoC devices in general can be challenging for non-specialists, due to the learning curve of the design tools and processes. Enabling access to these powerful platforms from the user's familiar domain-specific tools widens access to them, and is an important consideration of this research. This paper focuses on Software Defined Radio (SDR) applications, and demonstrates how the widely adopted and popular GNU Radio tool can be used in conjunction with the RFSoC device and the PYNQ software/hardware framework (further introduced in Section 2) to form a flexible prototyping environment. GNU Radio is the most popular Free and Open Source Software (FOSS) SDR development toolkit, and it has enabled a plethora of academic and industrial research endeavors to date, due to its inherent flexibility and ease-of-use. In this work, we specifically consider applications requiring high rate data transfer from / to the RFSoC device.

This work aims foster greater adoption of the RFSoC platform by the GNU Radio community, by providing a reference design targeting the AMD RFSoC 4x2 board [7]. The RFSoC 4x2 is a modestly priced board with four receive and two transmit channels, which is extensively supported via open-source hardware and software [8], reference and demonstration designs [9], a book [10], and other learning and supporting materials [11].

Other tooling options are available for the design of SDR systems based on RFSoC and related platforms. One of these is MathWorks Communications Toolbox with Zynq SDR Support [12], which offers radio-in-the-loop capabilities enabling quick iterations during the prototyping stage. In the research community, the CASPER [13] project is an example of a specialist community developing their own custom RFSoC-targeting tools, built upon MathWorks Simulink and Python, to best serve their processing needs.

Proceedings of the 13^{th} Annual GNU Radio Conference,

However, these proprietary tools may not fall within the financial budget of all potential users, or they may wish to take a different approach.

This paper presents a novel design that leverages the Quad Small Form-factor Pluggable (QSFP) network interface to establish an up to 100GBit/s bi-directional data link between GNU Radio, running on a Personal Computer (PC), and the RFSoC 4x2 for arbitrary radio signal transmission and reception. The QSFP network is used to send signals generated on the PC to RFSoC where they are interpolated and mixed to Radio Frequency using the integrated Digital Up Converter (DUC), to be transmitted via the RF Digital to Analogue Converter (RF-DAC). The RF-DAC is connected via loopback cable to RF Analogue to Digital Converter (RF-ADC) where the RF signal is sampled, decimated and demodulated using a Digital Down Converter (DDC). The Complex IQ (In Phase and Quadrature) samples are then sent back to the PC over the QSFP data link. Due to the high speed RF-ADCs and RF-DACs found on the RFSoC platform, a QSFP network interface is necessary to facilitate transmission of signals generated off-board.

The remainder of this paper is organised as follows. In Section 2, the RFSoC 4x2 board and the PYNQ framework are introduced, as well as a high level overview of the design architecture is presented. Section 3, gives a walkthrough of the hardware architecture. Section 4 considers the software architecture distributed between the board and PC implementations. Section 5 discusses the experimental results. The current status of PYNQ based GNU Radio block development is given in Section 6. Finally, in Section 7, the conclusions are drawn.

2. System Architecture

This section gives a quick introduction to the AMD RFSoC 4x2 development board and the AMD PYNQ framework. It then provides a high level overview of the design architecture and hardware setup.

2.1. RFSoC 4x2

The RFSoC 4x2 board (Figure 1) has been developed by AMD [15] in partnership with Real Digital [16] as a successor to the previously available RFSoC 2x2 board which was based on Gen 1 RFSoC architecture (the RFSoC 4x2 features a Gen 3 RFSoC, ZU48DR device). In addition to Zynq™ UltraScale+™ based FPGA Programmable Logic (PL) and an Arm-based applications Processing System (PS), the RFSoC 4x2 incorporates 4x 14-bit RF-ADCs and 2x 14-bit RF-DACs with maximum sampling rates of 5 Giga-Samples per Second (GSPS) and 9.85 GSPS, respectively [17]. 10 MHz to 10 GHz rated RF baluns have been incorporated into the Printed Circuit Board (PCB) of the development platform. This integration facilitates direct connection of antennas or other external signal sources and sinks through SubMiniature version A (SMA) coaxial connectors, thereby streamlining the prototyping process. A QSFP28 Ethernet interface is also present on the board that supports various network configurations with maximum throughput of 100Gbit/s.

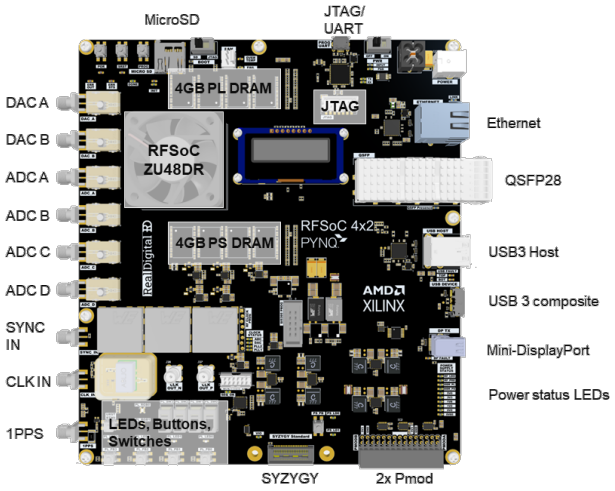

Figure 1. RFSoC 4x2 development board

2.2. PYNQ

AMD PYNQ [14] is an open-source project for System on Chip (SoC) devices, that exposes Intellectual Property (IP) cores within the FPGA fabric as a collection of Python objects. It provides Python Application Programming Interfaces (APIs) to conveniently load and manage FPGA bitstreams (called overlays), discover user controllable IPs, handle PL interrupts, and interact with memory mapped interfaces. Additionally, it provides drivers for commonly used IP blocks such as a Direct Memory Access (DMA) IP that can be used for large data transfers between PS and PL. With the release of the RFSoC 2x2 board, PYNQ was extended to include Python wrappers for RFSoC-specific functionality:

- xrfclk - to configure external RF reference clocks
- xrfdc - controlling and interacting with the RF Data Converters (RFDCs)
- xsdfec - drivers for Soft-Decision Forward Error Correction (SD-FEC) integrated block

Operating System (OS) images for officially supported boards [19], made by AMD and their partners, come with JupyterLab [18] web-based interactive development environment pre-installed, and ready to use out of the box. PYNQ can also be easily ported to other ZYNQ (*arm*) and ZYNQ Ultrascale+ (*aarch64*) based boards [20] making it a platform agnostic framework.

2.3. Architecture overview

The prototype platform is based on a Commercial Off-The-Shelf (COTS) workstation PC with a QSFP Network Interface Card (NIC) connected to the RFSoC 4x2 development board via an optical cable. GNU Radio flowgraph running on the PC is used to generate a baseband signal that is sent over the QSFP network using User Datagram Protocol (UDP). The RF-DAC channel B can be connected via loopback cable to RF-ADC channel B to facilitate flexible bandwidth signal introspection and visualisation using a Graphics Processing Unit (GPU) accelerated *gr-fosphor* [21] Out-Of-Tree (OOT) module. If connected to RF-ADC channel C, the received signal is decimated by a factor of 40 using the RFSoC's integrated DDC to reduce the amount of captured data and make the subsequent demodulation process less resource intensive. A high level overview diagram of the prototype platform is shown in Figure 2.

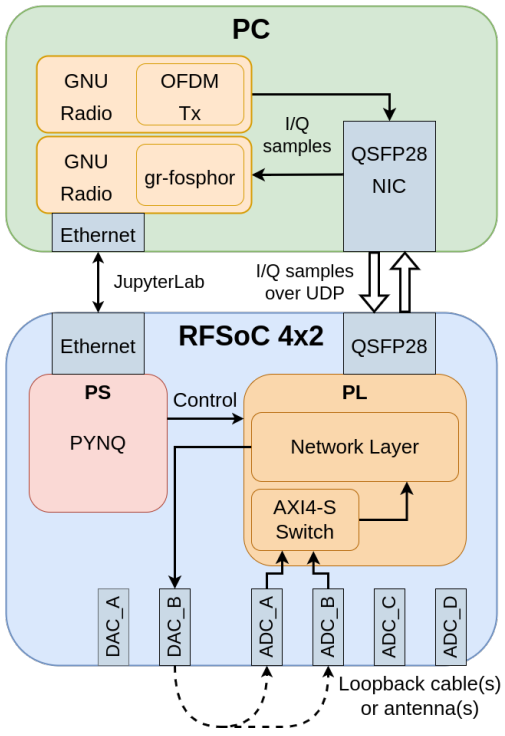

Figure 2. High level overview of the prototype platform

3. Hardware Architecture

In our previous work [22] we presented an open-source hardware and software design for high speed RF sample offload from RFSoC to a PC using the QSFP network interface. This work extends the previous design to include a receive channel that is used to drive the RF-DAC.

The QSFP network interface was built utilising the UltraScale+ Integrated 100G Ethernet Subsystem, which provides the Physical and Data link Open Systems Interconnection (OSI) model layer functionality. This includes the Ethernet Media Access Controller (MAC), Physical Coding Sublayer (PCS) and Reed-Solomon Forward Error Correction (RS-FEC) components. The functionality of the upper OSI layers is achieved using the open-source Network Layer kernel [23] IP, which includes Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP) and UDP transport layer capabilities. The aforementioned cores, alongside supporting architecture and Clock Domain Crossing (CDC) data First In, First Out (FIFO) buffers are grouped into a design hierarchy to facilitate straightforward reuse of the network subsystem.

The receiving side of the Network Layer kernel IP parses the received packets and compares the socket information to the socket table. If the socket information matches the socket table entry, the packets are considered valid and are passed to the AXI4-Stream interface. If the socket information does not match the socket table entry, the data packets are still streamed over the AXI4-Stream link, but the TVALID signal is kept low indicating they are invalid. As the GNU Radio UDP sink block chooses a random available port on each run, the socket table needs to be updated with the sender's port number during runtime.

In this design, the data packets received from the QSFP network are passed to a custom *netlayer_tuser* IP that extracts sender's and receiver's IP addresses, as well as ports, from the AXI4-Stream TUSER data path, and presents them to the PS over the AXI4-Lite interface. Additionally the IP can be used to override the TVALID signal and force the packets to become valid as a way to overcome the issue of random port selection. Once the packets are considered valid, they are passed to the RF-DC core where a Digital Up Converter (DUC) interpolates the signal from 61.44 MSPS to the RF-DAC sampling rate of 4915.2 MSPS (a factor of 80). Gen 3 RFSoC DUCs are capable of interpolation rates of up to x40, thus a supplementary Image Reject (IMR) stage is required to achieve the additional x2 interpolation.

RF-DAC channel B is configured as the output in this design and, due to the board's integrated RF baluns, it can drive an antenna or loopback cable directly. Optional analogue filters and variable gain amplifier can be chosen to condition the output signal.

As in our previous design, RF-ADC channel B is configured as a flexible bandwidth sampler for signal introspection using a remote gr-fosphor based flowgraph. It is capable of capturing RF signals at a maximum sampling rate of 2457.5 MSPS and a minimum of 307.2 MSPS. Considering the transmitted wideband OFDM signal is around 60 MHz wide even at the lowest sampling rate of 307.2 MSPS, too much unnecessary data is captured for efficient signal demodulation using GNU Radio. An additional RF-ADC Channel C is enabled with its Digital Down Converter (DDC) set to x40 decimation to allow reception of

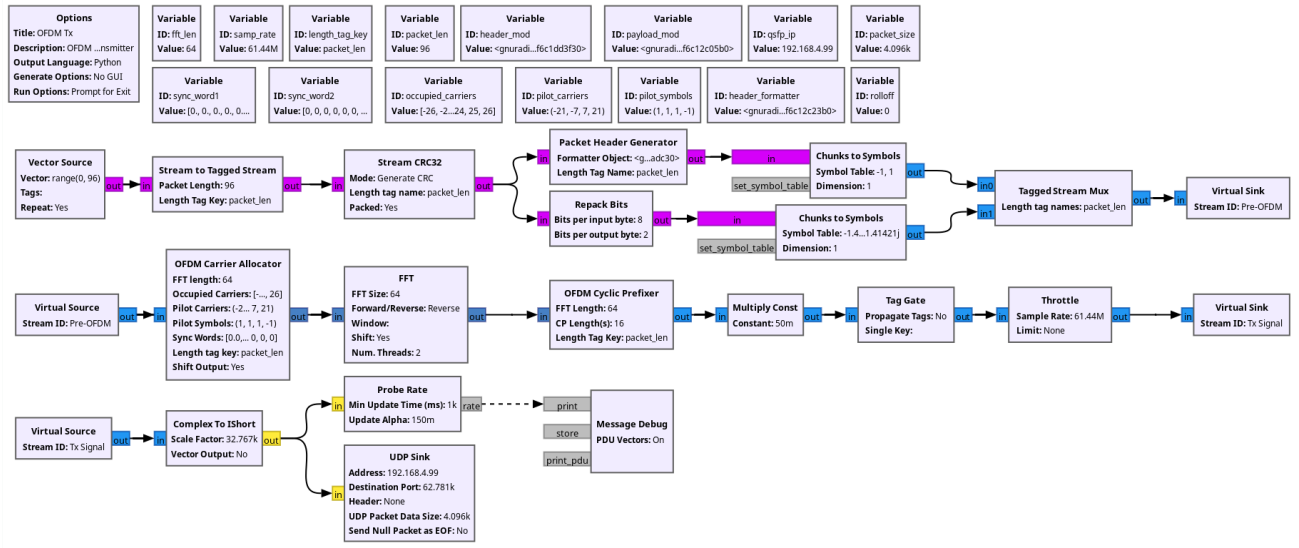

Figure 3. OFDM Transmitter GNU Radio flowgraph

the transmitted signal at a reasonable rate of 122.88 MSPS. The sampling rate could be further reduced using additional fabric decimation stages if desired.

AXI4-Stream outputs of both ADCs go through individual stages of I/Q interleaving and TDATA width repacking to achieve a 32-byte wide, stream before being passedto a AXI4-Stream switch IP which facilitates the runtime selection of active ADC output. The selected stream is then passed to the Network Layer kernel IP where it is encapsulated into UDP packets of user-selectable length and sent over the QSFP network interface.

4. Software Architecture

A modified OFDM transmitter (Figure 3) based on the example design from the *gr-digital* package is used to generate wideband OFDM signals throttled to 61.44 MSPS, which are converted to interleaved 16-bit IQ samples and transmitted via the QSFP network using the UDP network sink. The flowgraph is configured with the Graphical User Interface (GUI) disabled to achieve better performance and stability.

The primary components of the receiver flowgraph (Figure 4) are the UDP source block, configured to accept jumbo packets, data reinterpretation, and *gr-fosphor* [21] GPU accelerated, real-time spectrum visualisation interface blocks. Additional QT widget and XMLRPC Client blocks are present in the design, to facilitate runtime selectable RF-DC center frequency and DDC decimation rate. The QT GUI Waterfall sink provides an alternative waterfall view with slower update speed.

The software architecture for the RFSoC 4x2 board part of

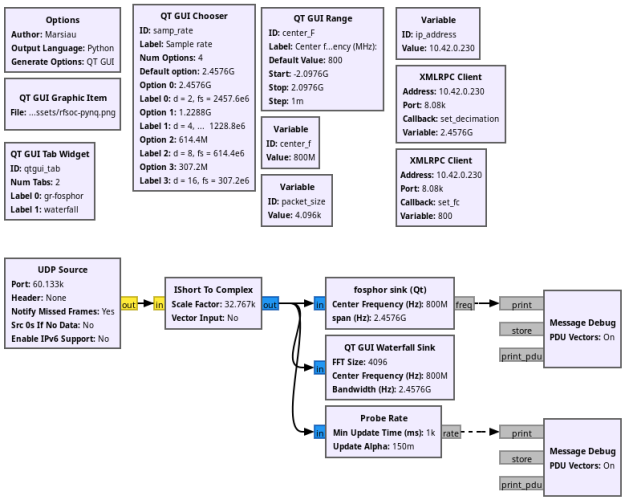

Figure 4. gr-fosphor based receiver flowgraph

the design is written the in Python programming language using the web-based JupyterLab environment. The PYNQ framework is used to load the designed bitstream, configure the QSFP network interface and provide initial configuration for the RF-DCs. Additionally, the Jupyter notebook starts a XML-RPC server in a separate thread to facilitate the remote RF-DC control.

As discussed in Section 3, the port the of GNU Radio UDP sink remains unknown until runtime. However it is required by the *netlayer* IP to validate the received UDP packets. Once the QSFP network is configured and established between the PC and the RFSoC 4x2 development board, the PYNQ MMIO library can be used to query the *netlayer_tuser* IP for the sender's port number and update the socket table accordingly.

5. Experimental Results

This section discusses the resource utilization of the implemented hardware design as well as tests carried out to confirm the functionality of the GNU Radio flowgraph.

5.1. Hardware Resource Utilization

Table 1 provides a summary of the PL hardware resource utilization required to implement the discussed design for the RFSoC 4x2 development board. The entire design consumes approximately 11% of the available Look-Up Tables (LUTs), 12% of the available Flip-Flops (FFs), and 6% of the available Block RAMs (BRAMs). This leaves a significant portion of the device unoccupied, allowing for the implementation of additional functionality such as hardware interpolation/decimation stages. The design's relatively modest hardware resource consumption can be attributed to the utilization of hardened RFSoC resources, including digital up and down converters, as well as the gigabit transceivers.

Table 1. Hardware Resource Utilisation. (target part: XCZU48DR)

Resource	Used	Utilisation
LUT	45276	10.65%
LUTRAM	3906	1.83%
Flip-Flops	100827	11.85%
Block RAM	61	5.65%
GTY Transceivers	4	25%

5.2. Network Test Results

To drive the RF-DAC at 4915.2 MSPS a 61.44 MSPS input signal has to be supplied to the RFSoCs DDCs set up with factor 80 interpolation. The network throughput (R) required to send the input signal can be calculated using the following equation:

$$R = B \times C \times Q \tag{1}$$

where bandwidth *(B)* is the RF-DAC sampling rate, channels *(C)* = 2 for complex (I and Q) data, and the RF-DAC resolution *(Q)* is 14 bits with 2-bit padding, giving 16 bits total for the RFSoC Gen 3 ZU48DR device present on the RFSoC 4x2 board.

The design was tested with Throttle block (configured to 61.44 MSPS) placed before the UDP Sink and Tx Signal conversion from float to interleaved short IQ samples, as well as with throttle block removed and flowgraph unconstrained. Tests were repeated with realtime scheduling enabled, but showed no significant performance improvement over the default scheduling policy. The sampling rate results were acquired using the *Probe Rate* and *Message Debug* blocks, whilst the network throughput test results were acquired from the client system using the *nload* [24] network monitoring application for Linux systems. The results are summarized in Table 2.

Table 2. Network Throughput Test Results.

Test	Sampling Rate (MSPS)	Measured Avg. Throughput (GBit/s)
Expected	*61.44*	*1.94*
Throttled	61.44	1.85
Unthrottled	64.65	1.92

Figure 5 shows the spectrum of the transmitted wideband OFDM signal that was captured by RFSoC RF-ADC channel B and displayed using the GNU Radio gr-fosphor block.

The test results show that the system is capable enough to generate a stable 61.44 MSPS OFDM signal and send it over the OFDM network interface in real time. The reported network use is slightly less than expected but this is consistent with previous results and still under investigation.

6. Further Investigations

Within the SDR community, there has been, a few attempts to offload parts of the processing flow to FPGA on ZYNQ-based devices, namely GNU Radio ZYNQ support that was a result of the 2013 Google Summer of Code project [25] and RFNoC [26] project developed by "Ettus Research" which is mostly aimed at USRP SDRs. Both of these approaches offer limited design flexibility and support only a limited amount of development boards and devices. Since the release 3.8 of GNU Radio, when Python 3 support was officially added, it has been possible to utilize the PL abstraction layers provided by the PYNQ framework. This could provide a platform agnostic and easily extensible way to achieve FPGA acceleration of GNU Radio flowgraphs.

The remainder of this section considers the author's progress with running GNU Radio directly on RFSoC based platforms, and developing GR blocks based on the PYNQ framework for FPGA bitstream management, data transactions between PS and PL, as well as RFSoC-specific functionality.

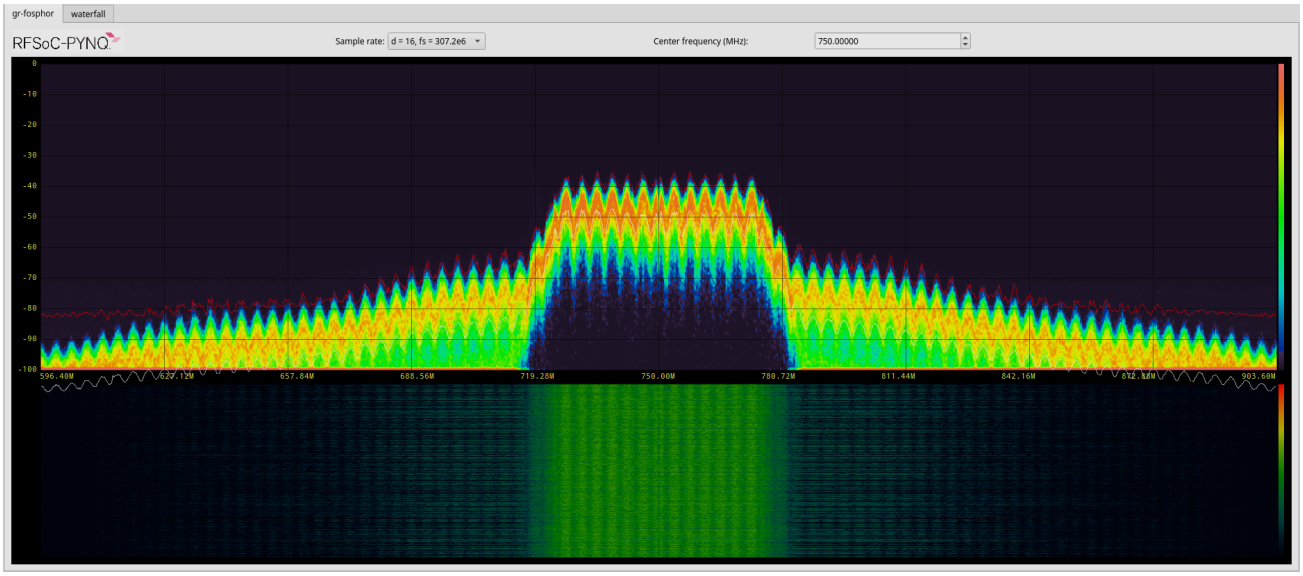

Figure 5. OFDM signal Spectrum

6.1. Board Setup

The installation of GNU Radio on the target RFSoC 4x2 development board has been achieved using a combination of the *radioconda* [27] package repository and *miniforge* [28] installer with the *mamba* [29] package manager in its base environment. This app roach allows GNU Radio to be installed on linux-aarch64 platforms (such as the RFSoC 4x2 PS) without the need to build it from source. The *radioconda* default environment can be further extended using *pip* [30] to install the necessary PYNQ libraries.

Although GNU Radio Companion can be run directly on the board and accessed by connecting a monitor, mouse and keyboard, a far more convenient option is to use a Virtual Network Computing (VNC) or Remote Desktop Protocol (RDP) connection to access the board's display remotely. Alternatively, GR flowgraphs can be developed on a PC in a clone of the board's environment, then transferred onto the board for use.

6.2. gr-pynq

An out-of-tree module of PYNQ based GNU Radio blocks called gr-pynq has been in development to cover some of the most commonly used functionality. The blocks are written in Python and use PYNQ libraries to control the hardware. Example gr-pynq blocks are shown in Figure 6 and the development status of the blocks is summarized in Table 3.

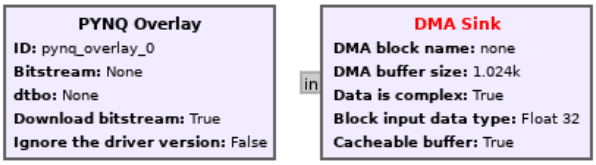

Figure 6. gr-pynq blocks

Table 3. PYNQ-based GNU Radio block development status

Block name	Function	Status	Testing
Overlay	Loads and manages FPGA bitstreams	Done	Done
DMA Sink	PS to PL DMA transfer	Done	Done
DMA Source	PL to PS DMA transfer	Done	WIP
xrfclk	Controls external RF reference clocks	WIP	WIP
xrfdc tile	Controls the RF Data Converters	WIP	WIP

7. Conclusions

In this paper, we highlight RFSoCs unique combination of RF frontend, FPGA fabric, and Linux-capable Arm-based processing system within a single chip, and present a novel design that leverages the QSFP network interface to establish a bi-directional data link between GNU Radio, running on a PC, and the RFSoC 4x2 for wideband OFDM signal transmission and reception. Although OFDM is used as an example, the design supports any arbitrary waveforms generated by GNU Radio. Through our work, we have demonstrated the potential of leveraging PYNQ in tandem with GNU Radio for developing RFSoC applications, and by providing a reference design, we hope to encourage broader adoption of this powerful SDR platform in the GNU Radio community.

8. Acknowledgements

The authors warmly thank Patrick Lysaght, Graham Schelle and Šarūnas Kaladė (all of AMD) for their support of this work. They would also like to acknowledge CENSIS co-funding for Marius Šiaučiulis.

References

[1] C. Liu, M. E. Jones, and A. C. Taylor, "Characterizing the performance of high-speed data converters for RFSoC-based radio astronomy receivers", *Monthly Notices of the Royal Astronomical Society*, vol. 501, no. 4, pp. 5096–5104, Jan. 2021, doi: 10.1093/mnras/staa3895.

[2] X. Pei, J. Li, X. Duan, and H. Zhang, "QTT Ultra-wideband Signal Acquisition and Baseband Data Recording System Design Based on the RFSoC Platform", PASP, vol. 135, no. 1049, p. 075003, Jul. 2023, doi: 10.1088/1538-3873/ace12d.

[3] StrathSDR, "Spectrum Analyser on PYNQ", https://github.com/strath-sdr/rfsoc_sam

[4] J. Goldsmith, C. Ramsay, D. Northcote, K. W. Barlee, L. H. Crockett, and R. W. Stewart, "Control and Visualisation of a Software Defined Radio System on the Xilinx RFSoC Platform Using the PYNQ Framework", IEEE Access, vol. 8, pp. 129012–129031, 2020, doi: 10.1109/ACCESS.2020.3008954.

[5] L. Stefanazzi et al., "The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors", *Review of Scientific Instruments*, vol. 93, no. 4, p. 044709, Apr. 2022, doi: 10.1063/5.0076249.

[6] R. Gebauer, N. Karcher, M. Güler, and O. Sander, "QiCells: A Modular RFSoC-based Approach to Interface Superconducting Quantum Bits", ACM Trans. Reconfigurable Technol. Syst., vol. 16, no. 2, p. 32:1-32:23, May 2023, doi: 10.1145/3571820.

[7] "RFSoC 4x2 Overview", RFSoC-PYNQ. http://www.rfsoc-pynq.io/rfsoc_4x2_overview.html (accessed Sep. 02, 2023).

[8] "RFSoC 4x2 — Real Digital". https://www.realdigital.org/hardware/rfsoc-4x2 (accessed Sep. 02, 2023).

[9] "RFSoC-PYNQ overlays", RFSoC-PYNQ. http://www.rfsoc-pynq.io/overlays.html (accessed Sep. 02, 2023)

[10] Louise H Crockett, David Northcote, and Robert W Stewart, Software Defined Radio with Zynq® UltraScale+™ RFSoC. Strathclyde Academic Media, 2023. Available: https://www.rfsocbook.com/

[11] "RFSoC Tutorials", RFSoC-PYNQ. http://www.rfsoc-pynq.io/tutorial.html (accessed Sep. 02, 2023).

[12] "Zynq SDR Support from Communications Toolbox". https://uk.mathworks.com/hardware-support/zynq-sdr.html (accessed Sep. 02, 2023).

[13] "CASPER - Collaboration for Astronomy Signal Processing and Electronics Research", https://casper.berkeley.edu/ (accessed Sep. 02, 2023).

[14] "PYNQ - Python productivity for Zynq", http://www.pynq.io/ (accessed Sep. 02, 2023).

[15] "AMD - Advanced Micro Devices, Inc", https://www.amd.com/en (accessed Sep. 02, 2023).

[16] "Welcome to Real Digital". https://www.realdigital.org/ (accessed Sep. 02, 2023).

[17] AMD Inc, "Understanding Key Parameters for RF-Sampling Data Converters WP509". Feb. 20, 2019. Accessed: Feb. 16, 2023. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/white_papers/wp509-rfsampling-data-converters.pdf

[18] "JupyterLab: A Next-Generation Notebook Interface". https://jupyter.org/

[19] "PYNQ Development Boards". http://www.pynq.io/board.html

[20] "PYNQ SD Card image". https://pynq.readthedocs.io/en/latest/pynq_sd_card.html

[21] "gr-fosphor - GNU Radio block for RTSA-like spectrum visualization using OpenCL and OpenGL acceleration.", gr-fosphor - GNU Radio block for RTSA-like spectrum visualization using OpenCL and OpenGL acceleration. https://osmocom.org/projects/sdr/wiki/Fosphor. (accessed Feb. 16, 2023).

[22] M. Šiaučiulis, D. Northcote, J. Goldsmith, L. H. Crockett, and Š. Kaladè, "100GBit/s RF sample offload for RFSoC using GNU Radio and PYNQ", in 2023 21st IEEE Interregional NEWCAS Conference (NEWCAS), Edinburgh, United Kingdom: IEEE, Jun. 2023, pp. 1–5. doi: 10.1109/NEWCAS57931.2023.10198070.

[23] AMD Inc, "XUP Vitis Network Example (VNx)". https://github.com/Xilinx/xup_vitis_network_example(accessedFeb.16,2023).

[24] R. Roland, "nload - Real time network traffic monitor for the text console". https://github.com/rolandriegel/nload (accessed Feb. 16, 2023).

[25] "Zynq - GNU Radio", Wiki.gnuradio.org, 2019. [Online]. Available: https://wiki.gnuradio.org/index.php/Zynq

[26] "RFNoC (UHD 3.0) - Ettus Knowledge Base". https://kb.ettus.com/RFNoC_(UHD_3.0) (accessed Sep. 14, 2023).

[27] "Cross-platform installers for a collection of open source software radio packages". https://github.com/ryanvolz/radioconda

[28] "Miniforge installer with Mamba in the base environment". https://github.com/conda-forge/miniforge

[29] "The Fast Cross-Platform Package Manager". https://github.com/mamba-org/mamba

[30] "Package installer for Python". https://pypi.org/

[31] AMD Inc., "UltraScale+ Devices Integrated 100G Ethernet Subsystem v3.1 LogiCORE IP Product Guide ". Accessed: Feb. 16, 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/pg203-cmac-usplus.