Architecture and Benchmark of an Experimental CRAN Platform over CPRI

Tayyebeh Asgari Gashteroodkhani, Iresha Amarasekara, Aveek Dutta and Dola Saha
Department of Electrical and Computer Engineering
University at Albany SUNY, Albany, NY 12222 USA
{tasgari, iamarasekara, adutta, dsaha}@albany.edu

***Abstract*—Cloud Radio Access Network (CRAN) is an architecture for wireless communication networks, particularly in the context of WiFi, cellular networks, and beyond. The main objective of CRAN is to centralize and virtualize the baseband processing functions of the network to provide several benefits in terms of scalability, efficiency, and flexibility. At first, this paper suggests a design to enhance the core capabilities of CRAN. As a result, two Radio Frequency Systems on Chip (RFSoCs) devices are used to communicate between the Radio Equipment Controller (REC) and the Radio Equipment (RE) through the Common Public Radio Interface (CPRI). The designed flow and the required FPGA resources are discussed. Secondly, the main purpose of this paper is to perform the Fast Fourier Transform (FFT) on the Orthogonal Frequency Division Multiplexing (OFDM) data when the slave sends back OFDM data to the master.**

***Keywords*— CRAN, CPRI, RFSoC, ZCU111, RFDC.**

I. Introduction

The exponential growth of networked users and devices has led to extreme bandwidth requirements to support novel applications. At the same time, seamless user experience and ultra-low latency have become major factors in wireless networks. CRAN is a promising architecture that combines scalable centralized baseband processing in a cloud environment, commonly referred to as the REC, and spatially distributed remote radio equipment, or RE. This is designed to provide scalability, flexibility, and cost savings by efficient sharing of hardware and software resources. In order to provide bottleneck-free, high-speed connectivity between the REC and the REs, the CPRI protocol is used to send and receive baseband samples over multimode optical fiber.

Current commercial CRAN implementations like Nokia Air Scale, Ericsson Cloud RAN, Huawei Cloud RAN, ZTE Cloud RAN, Samsung Cloud RAN, etc. are proprietary solutions tailored to specific use cases, offering limited features and restricted access to the public. These solutions are not designed for experimental research. Experimental CRAN implementations such as COSMOS, POWDER, AERPAW, and ARA [1], on the other hand, leverage Software Defined Radio (SDR) technologies, such as Universal Software Defined Radio Peripheral (USRP) and Zynq UltraScale. One of the bottlenecks of USRP is the latency in moving the radio samples between the hardware and the compute node used for processing [2]. Furthermore, using coaxial cables limits the distance and bandwidth as well. Therefore, we re-imagine a CRAN testbed to create an experimental platform for prototyping various algorithms that cater to the needs of the networking research community. In the future, the testbed can be extended to support protocols and integrated functions as mandated by the ORAN alliance [3]. By providing general-purpose Layer 1 connectivity between the REC and RE over CPRI, the testbed enables many research directions, such as AI/ML-based wireless communication, intelligent beamforming, scheduling, etc., that would not have been possible otherwise.

To achieve real-time operation for specific wireless standards, like WiFi, FPGA implementation or FPGA-accelerated software becomes necessary. Even for mobile operators transitioning into the era of 6G, achieving ultra-low latency is a priority. Exploring an architecture that combines FPGA performance with software flexibility presents a more appealing option for the next generation of wireless networks. This approach allows us to address latency requirements while retaining software-like flexibility, including virtualization capabilities. Our CRAN architecture leverages the Zynq Ultra Scale (RFSoC) [4] in conjunction with fiber optic connectivity to meet the latency demands and facilitate high-speed, long-distance, and data transmission in the next generation of wireless communications.

The paper is organized as follows: The CPRI protocol is reviewed in Section II. Section III describes the overall architecture of the CPRI *Master* and *Slave* nodes along with their specific hardware and software components. Sections IV and V detail the testbench, simulation, and implementation of the combined CPRI *Master-Slave* configuration. Finally, to show the generality of the testbed, we present a Python productivity for Zynq (PYNQ) application to plot the Fourier spectrum in section VI.

II. Common Public Radio Interface (CPRI)

CPRI enables the communication between REC and RE to carry digitized radio signals over coax or fiber, which are linked to the fronthaul of the network. Each REC can connect to multiple REs for wider coverage. At the REC, baseband signals are generated or received, and subsequently sent from and to the RE. RE translates the baseband signals to a bandpass signal and broadcasts it over the air. The RE is also responsible for any additional signal processing that may be required for specific waveforms, such as analog beamforming, synchronization of multiple front-end radios, etc. The RE

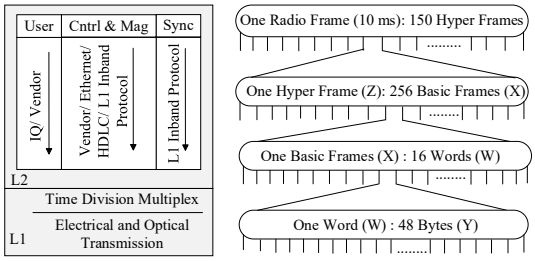


Fig. 1: Overview of CPRI Protocol and Frame Format

can also decode the wireless frames locally before sending the information bits to the REC, thus offering many flexible CRAN architectures as required by the application.

Protocol Overview: The User data, Control and Management data, and Synchronization data are the three types of information flows in CPRI as shown in Figure 1. All CPRI information flows are multiplexed onto Layer 1 and Layer 2, which are designed to transfer data on the digital communication line [5]. I/Q data is transmitted under User data flow in the form of in-phase (I) and quadrature (Q) baseband modulated data. Part of vendor-specific data, also part of User data, is transmitted on specific basic frame numbers within each Hyperframe. Control and management data flow consist of the vendor-specific, High-Level Data Link Control (HDLC) protocol and Ethernet protocols that are frequently exchanged between REC and RE, or two REs for flow management. Synchronization information is used for timing alignment of the CPRI frame in order to detect the beginning of CPRI Hyperframe. Time-Division Multiplexing (TDM) is used to multiplex the different flows on optical or electrical media.

Frame Structure Radio signals are sampled and sent or received as frames between REC and RE. The framing structure is also shown in Figure 1, which is critical to support interoperability between various vendors, I/Q data and other radio parameters. CPRI supports a range of line rates, spanning from 614.4 Mbps to 24,330.2 Mbps. A CPRI radio frame is 10 milliseconds and is composed of 150 Hyperframes. Each Hyperframe has 256 Basic frames spanning 66.67 ms. Each Basic frame consists of 16 Words, which is 260.42 ns long. In general, each word in the CPRI frame is referenced using the format *Z.X.W.Y*, where *Z, X, W,* and *Y* represent the Hyperframe number, Basic frame number, Word number, and Byte number per word respectively [5]. Therefore, to maintain these protocol-specified times, different line rates require different clock frequencies for the CPRI-Master and CPRI-Slave nodes. In this implementation, we have chosen the highest line rate supported by CPRI, 24,330.2 Mbps.

III. CRAN System Architecture

In this work, the REC (configured as *Master* node) and the RE (configured as *Slave* node) are implemented on the Xilinx RFSoC platform. The major building blocks at a system level are shown in Figure 2-Left. The *Master*/*Slave* configuration requires setting up the Xilinx *CPRI IP* in each device separately over the AXI registers or can be hard-coded during core generation. A fiber-optic connection is used as the fronthaul of this architecture. As shown in Figure 2, the CPRI-Master is connected to a client PC, which sends various commands through a TCP socket to control various blocks in the node. The Master-Processing System (Master-PS) filters out these commands and executes them on the Master-Programmable Logic (Master-PL) via the AXI communication interface. Commands sent via the TCP connection are used to configure the *Master*, send messages, or send data when required. The commands execute specific tasks, such as enabling *BRAM_Tx*, which is used to store the I/Q data, send vendor data, etc. After enabling the *BRAM_Tx*, I/Q data is transmitted from the *Master* to the *Slave* via the *CPRI IP*, which is responsible for proper framing and synchronization. The *Slave* is connected to a set of antennas using DAC tiles using the Xilinx RF Data Converter IP for over-the-air transmission. In the receive path, ADC captures analog data from the antenna and converts it into digital samples after direct sampling and digital down-conversion. Subsequently, the CPRI-Slave transmits this data to the *Master*, which can be stored for further processing using Direct Memory Access (DMA), either using the PYNQ or C/C++ in the PS or MATLAB in the host PC.

A. CPRI Master Node

Hardware Components in PL: The PL in the CPRI-Master comprises of Zynq core instance, AXI interface, *vendor module* for decoding vendor messages and interfacing with the *CPRI IP*, *BRAM_Tx*, *MMCM* and other logic to manage different clock domain crossings for data and control signals. These functional blocks are shown in Figure 2, where the PL fabric and the AXI interface are set to operate at 125 MHz. The reference clock frequency, management clock rate, and free-run clock rate are 245.76 MHz, 125 MHz, and 7.2 MHz, respectively. The core utilizes the information about the management clock rate to verify the timing accuracy of the reset sequence for the transceiver and calibrate the channel phase-locked loop. A *MMCM* is added to generate the required clocks for the *CPRI IP*.

The *vendor module* is tasked with sending and receiving vendor-specific data [5], with its core components being the vendor-decode and CPRI-interface. Address decoding and registration supporting the *vendor module* are integral functions within this component. It is a user-defined AXI peripheral that contains state machines to emulate the behavior of AXI-slave module. This allows for vendor-specific data to be transmitted from the host to the PL and then inserted in specific control words within the CPRI frame by the *CPRI IP*. In this implementation, we have initialized the *BRAM_Tx* with a stream of baseband samples for a 20 MHz bandwidth OFDM packet. The TCP client sends the command to enable the *BRAM_Tx* through the socket, and it transmits the OFDM-modulated data to the CPRI continuously. In the receive path, *BRAM_Rx* stores the samples from the slave before forwarding them to the *AXI DMA* as shown in Figure 2-Left.

Software Components in PS: The Top-right inset of Figure 2 depicts the multithreading architecture comprised of three

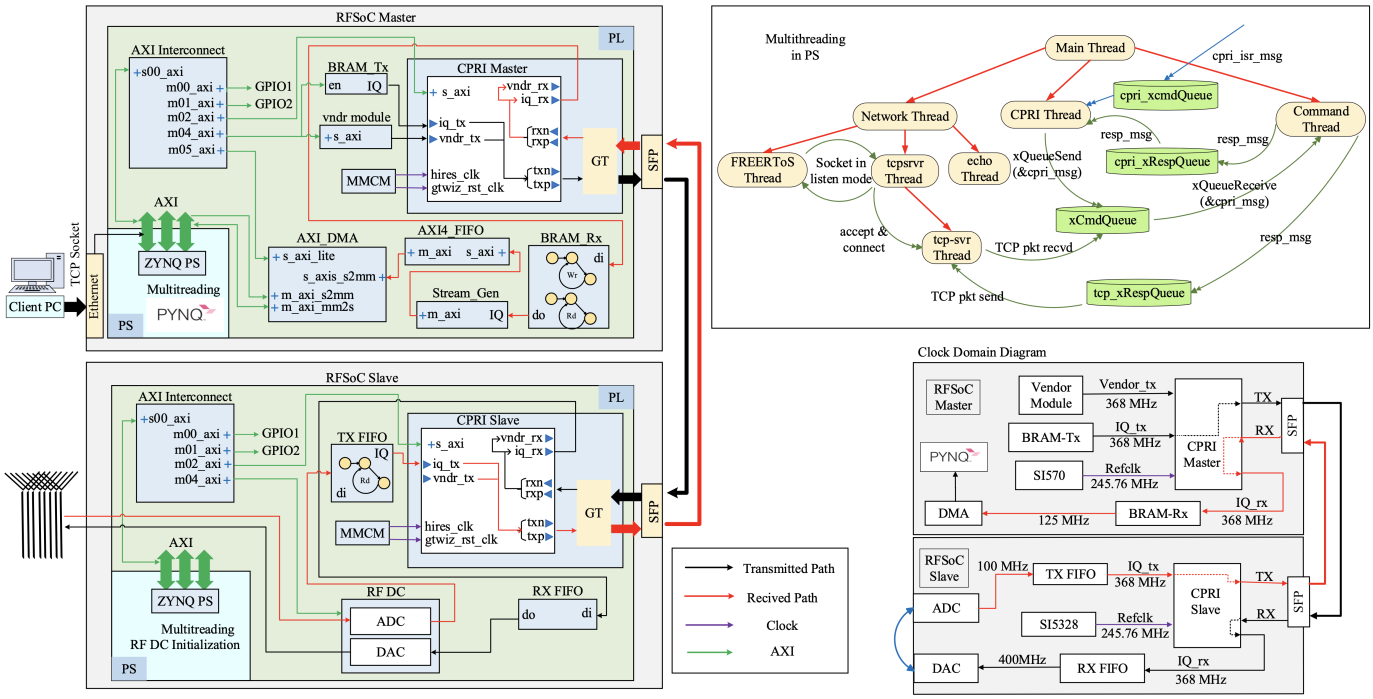


Fig. 2: ***Left:*** *High level block diagram Master-Slave CRAN configuration in PL and PS.* ***Top-right:*** *Multi-threading mechanism in the PS and inter-process communication.* ***Bottom-right:*** *Clock domain and synchronization between various PL entities.*

main threads in the application: a) the network thread, b) the CPRI thread and c) the command thread. A thread is spawned for each TCP connection and TCP packets are sent to the command processor. The network thread is associated with TCP and echo connections, which use the open-source lightweight IP (lwIP) for the TCP/IP stack. The network thread spawns the *echo*, *tcpsrvr*, and *FREERToS* threads. The *FREERToS* thread invokes the function *xemacif_input_thread* from the Xilinx lwIP Ethernet driver. It receives the incoming packet from the Ethernet MAC and passes it to the lwIP stack for processing. The *tcpsrvr* thread is a common TCP server that connects to the client and accepts the packet from the client. This creates another *tcp_srvr* thread, which receives TCP packets from the socket and forwards the payload to the command queue (*xCmdQueue*) and receives TCP pkt send from the response queue (tcp_xRespQueue).

To manage the order of the operation, the design includes several queues, such as *cpri_xCmdQueue, cpri_xRespQueue, xCmdQueue*, and *tcp_xRespQueue*. The command queue (*xCmdQueue*) receives the commands (*tcp_msg* or *cpri_msg*) from the network thread (*tcp_srvr*) and CPRI thread (*CPRI_THRD*). The CPRI thread is related to messages from the vendor interface with the Interrupt Service Routine (ISR). The command thread parses and executes the commands, including reset, vendor data transmission and reception, enabling the BRAM, and more. After the execution, the response is sent to the CPRI response queue (*cpri_xRespQueue*) or TCP response queue (*tcp_xRespQueue*). The CPRI command queue (*cpri_xCmdQueue*) receives the CPRI ISR message by raising an interrupt from the vendor module in PL.

B. CPRI Slave Node

Hardware Components in PL: The CPRI-Slave receives the I/Q and vendor-data from the CPRI-Master over fiber and forwards the I/Q data to the RF Data Converter for onward transmission over the air. The *tx-n/p* and *rx-n/p* are the differential serial outputs of the GTYE4 transceiver [6], respectively. The master's *tx-n/p* connects to the slave's *rx-n/p* via a generic SFP28 transceiver and OM4 fiber. The 64-bit *iq_rx* in the CPRI-Slave is stored in *RX_FIFO*, which is forwarded to the RF data converter. There are two ways to configure CPRI as a slave. The *CPRI IP* core can be configured as a slave core or by setting the *core_is_master* port to zero in the CPRI master IP. The CPRI-Master generates synchronization signals and timing data to facilitate the synchronization and alignment of the CPRI-Slave with the transmitted data. The High-Frequency Normal SYNC (HFNSYNC) is a synchronization signal that is used in the CPRI specification to provide precise synchronization between the master and slave [5].

The Xilinx ZCU111 board consists of four ADC tiles and two DAC tiles. Each ADC tile has two converters, while a DAC tile has four converters [7], [8]. In order to convert digital I/Q data using the RF DC, two converters in a DAC tile (that use the same sampling clock) are enabled. In this design, the sampling rate (DAC_S_{rate}) of DAC is set to 6.4Gbps, and the sampling clock is generated from the on-chip PLL. The reference clock of this PLL is 400 MHz, and it is driven by an onboard oscillator (see Section V-A for details). The mixer frequency is set to 2.484 GHz for DAC. Since the size of the *iq_rx* port of the *CPRI IP* is 64-bit and DAC has a 14-bit resolution with a 16-bit digital signal processing path, the DAC

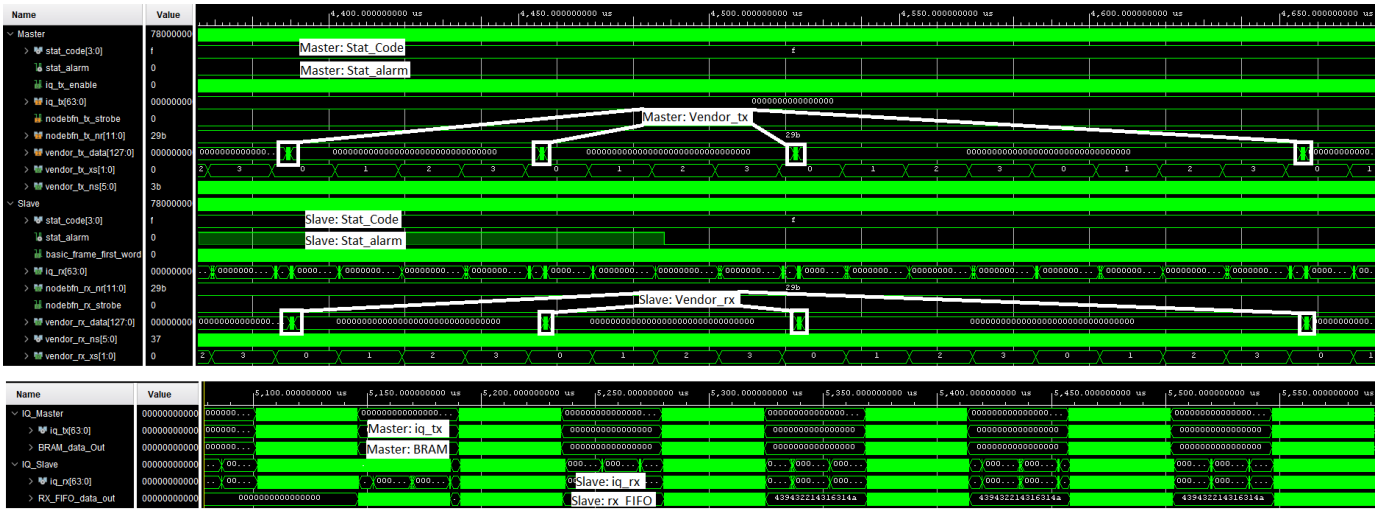


Fig. 3: Master and Slave Combined Simulation: ***Top:*** *Transmit four Vendor messages to the Master and receiving them in the Slave.* ***Bottom:*** *Transmit IQ data from BRAM_Tx in the Master to the Slave and storing in RX_FIFO*

.

is configured to receive 4 Samples per Clock (SpC). These 4 samples contain two I and two Q data when forwarded to the DACs via the AXI4 stream protocol by the *RX_FIFO*. Therefore, the required stream clock is given by:

$$\mathrm{DAC_{clk}} = \frac{\mathrm{DAC\_S_{rate}} \times \mathrm{IQMode}}{\mathrm{I_{rate}} \times \mathrm{SpC}} = \frac{6.4 \times 2}{8 \times 4} = 400\,\mathrm{MHz} \quad (1)$$

where the interpolation rate ($\mathrm{I_{rate}}$) is 8 and IQMode is defined as 2 for IQ digital data mode.

Similarly, two ADCs are enabled to receive I/Q data using RF DC. The ADC is configured to use the on-chip PLL to generate the sampling clock required at a sampling rate of 3.2 Gbps. This sampling clock is used by all the converters in the tile. Similar to the DAC, the on-chip PLL in the ADC uses an external reference clock of 400 MHz. The mixer frequency of the ADC is 2.484 GHz, and therefore, it is set to operate in the second Nyquist zone. The digital data path of the ADC also uses the AXI4 stream protocol, and two ADC converters send 4 samples of I and Q data for each clock cycle in parallel. Therefore, the ADC clock rate is given by:

$$\mathrm{ADC_{clk}} = \frac{\mathrm{ADC\_S_{rate}} \times 2\mathrm{G_{IQMode}}}{\mathrm{D_{rate}} \times \mathrm{SpC}} = \frac{3.2 \times 1}{8 \times 4} = 100\,\mathrm{MHz} \quad (2)$$

where the decimation rate ($\mathrm{D_{rate}}$) is 8 and $2\mathrm{G_{IQMode}}$ is 1.

Software Components in PS: Similar to Master-PS, the Xilinx FreeRTOS application running on Slave-PS is responsible for configuring hardware components in Slave-PL. In addition, Slave-PS is also responsible for initializing RF DC. This involves initializing and verifying the state of each ADC tile and DAC tile separately.

IV. Testbench and Simulation

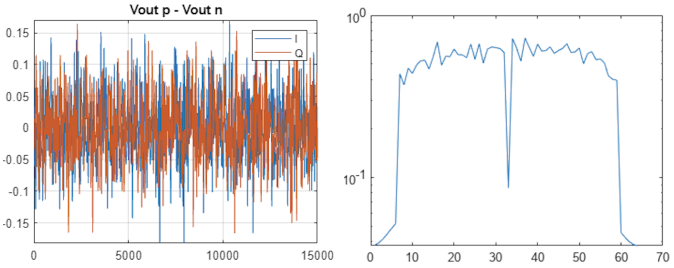
Xilinx Verification Intellectual Property (VIP) [9] is used to simulate the behavior of the system on the Zynq UltraScale+ by emulating the command and control over the AXI interface to initialize the RF DC, enable *BRAM_Tx* to transmit the I/Q data, send vendor data, and interrupt handling.

A. Master and Slave Node

Figure 3 depicts the *master* and *slave* combined simulation. The master *tx-n/p* and *rx-n/p* ports are connected to the slave *rx-n/p* and *tx-n/p* ports in the testbench, respectively. The master's testbench contains GPIO reset, enabling the *BRAM_Tx* and sending vendor-specific data. The slave section includes the GPIO reset. The CPRI status alarm represents a bit of the status and alarms interface that declares Loss of Frame (LOS), Loss of Signal (LOS), reset bit, etc. When I/Q data is transmitted, the CPRI status alarms are low for both master and slave as shown in Figure 3).

In Figure 3-Top four vendor data items are sent via VIP to the *vendor_tx_data* CPRI-Master port. The *Slave* receives vendor data from the master via *vendor_rx_data* CPRI-Slave port. Each Hyperframe consists of 256 control words, which are organized into 64 subchannels across 4 control words. The subchannel index is denoted as N_s and spans from 0 to 63 and the control word index X_s, ranges from 0 to 3. Following the CPRI specification, the transfer of 128 bits of vendor data occurs in particular subchannels 15, 16, 17, and 18 when the control word index X_s equals 0 [5]. After enabling *BRAM_Tx*, I/Q data is transmitted to the CPRI core through *iq_tx* port. Since both cores are in operational mode, the *Slave* receives the I/Q data and sends it to *RX_FIFO* through the *iq_rx* port, as shown in Figure 3-Bottom. The output of *RX_FIFO* is transmitted to the DAC interface of the RFDC. The output of *RX_FIFO* is stored in a file and is used as input for the RF DC to simulate the receive path.

Frame synchronization is a process that includes synchronization on the transmit and receive sides. The Node-B Frame Number (BFN) value in the *nodebfn_tx_nr* signal must be kept for the duration of the Universal Mobile Telecommunications System (UTMS) frame (10 ms) and the user logic should generate the strobe signal, *nodebfn_tx_strobe* every 10 ms, which updates the value of the Node-BFN. Similarly, the



(a) Output of RFDC *(b) Baseband Power Spectrum*

Fig. 4: OFDM Spectrum (without carrier) in MATLAB

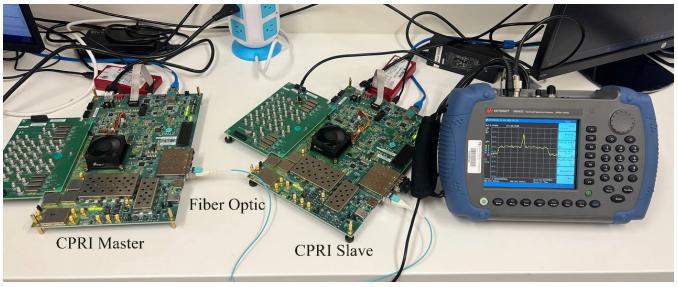


Fig. 5: CPRI Master-Slave Implementation

strobe signal, *nodebfn_rx_strobe* must be maintained for the chip period (T_c) on the receive side and the frame number, *nodebfn_rx_nr* will get the value when the strobe signal is asserted. Figure 3-Top shows the values of *nodebfn_tx_nr* and *nodebfn_tx_strobe* in the *Master* at the beginning of the simulation. However, at the *Slave*, the *nodebfn_rx_nr* is updated once it is received from the CPRI-Master.

Test Data Preparation: For testing, we generated an OFDM data packet with 10 OFDM symbols with 16QAM modulation scheme. As the sampling frequency of the OFDM packet is 20 MHz, and RF DC expects 2Is and 2Qs within one clock cycle of DAC_{clk}, the generated packet was up-sampled by a factor of 40 as per (1). Then, up-sampled I/Q data were arranged into 64-bit samples comprised of alternating sequences of I and Q data. Figure 7 shows the spectrum of the OFDM packet, which is converted to analog by the RF DC and shows the same OFDM packet that is stored in *BRAM_Tx*. Figure 4a shows the differential output of the RF DC and the baseband power spectrum without carrier is shown in Figure 4b.

V. Implementation and Results

Figure 5 shows the bench setup of CPRI-Master and Slave nodes with two RFSoCs, connected using two generic SFP28 transceivers and OM4 fiber. The external clocks are configured for both *Master* and *Slave* as described in Section V-A. Once the *Master* and the *Slave* are in the operational state if the *BRAM_Tx* is enabled at the *Master*, it sends the OFDM packet to the *Slave*. The *Slave* then transmits the received I/Q data via RF DC. The spectrum analyzer in Figure 5 shows the power spectrum of the OFDM packet along with the carrier.

A. External Clock Configuration

Both the *Master* and *Slave* nodes clock signals that are driven by onboard programmable oscillators/PLLs. To program these oscillators/PLLs, the Xilinx ZCU111 system controller user interface (SCUI) [4] is used either by setting the required frequencies on the GUI or using a register map. SCUI uses the onboard system controller (MSP430) and I2C for programming. The required connection between the host PC and the system controller IC is established via a USB-to-UART connection. The *Master* and *Slave* designs utilize the onboard oscillators: *USER_SI570* and *USER_MGT_SI570* (for the GTYE4 transceiver), which provide low jitter, high resolution, and a wide range of clock signals. The *USER_SI570* oscillator generates a 300 MHz clock to drive the reference clock input of MMCM in both master and slave designs.

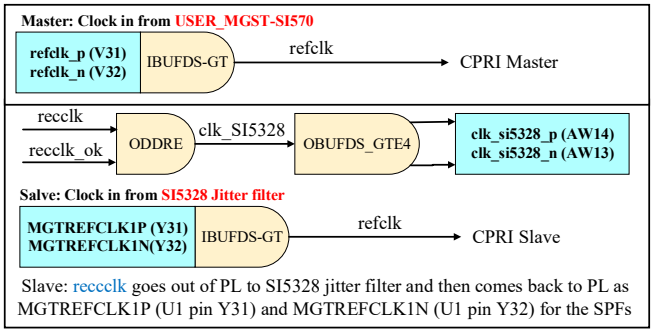


Fig. 6: CPRI Reference Clocks for Master and Slave

High-performance transceivers require a precise and noise-free clock for high-speed serial communication. In particular, for the chosen line rates of CPRI, this clock must be a 245.76 MHz clock. In *Master* and *Slave* designs, this clock is driven by two different types of oscillators. CPRI-Master uses the output of *USER_MGT_SI570* oscillator as its reference clock, while in the *Slave* this reference clock is recovered from its received signal as shown in Figure 6. CPRI IP core outputs a clock based on the received signal, called *recclk*, and the frequency of that can be different based on the line rate of the received signal. The slave design has a dedicated logic to convert this clock to a fixed clock rate of 15.36 MHz for all the line rates. This pre-scaled clock is then routed through an external jitter-removal PLL to generate 245.76 MHz clock, which is used in *Slave* as CPRI reference clock. The SI5328 is the onboard jitter-removal PLL on ZCU111 RFSoC, and it is programmed using the SCUI and the corresponding register map is generated using ClockBuilder Pro Software from Skyworks.

In *Slave*, the required 400 MHz PLL reference clock of any ADC/DAC tile is generated from onboard cascaded oscillators, LMK00304 and LMX2594. The register maps for these are generated using Texas Instruments's TICSPRO-SW support software and are programmed by the SCUI tool.

B. FPGA Utilization

The FPGA utilization and power consumption of *Master* and *Slave* are discussed in Table 1. It shows that the *Master* comprises 22.3% of Block RAM (BRAM) and 5.8% of Configurable Logic Blocks (CLB). In the *Slave*, 36.1% of BRAM and 4.3% of CLBs are utilized. Additionally, the power

	Power (W)	BRAM	CLB (LUTs)	MMCM
Master	5.063	241(22.31%)	24872(5.85%)	1(12.5%)
Slave	7.608	390(36.11%)	18359(4.32%)	1(12.5%)
RFDC	2.192	0(0%)	3194(0.75%)	0(0%)

TABLE I: FPGA utilization for master, slave, and RFDC.

consumption for the *Master* and *Slave* is 5 watts and 7.6 watts, respectively.

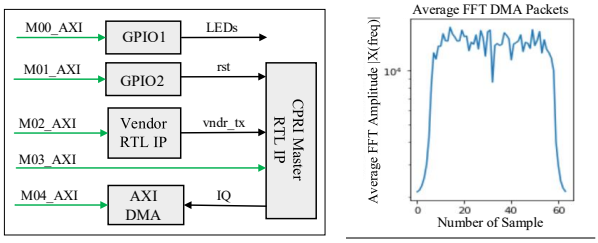
VI. FFT Application over CPRI

Bi-directional communication in wireless networks enables real-time interaction between a master and a slave. In the transmission path, OFDM data is sent by activating *BRAM_Tx* on the *Master* to the CPRI-Master. This information is received by the CPRI-Slave and then stored in the *RX_FIFO* on the *Slave* before being forwarded to the DAC. A cable establishes a connection between the ADC and DAC. In the reception path, the storage of ADC data in the *TX_FIFO* is a result of clock domain crossing. The clock domain diagram depicted in Figure 2 highlights the distinction between the CPRI frequency (368 MHz) and the clock frequencies (100 MHz for ADC and 400 MHz for DAC).

By enabling the transmit bit in the CPRI-Slave, the *Slave* can send data back to the *Master*. The received data in the master is captured by *RX_BRAM*. *AXI DMA* and *AXI FIFO* are utilized to send the DMA packet out of the PL shown in Figure 2. PYNQ, an open-source project from Xilinx, seamlessly integrates the simplicity of Python programming with the capabilities of PL and PS elements in Xilinx's Zynq system-on-chip (SoC) devices [10]. In the context of the PYNQ framework, a fundamental restructuring of our system architecture into IPs is necessary. Each IP is allocated a unique address space. The five identified IPs in this context include *AXI DMA*, *GPIO1*, *GPIO2*, *Vendor RTL IP*, and *CPRI Master RTL IP*. *GPIO1* and *GPIO2* are specifically designated for managing LEDs and CPRI reset, respectively. Combining the *vendor module* and *BRAM_Tx* forms a cohesive *Vendor RTL IP*, while the *CPRI IP*, *MMCM*, and *BRAM_Rx* collectively serve as the RTL components within the CPRI Master IP. This systematic approach enhances the organization and efficiency of our system design. Figure 7a illustrates the IPs layout diagram for the PYNQ framework.

The FFT application in PYNQ involves a series of tasks. These tasks include loading the FPGA bitstream file, initiating the reset sequence, activating *BRAM_Tx*, initializing *AXI DMA*, and allocating memory for each DMA packet. Following this, the system processes 20 DMA packets, converts them to signed decimal numbers, and then applies FFT to the decimated data. As a result, the received DMA packets undergo FFT processing to visualize the OFDM data.

Implementation and Results: The transmission of OFDM data to the CPRI-Master is continuous, with a noticeable delay between successive packets of OFDM data. As a result of this clock domain crossing, when the CPRI-Slave receives I/Q data, the corresponding OFDM data is stored in the *RX_FIFO* on the *Slave* in the transmitted path. The *TX_FIFO* is utilized to temporarily store data before its transmission back to the



(a) IP layout in the PYNQ *(b) Average spectrum of 20 DMA*

Fig. 7: PYNQ framework for transferring OFDM waveform

Master in the received path. Figure 7b illustrates the FFT average of 20 DMA packets within the PYNQ environment

VII. Conclusion

This paper presents a system architecture aimed at enhancing the CRAN core. The proposal involves configuring two Zynq UltraScale+ RFSoCs as master and slave components. The OFDM data packet is transmitted to the *Master* by activating *BRAM_Tx* through the TCP socket or PYNQ. Subsequently, the CPRI-Slave receives the I/Q data, directs it to the DAC, and sends back the data from the ADC to the master. Simulation results demonstrate the functionality of both the *Master* and *Slave*. FPGA utilization metrics indicate the number of resources employed, with an explanation attributing the low resource usage to the expansive size of the FPGA. Additionally, the paper explores the FFT of the received data from the slave within the PYNQ framework.

VIII. Acknowledgment

This work is supported by the National Science Foundation (NSF) Award *#1823225-CRI: II-NEW: CHRONOS: A Cloud based Hybrid RF-Optical Network Over Synchronous Links*.

References

[1] Platform for advanced wireless research. [Online]. Available: https://advancedwireless.org/

[2] X. Jiao, I. Moerman, W. Liu, and F. A. P. de Figueiredo, "Radio hardware virtualization for coping with dynamic heterogeneous wireless environments," in *Cognitive Radio Oriented Wireless Networks*, P. Marques, A. Radwan, S. Mumtaz, D. Noguet, J. Rodriguez, and M. Gundlach, Eds. Cham: Springer International Publishing, 2018, pp. 287–297.

[3] Open radio access network. [Online]. Available: https://www.o-ran.org/

[4] "ZCU111 Evaluation Board User Guide PG1271," 2018. [Online]. Available: https://www.xilinx.com/support/documents/boards_and_kits/zcu111/ug1271-zcu111-eval-bd.pdf

[5] "CPRI Specification V7.0." [Online]. Available: http://www.cpri.info/downloads/CPRI_v_7_0_2015-10-09.pdf

[6] "UltraScale Architecture GTY Transceivers ug578," 2021. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/user_guides/ug578-ultrascale-gty-transceivers.pdf#page=241

[7] "Zynq UltraScale + RFSoC RF Data Converter 2.2 PG269," 2019. [Online]. Available: https://docs.xilinx.com/r/en-US/pg269-rf-data-converter/Introduction

[8] "Zynq UltraScale+ RFSoC RF Data Converter Evaluation Tool (ZCU111) PG1287," 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/boards_and_kits/zcu111/2020_2/ug1287-zcu111-rfsoc-eval-tool.pdf

[9] "https://docs.xilinx.com/v/u/en-US/ds941-zynq-ultra-ps-e-vip (DS941)," 2021. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds941-zynq-ultra-ps-e-vip

[10] Pynq introduction — python productivity for zynq (pynq). [Online]. Available: https://pynq.readthedocs.io/en/latest/