100GBit/s RF sample offload for RFSoC using GNU Radio and PYNQ

Marius Šiaučiulis*, David Northcote, Josh Goldsmith, Louise H. Crockett and Šarūnas Kaladė†
Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK
†Advanced Micro Devices, Inc.
Email: *marius.siauciulis@strath.ac.uk

***Abstract*—Modern software defined radio systems are capable of multi-gigabit-per-second sampling rates producing unprecedented amounts of digitized RF data. In applications such as wideband spectrum sensing and machine learning algorithms for cognitive radio, prototyping, and instrumentation, it is often impractical to process the acquired data locally in real-time. This motivates the need for a high speed connection to offload data to an accelerator application running on a secondary processing resource. In this paper, we present a novel hardware and software co-design using the AMD RFSoC 4x2 platform, PYNQ and GNU Radio projects. The demonstrated system is capable of continuous 80GBit/s offload in a 100GBit/s channel, utilising a GPU acceleration to rapidly process the Fast Fourier Transforms of a full 2GHz bandwidth RF signal at 60 frames per second.**

***Index Terms*—high speed offload, GNU Radio, PYNQ, Software Defined Radio, Zynq UltraScale+ RFSoC, hardware software co-design**

I. Introduction

Software Defined Radio (SDR) is a key technology driving innovation in various aspects of the wireless communications field. Modern SDR platforms with high speed data converters (ADCs and DACs) are also useful in other fields like quantum instrumentation [1], radio astronomy [2], etc. The data converters present on AMD's Zynq Radio Frequency System on Chip (RFSoC) [3] devices are capable of multi-gigabit-per-second sampling rates, enabling direct-RF sampling applications and vast amounts of instantaneous wireless spectrum data. It is very difficult for on-chip processors, even with hardware accelerators, to process such data rates. For applications such as machine learning algorithms for cognitive radio, and spectrum monitoring, it is often necessary to transfer the gathered data to a powerful server for processing and analysis.

Offloading is not a trivial problem, as it requires coordinating a hardware-software solution on both the edge device that is gathering the data, perhaps with some pre-processing on-chip, and the connected server machine. A reliable high speed network connection with minimal latency is difficult to achieve, due to implementation complexity and the absence of well-tested reference designs. Therefore there is a need for a multi-gigabit network implementation for SDR use cases. One such example is 100GBE-PYNQ [4] project which provides example implementations of high speed network interfaces for first generation AMD RFSoC based ZCU111 board. The designs use passive loopback cables and do not include network layer functionality but serve as a great starting point for working with high speed networks on AMD SoCs. Another example is a CASPER group tutorial for their 100Gbit/s block [5], which implements reusable modules for high speed offload including User Datagram Protocol (UDP) network layer, data gearbox and packetisation. Although open-source, the CASPER toolflow relies on proprietary MathWorks Simulink [6] and AMD Model Composer [7] software, which might not be readily accessible and may be difficult to incorporate into existing design flows.

In this paper, we present an RFSoC offload design that features high-speed data transfer between the RFSoC 4x2 board and a Commercial Off-The-Shelf (COTS) computer via the Quad Small Form-factor Pluggable (QSFP28) port. QSFP28 allows speeds of up to 100GBit/s which enables data to be transferred and processed off-device directly from the RF-ADCs at RF data rates with minimum latency. The custom hardware design blocks are written in VHDL and implemented with AMD Vivado [8] requiring no additional tools. A *GNU Radio*-based software client with Graphics Processing Unit (GPU) accelerated Fast Fourier Transform (FFT) was developed to visualise the spectrum of the captured RF signal, as well as remotely control the RFSoC's centre frequency and bandwidth. The authors hope to be able to make this design available on the StrathSDR GitHub page [9] in the future.

This section has established the background and motivation for the high speed RFSoC offload design. In Section II, the RFSoC 4x2 board is introduced and a high level overview of the design architecture is presented. Section III describes the hardware architecture and its important modules. Following, Section IV considers the software architecture of the design and GNU Radio-based software client implementation. Results and analysis are considered in Section V. Finally, in Section VI, the conclusions are drawn.

II. System Description

The developed system was designed for the *RFSoC 4x2* board [10] which is based on the 3^{rd} generation AMD RFSoC architecture. In addition to FPGA fabric resources and an Arm-based applications Processing System (PS), it incorporates 4x 14-bit RF Analog to Digital Converters (RF-ADCs) and 2x 14-bit RF Digital to Analog Converters (RF-DACs) with maximum sampling rates of 5 Giga-Samples per Second (GSPS) and 9.85 GSPS, respectively [11]. Additionally, a QSFP28 Ethernet interface is available on the board that supports 4x25GBit/s, 2x50GBit/s or 1x100GBit/s network configurations. The development board was designed to make full use of the PYNQ framework [12], which provides Python Application Programming Interfaces (APIs) for interacting with the Programmable Logic (PL), as well as RFSoC-specific functionality such as controlling the sampling and centre frequencies of the RF-ADCs and RF-DACs, and the frequencies of the internal reference clocks.

The RFSoC offload design was implemented to be compliant with the 1x100GBit/s QSFP28 Ethernet configuration and utilise the UltraScale+ Integrated 100G Ethernet Subsystem found on RFSoC devices. UDP is used for data transfer via the QSFP28 network, which provides no handshaking or guarantee of delivery but reduces network overhead compared to Transmission Control Protocol (TCP). Another advantage of UDP over TCP for time-sensitive applications such as this design is that retransmission is not employed, thus packets with errors are discarded, avoiding the latency required for selective retransmission and the associated signalling. An additional RJ45 Ethernet network interface is used to access a JupyterLab server running on the board (part of PYNQ) and send configuration commands from the GNU Radio-based client interface. The hardware setup is shown in Figure 1.

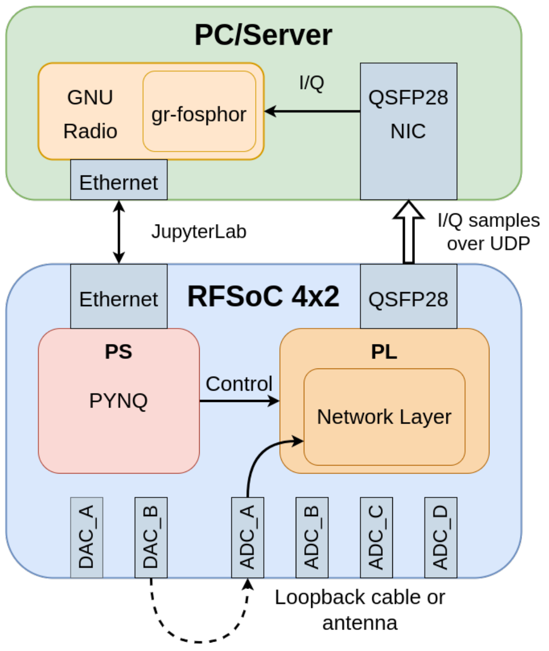

Fig. 1. Hardware setup diagram.

III. Hardware Architecture

To provide the maximum network bandwidth and minimum latency, the high speed network architecture is implemented entirely on the PL of the RFSoC 4x2 board, with software being used only for configuration and performance monitoring. The high-level hardware architecture of the RFSoC QSFP28 offload system is presented in Figure 2.

The network interface layer can be fed from one of the two available sources facilitated by an AXI-Stream switch:

- User Data Direct Memory Access (DMA) — a PS-connected DMA engine that allows custom payloads to be sent from the software environment. It can be used for system and network debugging or to send user-generated data to the client.
- RF-ADC source — ZYNQ Ultrascale+ RF Data Converter (RFDC) Intellectual Property (IP) core which provides a configurable wrapper around the hardened RF-ADC and RF-DAC blocks.

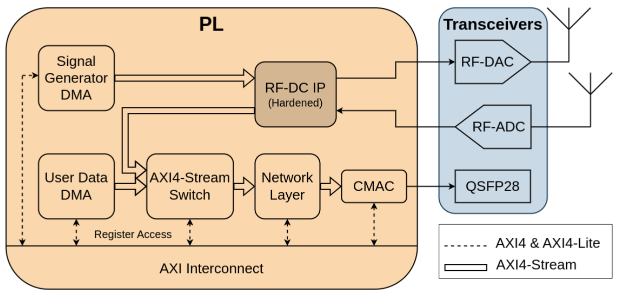

Fig. 2. High level overview of the hardware system.

The RF-ADC source option allows this design to capture RF data from RF-ADC Channel A at a maximum of 2.46 GSPS, to be sent via the QSFP28 network interface. The captured data is output by the RFDC IP in 8 parallel samples per AXI4-Stream cycle configuration for each 16-bit I and Q sample stream. The I/Q output streams are then interleaved and combined in multiple stages to a single 32-byte AXI4-Stream interface. Before being fed to the Network Layer IP, the data is packetised into 64-byte wide packets of user-selectable length.

RF-ADC Channel A can be connected to an antenna via an optional external Variable Gain Amplifier (VGA) to examine the spectrum of captured RF signals (as shown in Figure 3), or be connected to RF-DAC Channel B via a loopback cable to inspect user-generated RF signals.

The hardware signal generator module present in this design allows signals generated in Python to be interpolated and mixed to RF frequencies using the integrated RFDC Digital Up Converter (DUC), to be transmitted via the RF-DAC. The signal generator module is implemented using a DMA module configured in a cyclic mode which allows a segment of a signal to be transmitted continuously without user intervention or additional resources.

When driven by the RF-ADC source, the RFSoC offload design also supports a runtime configurable bandwidth selection by leveraging the RF-ADC programmable decimator. It is a software-controllable chain of hardened decimating filters that can be cascaded together to achieve a desired decimation rate. The following interleaving and packetisation stages are driven by the RFDC module's fabric clock output, which can be synchronised to the programmable decimator output rate. This design feature removes the need for partial bitstream loading and allows continuous operation during the bandwidth change.

Not all of the Gen 3 RFSoC decimation rates are supported by this architecture because the RFDC module is only able to provide output clocks that are equal to the maximum reference clock divided by powers of two. Additional decimation rates could be implemented using PL fabric at the cost of additional hardware resources and design complexity. The available bandwidth selection options are listed in Section V, Table II.

The proposed design makes use of an open-source Network Layer kernel [13], which is a collection of High-Level Synthesis (HLS) modules that implement the network layer functionality including translation between Internet Protocol (IP) and Media Access Control (MAC) addresses, ping capability and UDP transport layer functionality. Additionally, the CMAC IP [14] which requires a no-charge licence, encapsulates the UltraScale+ Integrated 100G Ethernet Subsystem, and is used to provide Ethernet Media Access Controller (MAC), Physical Coding Sublayer (PCS) and Reed-Solomon Forward Error Correction (RS-FEC) functionality.

IV. Software Architecture

This section describes the software architecture of the RFSoC 4x2 offload design, which is split between the development board and the client implementations. The RFSoC 4x2 board contains a Quad-core ARM Cortex A53 processing system which is capable of running an Ubuntu-based PYNQ operating system with JupyterLab as the primary development environment. The software for the board portion of the design is written entirely in Python, and makes use of the PYNQ framework libraries for hardware design access. The notebook contains comments and visual aids as well as code to guide users through the setup and example applications. The following tasks are performed by the RFSoC 4x2 board notebook:

- Configure and start the CMAC core.
- Set the IP address for the board's QSFP28 network interface.
- Open the desired communications socket and fill the table of network socket connections with the relevant information.
- Switch between User Data DMA and RF-ADC as the data source.
- If used, initialise the RF-ADC and RF-DAC.
- Set the UDP packet size.
- Initialise the remote access server for GNU Radio.

As the JupyterLab environment provides an interactive Python session, all of the software and available hardware parameters can be dynamically adjusted during runtime. Custom network payloads can be generated and sent via the User Data DMA directly over the network. Custom signals can also be generated in Python and transmitted from the RF-DAC using the integrated hardware signal generator module.

The client interface is implemented using GNU Radio, a Free and Open Source Software (FOSS) toolkit for SDR design development. The GNU Radio flowgraph mainly consists of a UDP receive block, which is configured to accept packets of user-defined length from the QSFP28 network, a data reinterpretation block, and a *gr-fosphor* [15] Out-Of-Tree (OOT) module. An OOT module is a custom GNU Radio component that needs to be installed separately. The gr-fosphor block provides a real-time spectrum visualisation interface that can utilise GPU hardware acceleration to compute the FFTs required for plotting spectrum data. It is necessary to use a hardware-accelerated block for this task as even modern CPUs are unable to process such extensive amounts of data (>70GBit/s) in real time. Additionally, an Operating System (OS) agnostic protocol for remote procedure calls, XML-RPC [16], is used to facilitate remote control over the board.

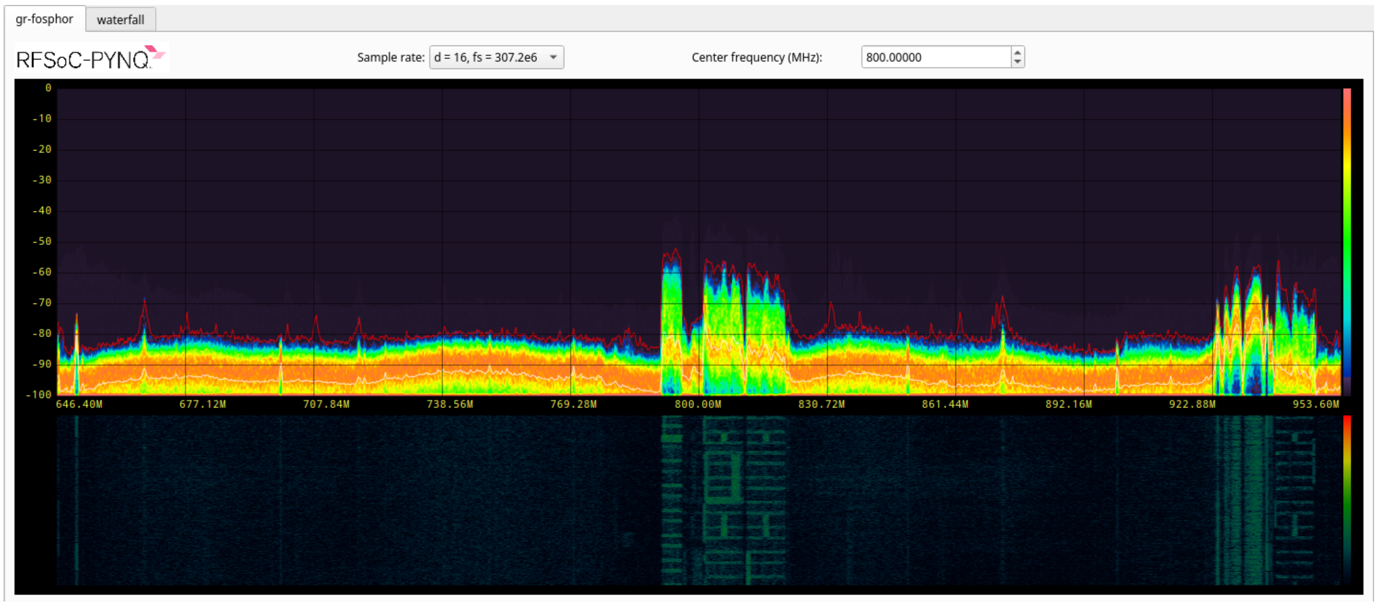
Sample rate: d = 16, fs = 307.2e6
Center frequency (MHz): 800.00000

Fig. 3. Spectrum of the 4G 800MHz band captured by the RFSoC and displayed using GNU Radio.

Using convenient Graphical User Interface (GUI) widgets, users can change the required bandwidth (Table II) and centre frequency of the RF-ADC. An example of the client interface centred on LTE Band 20 (800MHz) is shown in Figure 3.

V. Experimental Results

Using the RFSoC 4x2 development board, the PL hardware resource utilisation required to implement the full RFSoC offload design is summarised in Table I. As the design contains additional features, a subsection of the table is dedicated to the resource utilisation required to implement only the network layer and "ZYNQ Ultrascale+ RF Data Converter" modules. The full design uses approximately 16% of the available Look-Up Tables (LUTs), 15% of the available Flip-Flops (FFs) and 11% of the available Block RAMs (BRAMs), leaving the majority of the device free for implementing other functionality. Additionally only 4 out of the 16 available gigabit transceivers (GTY) are used, meaning multiple instances of the Network Layer module could be implemented on boards supporting additional Small Form-factor Pluggable (SFP) connectors, such as the Zynq UltraScale+ RFSoC ZCU208 board. The relatively low hardware resource usage for the design can be attributed to the use of hardened RFSoC resources including digital up and down converters, as well as gigabit transceivers.

TABLE I
PL Hardware Resource Utilisation.

	Network Layer and RFDC		**Full System**	
Resource	Used	Utilisation	Used	Utilisation
LUT	37822	8.89%	67584	15.89%
LUTRAM	2195	1.03%	5115	2.39%
Flip-Flops	87452	10.28%	120878	14.21%
Block RAM	61	5.65%	110	10.19%
GTY Transceivers	4	25%	4	25%

Referring to the experimental setup shown in Figure 1, the aforementioned RFSoC 4x2 development board was connected to a computer with a PCIe QSFP28 Network Interface Card (NIC) via QSFP28 transceiver modules and a female-to-female fibre-optic cable. To fully leverage the NIC, it has to be inserted in a compatible PCIe slot with x16 lanes available. Another slot with at least x4 PCIe lanes available is required for the GNU Radio client implementation, as it uses GPU acceleration to rapidly process the FFT data. To achieve the maximum network throughput, Jumbo Frames need to be enabled in the client network controller for the QSFP28 interface. Jumbo Frames are Ethernet frames that can transfer up to 9000 bytes of payload compared to the 1500 bytes of standard Ethernet frames. This significantly reduces the amount of Ethernet frame header data sent over the network, cutting the number of CPU cycles required to process the frames and allowing more user data to be sent.

The network throughput test results together with calculated RF-ADC data output rates are summarised in Table II. The RF-ADC data output rate (R_{RF-ADC}) is the maximum amount of data in bits/s that RF-ADC produces for a given bandwidth selection, and thus the maximum amount of data that is expected to be sent via the network. It can be obtained using

$$R_{RF-ADC} = B \times C \times Q \tag{1}$$

where bandwidth *(B)* is the selected RF-ADC sampling rate, channels *(C)* = 2 for complex (I and Q) data, and the RF-ADC 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 network throughput test results were acquired from the client system using the *nload* [17] network monitoring application for Linux systems.

TABLE II
Network Throughput Test Results.

Decimation	Bandwidth (MHz)	RF-ADC Data Output Rate (GBit/s)	Measured Avg. Throughput (GBit/s)
2	2457.6	78.6432	73.40
4	1228.8	39.3216	36.81
8	614.4	19.6608	18.41
16	307.2	9.8304	9.20

As is evident from the third and fourth columns of Table II, the experimental results are very close to the calculated ideal throughput, and the network is able to achieve a constant speed of $>$70 GBit/s. Although the Linux kernel reports that no packets are received with errors, around 6% of the incoming packets are dropped when 2457.6MHz bandwidth is selected, due to the receiving system being unable to keep up with the traffic. The relatively small error between the experimental results and calculated throughput could be related to packet forming latency, UDP network overhead, or PCIe lane throughput limitations of the client PC as these are shared with other devices within the computer.

VI. Conclusion

In this paper, we have presented, validated via demonstration, and evaluated the performance of a novel practical design for direct-RF capture and offload to a GPU accelerator. The software solution opens a UDP socket allowing for arbitrary data transfer, meaning the solution can be applied to not only RF data and communications, but to various instrumentation applications as well. Moving forward, the amount of data enabled by SDR technology is likely to increase – being able to offload this data from power-limited edge devices to datacentres is only going to become more and more important, especially for spectrum monitoring and machine learning applications. Capable software solutions will be necessary to progress in this space.

Acknowledgment

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

References

[1] 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.

[2] 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.

[3] AMD Inc., "An Adaptable Direct RF-Sampling Solution WP489". Feb. 20, 2019. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp489-rfsampling-solutions.

[4] Jenny Smith, "100GBE-PYNQ". https://github.com/JennySmith888/100GBE-PYNQ (accessed April. 28, 2023)

[5] "Tutorial 4: 100GbE — CASPER Tutorials 0.1 documentation". https://casper-toolflow.readthedocs.io/projects/tutorials/en/latest/tutorials/rfsoc/tut_100g.html#introduction (accessed Feb. 16, 2023).

[6] MathWorks Inc., "Simulink - Simulation and Model-Based Design". https://uk.mathworks.com/products/simulink.html (accessed Feb. 16, 2023).

[7] AMD Inc., "Vitis Model Composer". https://www.xilinx.com/products/design-tools/vitis/vitis-model-composer.html (accessed Feb. 16, 2023).

[8] AMD Inc., "Vivado ML Overview". https://www.xilinx.com/products/design-tools/vivado.html (accessed Feb. 16, 2023).

[9] StrathSDR, "University of Strathclyde - Software Defined Radio Research Laboratory GitHub organization page". https://github.com/strath-sdr (accessed April. 28, 2023)

[10] "RFSoC 4x2 Overview", RFSoC-PYNQ. http://www.rfsoc-pynq.io/rfsoc_4x2_overview.html (accessed Feb. 16, 2023).

[11] 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

[12] "PYNQ - Python productivity for Zynq", PYNQ - Python productivity for Zynq. http://www.pynq.io/ (accessed Feb. 16, 2023).

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

[14] 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.

[15] "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).

[16] "What is XML-RPC?" http://xmlrpc.com/ (accessed Feb. 16, 2023).

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