Introducing ReCPRI: A Field Re-configurable Protocol for Backhaul Communication in a Radio Access Network

Juan Camilo Vega
Department of Electical and Computer Engineering
University of Toronto
Toronto, Canada
camilo.vega@mail.utoronto.ca

Qianfeng (Clark) Shen
Department of Electical and Computer Engineering
University of Toronto
Toronto, Canada
qianfeng.shen@mail.utoronto.ca

Alberto Leon-Garcia
Department of Electical and Computer Engineering
University of Toronto
Toronto, Canada
alberto.leongarcia@utoronto.ca

Paul Chow
Department of Electical and Computer Engineering
University of Toronto
Toronto, Canada
pc@eecg.toronto.edu

***Abstract*—We present the Re-configurable Ethernet Common Public Radio Interface (ReCPRI) protocol as a replacement of the existing Ethernet Common Public Radio Interface (eCPRI) standard. Using the same communication infrastructure, this protocol is shown to provide the same functionality as eCPRI, maintaining full backward compatibility, all while reducing the required data bit rate and increasing the ADC sample rate by 4.4x. This is achieved using hardware-based data pre-processing at the radio tower. This protocol also makes the backhaul network SDN compatible. ReCPRI is capable of modifying the data bit rate dynamically as usage changes, modifying the hardware infrastructure in real time, migrating towers dynamically between Base Band Units, dynamically adding new functionality, and modifying the baseline infrastructure. This is all accomplished in real time, without interrupting service, and all from the Base Band Unit's software layer.**

***Index Terms*—5G, CPRI, eCPRI, RAN, Backhaul Network, FPGA, IOT, Software Defined Networking**

I. Introduction

Wireless communication is no longer a platform used almost exclusively for phone calls with low data transfer requirements. Users of wireless communication now expect high speed internet access, and the number of users is growing exponentially as the Internet of Things (IoT), the idea of using millions of wireless interconnected sensors to create smart homes, workplaces, or even cities, is becoming mainstream [1] [2]. Machina Research Group estimates that there will be 27 billion wireless machine-to-machine(M2M) communicating devices worldwide by 2024 [3]. This has resulted in the demand for wireless communications to grow faster than the technology, resulting in the need to reconsider the architecture itself.

Research proposals exist for improving the fronthaul network – the wireless network between the devices and the cellular towers. This includes ideas such as Massive MIMO, Millimeter wave technology, device-to-device(D2D) direct communication, and others [1]. However, less research has been done on the backhaul network – the network that exists between the radio towers, and the centralized servers which link this network to the cloud – resulting in protocols that are highly outdated and incapable of supporting these new fronthaul technologies [4] [5].

Ethernet networks have also addressed the challenge of exponentially increasing bandwidths. A recent solution to this problem showing excellent results is Software Defined Networking (SDN) [6] [7] [8]. Traditional Ethernet networks utilize switches with fixed routing tables. Therefore, if a packet travels from node A to node B, the directions it follows are always the same regardless of usage and congestion. With SDN, sensors are added to document the congestion at different switches in the system and software based applications monitor these values in real time, modifying the routing tables, or paths the packets take, to spread the traffic over the network, thereby removing highly congested bottlenecks from the system [6] [7] [8].

We propose the ReCPRI protocol to enable backhaul networks to properly support the new fronthaul technologies. ReCPRI adds reconfigurability and sensors to the backhaul networks, enabling SDN for this network, while maintaining compatibility with eCPRI nodes [1]. It also makes use of a hardware Field Programmable Gate Array[2] (FPGA) device in the radio tower to perform pre-calculations on the incoming data to reduce the amount of traffic that needs to be transmitted from the tower to the central server. Overall, ReCPRI is shown to reduce the amount of data needed to be transferred while increasing the number of devices that can be simultaneously connected to the tower compared to eCPRI. This is done without changing[3] the eCPRI-designed network infrastructure.

Section II provides background on the current infrastructure. Section III showcases the ReCPRI protocol highlighting the changes from eCPRI. Sections IV and V explain the hardware and software used to create a prototype ReCPRI radio tower,

[1]an eCPRI node can still communicate with an ReCPRI node
[2]A re-configurable digital hardware circuit
[3]no new Ethernet wires or switches were added

978-3-903176-15-7 © 2019 IFIP

and the performance results. Section VI then explores related works that have attempted to address this problem. The paper concludes in Section VII.

II. Background

A. Radio Access Networks

The purpose of Radio Access Networks (RANs) is to provide a data link between wireless devices, such as cellular phones, and a common telecom virtual cluster, to form a link with another telecom device (to make phone calls) or with the Internet (cellular data, messaging, etc.) [4] [9]. This link comprises two logical sectors. The fronthaul network is the sector used to bridge the wireless gap between the cellular device and the radio tower. The second sector is the backhaul network, responsible for linking the central cloud cluster with the analog cellular towers [4]. In this study we improved the backhaul network.

The backhaul network, consists of various data processing modules (Fig.1). Starting with the antenna, the raw analog signal is first amplified and filtered. The signal is then turned into a digital signal by using a digital to analog converter (DAC) for outgoing data and an analog to digital converter (ADC) for incoming data. It then goes through digital filtration stages to remove known distortion and is then up or down converted to the voltage range that the processing unit can handle. A Fourier/Inverse Fourier transform is then performed to bridge between the frequency and time domain. A Data Encoder and Decoder then re-converts those amplitudes and phases into the binary signal the cellular unit was attempting to transmit, and then signal goes through a series of Post Processing and Packaging stages before it is inserted into the cloud network [4] [10].

Expensive processing units are required to perform all of the data processing steps in the backhaul network in real time. However, the incoming data rate per tower is relatively low and as a result, if an independent processing unit is added for each tower, it would result in high infrastructure costs and under-utilization of computational resources. Furthermore, radio towers have high environmental exposure complicating local data processing. The solution is to batch process the incoming data from multiple towers in one larger processing unit [4] [10].

CPRI, was created as a method of splicing the backhaul network by communicating data from multiple towers to a centralized processing unit [4] [11]. Due to the low DSP capabilities when CPRI was developed, it was decided that everything before the Fourier transform would be performed at the tower in a unit called the Remote Radio Head (RRH), which consists of analog circuits, an ADC/DAC, a DSP, and occasionally an FPGA. Similarly, the Fourier transform and all data processing afterwards would be performed on a centralized processor called the Base Band Unit (BBU). Fig.1 shows the data path of CPRI and eCPRI. [4] [10] [11].

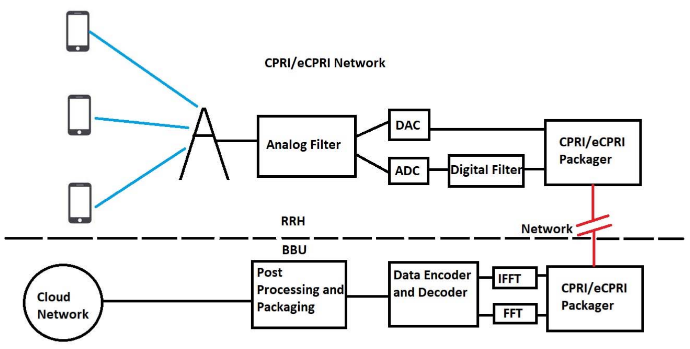

Fig. 1. CPRI/eCPRI data path

B. CPRI

One of the requirements for CPRI, needed for the real-time Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) stages, was for it to be transparent. Data cannot arrive in bursts. Data must enter the FFT at the same rate it entered the CPRI block in the RRH [12]. To solve this issue, CPRI was developed as a data stream serial communication protocol. In serial protocols there are no messages, rather a clock runs alongside the data, which transmits one bit per cycle continuously. Thus to make sense of the data, a circular datastructure called a hyperframe was created to inform the recipient how to interpret the data based on its position in the stream. Synchronization points were then added to add some robustness to this system, since a missed bit will shift the interpretation of every bit thereafter. Given that the data flow is continuous and there are no packets or messages, the CPRI protocol lacks data verification, check sums, acknowledgements, encryption, retrying packets or any other robustness or security mechanism typical of long range communication protocols. Critical data such as reset flags are only acknowledged after multiple hyperframes flag them to prevent false positives at a cost of a delayed reaction [11].

The second consideration when using CPRI is that the data line is continuously busy. As a result, all data communication must be done within the hyperframe and the structure of the hyperframe cannot change. For instance, a sector of the hyperframe is dedicated to transferring error flags and messages. This sector must be sent alongside the rest of the hyperframe for every iteration, including those without errors, wasting bandwidth [11]. Examining the structure of the hyperframe [11], 28.2% of the data transfered is used for synchronization, reset/error flags and messages, and configuration data. These are not needed in most data transfers, artificially inflating the bandwidth requirements.

Wave division multiplexing (WDM) and Quadrature Frequency Spatial Processing (QFSP) in optical fiber lines also allow us to transfer more than one bit per clock cycle. However, since CPRI assumes that one bit is transferred each clock cycle, when these multiplexing techniques are used, each multiplexed sector is assumed to be an independent CPRI communication line. This is useful, for instance, for combining the datalines from multiple cellular antennas into a single optical fiber line by using a different frequency and phase for each antenna [11]. However, this also means we cannot use WDM or QFSM to expand the data bit rate of one RRH unit.

C. eCPRI

Moving CPRI to Ethernet, as proposed by [5], provided new opportunities for CPRI. Primarily, unlike the serial based communication found in CPRI, Ethernet CPRI (eCPRI) is message based, with the data being serialized after arriving at the destination. As a result, different message types can be sent, and one can transition from having to use the hyper frame to transfer all data, even data that may not be currently relevant, to instead sending a variety of packets, each with less unnecessary overhead. Research into these possibilities lead to the initial draft of the eCPRI standard in August 2017 [13] with the latest expanded version of the standard being released in January 2018 [14].

The eCPRI architecture allows up to 255 different message types. Of these, version 1.1 of the standard specifies 8 message types and then reserves a further 56 message types for future extensions of the eCPRI protocol. The user is free to add new custom built message types not covered under the standard to occupy the remaining 192 message types [14]. The ReCPRI extension uses some of the 56 extension message types to add the proposed enhancements to the protocol.

The eight specified message types are as follows [14]:

- A type 0 message is an In-Phase and Quadrature (IQ) data message. This message contains the raw data sampled from the ADC to be interpreted by the BBU, or incoming data from the network to be broadcast in the RRH.
- A type 1 message is a Bit Sequence message. This message allows the user to send a long sequence of raw bits, and is usually accompanied by another message that specifies how those bits should be interpreted.
- A type 2 message is a Remote Transfer Control Data message. This message type is used to specify to the recipient how a group of Bit Sequence Messages should be interpreted.
- A type 3 message is a Generic Data Transfer message. It serves as an unconstrained message type that the user is free to utilize as desired.
- A type 4 message is a remote memory access message. It is used to read or write to the memory of the remote host. It can be used by either device to read or write to a shared memory space.
- A type 5 message is a one way delay measurement message. It is used to profile the latency in the network during the setup phase, which is necessary for calibration purposes to synchronize the RRH and BBU.
- A type 6 message is a remote reset message. It is used to force a remote unit to reset.
- A type 7 message type is an event indication message. It is similar to a remote reset but it raises a flag instead of forcing a reset.

At the start of the packet data is a four byte eCPRI header. This header specifies the message type, the length of the message, the protocol revision, and a single “last” bit used to concatenate messages that are too long to fit in one packet. Afterwards, the structure of each message depends on the message type as specified in [14].

III. The ReCPRI Protocol

We propose the ReCPRI or Reconfigurable Ethernet Common Public Radio Interface protocol. It is fully compatible with the eCPRI v1.1 protocol, utilizing the unused message types to enhance the protocol. It also requires an FPGA device to be added to the radio tower. Since the ReCPRI protocol is built as an enhancement of eCPRI, all of the message types found in eCPRI are also present in ReCPRI. The primary enhancements of ReCPRI over eCPRI are the re-configurable nature of the protocol, and the increased ADC/DAC sample rate using the same Ethernet requirements.

ReCPRI is an SDN enabler, not a user. It provides full hardware and software monitoring and modification capabilities from the BBU software layer as both the hardware and software are live reconfigurable. An SDN app running in the BBU can take advantage of the monitoring and modification capabilities in ReCPRI to perform real-time parameter optimization. The software can, for instance, dynamically change the range of frequencies required to be transferred depending on the usage, to minimize traffic. It can also add new software or hardware functions to the RRH and control when they are run, or provide terminal access to the RRH using ReCPRI messages.

The secondary enhancement involves shifting the responsibilities of the BBU and RRH units. ReCPRI makes use of the FPGA in the RRH unit to perform the time to frequency domain calculations, as well as the data Encoder and Decoder, which was shown to both reduce the required data rate as well as make this rate re-configurable, allowing a telecom company to raise or lower the rate depending on the current need.

A. FFT in RRH

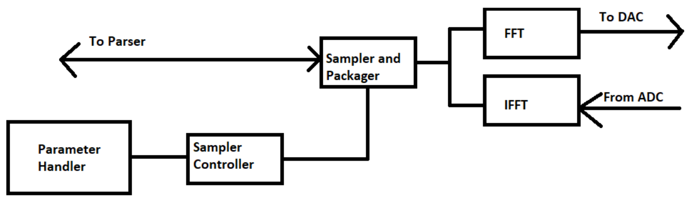

Fig. 2. FFT/IFFT in RRH

One of the main reasons why CPRI and eCPRI requires a large bandwidth is that to guarantee enough accuracy of the output of the FFT, the sample rate of the data transfer between the RRH and BBU has to be much higher than the actual frequency the mobile device sends. As a result, it takes multiple bits of signal to represent one bit of data thereby artificially increasing the required data rate. For ReCPRI (Fig.2) we utilize a Xilinx FFT and IFFT core built on the FPGA to take the sampled data and determine the amplitude and phase of each incoming data frequency. A decision circuit is used to determine the binary value that this signal represents to be able to recreate the signal being transmitted by each cellular device with no overhead. We can then package and send this data through the network to the BBU.

How often this data is sent as well as which frequencies are important is determined by the Sampler Controller. The Sampler Controller is fully configurable from the co-processor allowing its behaviour to be controlled by ReCPRI messages. As a result, we have the ability to modify the output data rate depending on current usage, saving on energy costs and providing extra bandwidth for the other towers. This is different from the CPRI or eCPRI approach where the tower's network data rate does not change regardless of current usage and it is necessary to provision for worst case usage.

B. ReCPRI network message types

A parser and bridge was added to the FPGA to process all incoming and outgoing messages to and from the RRH unit. The bridge deals with the Transmission Control Protocol/ Internet Protocol (TCP/IP) and the User Datagram Protocol/ Internet Protocol (UDP/IP) headers, reliability requirements and handling the required metadata, hiding the network complexity from the rest of the application. It is based on the hardware TCP/IP stack project found in [15] and [16]. The parser then forwards some of the messages to different hardware kernels, such as the IQ message handler, and it forwards other message types to the embedded processor. This processor can make changes to the hardware (i.e. toggle signal values), or make use of the FPGA's reconfigurable nature to modify the circuit in real time. Likewise, each of these components can also send messages through the parser to the bridge to be sent out to the BBU.

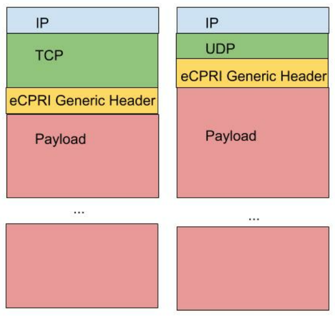

Fig. 3. ReCPRI Structure. (Left) Non-IQ packets, (Right) IQ packets

It was found to be optimal with ReCPRI that UDP/IP is used for IQ data and TCP/IP for all other data types as shown on Fig.3. UDP/IP is a very lightweight standard with very low overhead. However, UDP lacks robustness in that packets can be occasionally lost and unlike TCP/IP, UDP/IP does not retry or timeout. The IQ handler has a built in mechanism to retain accuracy despite packets being lost, and since the majority of the traffic is IQ, the lightweight nature was a desired feature.

TCP/IP has a much higher overhead than UDP/IP but guarantees that all packets will arrive and they will do so in order (by making use of a retry scheme). This is important when transferring a new FPGA bitstream or software script as losing or re-ordering a packet can cause file corruption. Furthermore, since most of the traffic is IQ data, the extra overhead is acceptable. As a result TCP/IP is used for all non-IQ packets.

C. New ReCPRI message types

To ensure reverse compatibility, the ReCPRI standard contains all of the eCPRI message types unchanged. Furthermore, it maintains the same eCPRI generic header as found in [14]. Instead, ReCPRI enhances the baseline standard by adding new message types that enable more control and the ability to reconfigure. Below is a description of the new messages ReCPRI adds:

- Type 8-9 are reserved for future extensions of eCPRI to keep ReCPRI compatible with eCPRI
- A type 10 message is an initialization message. This is the first message sent after establishing a socket connection, and it is always sent from the BBU to the RRH to initialize the global variables and port values, as well as setting the network IP and port numbers.
- A type 11 message is a terminal message. This message type may be enabled or disabled by the user depending on the security requirement. The payload of this message contains a script to be executed in the terminal of the RRH unit or a reply, allowing for changes in the software infrastructure of the RRH (such as installing new libraries). In our testing, we used type 11 messages to open a fully functioning terminal session for the co-processor in the BBU, which communicated via only the ReCPRI port.
- A type 12 message is a script transfer message. This message allows the BBU to extend the software code located in the RRH's co-processor in real time. After receiving this message, the RRH will compile this script and store it locally.
- A type 13 message is a script run message. It executes a previously transferred script within the same process (and thus memory space) as ReCPRI but in a newly formed thread.
- A type 14 message is a toggle IQ message. It is used by the BBU to turn IQ data transfers ON or OFF.
- A type 15 message is a modify message. It is used to change the ReCPRI environment and modify its global variables, bit rates, and even to change the target BBU in real time. We expect this message type to be used to enable SDN in the backhaul network.
- A type 16 message is a bitstream send message. It is used to send a new bitstream to the FPGA. This bitstream can

be used to reprogram one of the partially re-configurable sectors in the FPGA.

- A type 17 message is a bitstream load message. It is used to initiate a write to the FPGA using a previously transferred bitstream. This allows for real time modification of the hardware.

IV. Hardware implementation and results

This section discusses our hardware implementation of ReCPRI on the RRH unit and provides the performance results obtained.

A. Ethernet Subsystem

The Ethernet Subsystem is built on a modified version of the open source Galapagos system developed at the University of Toronto [17]. The Galapagos system was developed to allow a user to specify an arbitrary number of kernels, or application regions, and FPGA devices, where smaller kernels could share a single FPGA. Ethernet messages between kernels are routed using the kernel DEST number that Galapagos converts into the appropriate Ethernet header so that the message is delivered to the correct kernel, without requiring the application region to know physically where each kernel is located.

For ReCPRI, a handler for each message type is built on a separate kernel in the FPGA. The ReCPRI header of all messages is translated to the kernel DEST number using hardware logic. This allows for highly efficient and pipelined packet parsing, since each kernel can directly work on the payload without having to read the packet header (i.e. if the incoming IQ kernel receives a packet, it knows it contains incoming IQ data and does not need to read the header before processing it).

This efficient system has allowed IQ messages in isolation to be sent and received at a combined 9.6 Gbps and non-IQ messages at 6.4 Gbps. The slower speed of non-IQ messages is due to the added TCP overhead. Note, in combination, since a 10Gbps ethernet link is being used, the combined traffic cannot exceed 10Gbps.

In addition to the improved speed, the ReCPRI Ethernet Subsystem provides reconfigurability in the network layer. Using an ReCPRI modify message, followed by an initialization message, a tower migration scheme was implemented which allows the cloud BBU cluster to move an RRH from one BBU to another without disrupting service. This is useful if a particular BBU is overworked (load distribution) or if the cloud has detected a server has gone down. This is unlike CPRI and eCPRI where such a migration scheme was not possible.

B. Processing IQ data

The full ReCPRI data path is shown in Fig.4. An FFT core was used to translate signals between the frequency domain and the time domain. It can send data to the DAC at a rate of 16Gbps and can receive amplitude and phase updates to n frequencies (each 64 bits in resolution) at a rate of $\frac{2.5}{n}$ x10^8

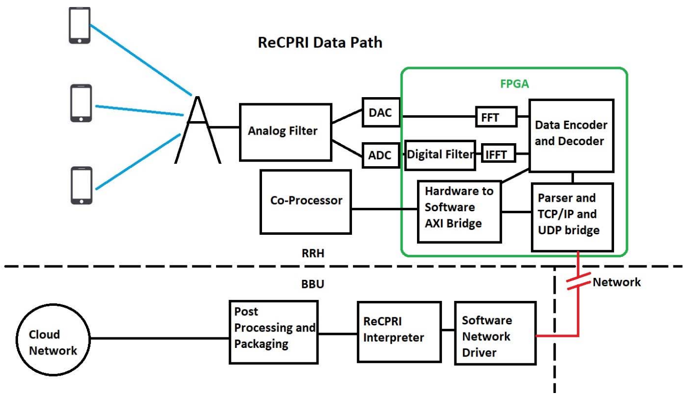

Fig. 4. ReCPRI Data Path

[18] updates per second where n is the number of frequencies being updated (i.e. the number of cellular units transferring data). Our data processing is fully pipelined with an initiation interval of 1 data frame/clock cycle (i.e. no stalls) and as a result is able to interact with the FFT core at its full speed.

Benchmarking tests showed that no stall states are required and 16 Gbps per direction communication was maintained between the ADC/DAC and the FPGA. This is much greater than the current eCPRI approach which uses the same 10 Gbps ethernet line between the RRH and BBU but can only support sending and receiving data to and from the ADC/DAC at a maximum rate of 3.6 Gbps per direction with data samples that are only 30 bits in resolution [19]. This limit comes from the bidirectionally of data, the data line overheads, and the protocol's need for bandwidth reserved for non-IQ messages. ReCPRI separates the ADC/DAC to FPGA communication from the FPGA to BBU communication removing this limitation. The higher sample rate and resolution allows for an increased number of frequencies that a radio tower can service (i.e. more users) and it allows for an increased bit rate for each cellular device, since each amplitude/frequency pair needs to remain stable for a shorter time duration to be correctly profiled by the FFT [20].

For incoming data, the FFT requires an input data bit rate of 16Gbps, but since we communicate with the BBU using a 10Gbps ethernet line – a line shared by the outgoing IQ data and all other ReCPRI traffic – we cannot possibly deliver this data rate by directly connecting the Ethernet to the FFT unit. Therefore, whenever an IQ data message is received from the Ethernet by the IQ handler, a circuit located inside the Data Encoder and Decoder first determines the new amplitude and phase for each outgoing frequency. These values are then stored in shift registers which output the data to the FFT at the required rate of 16Gbps, repeating the sequence until a new IQ message updates it.

Outgoing data is slightly more complicated. The FFT can accept an update for the amplitude and phase of n frequencies at a rate of $\frac{2.5}{n}$ x10^8 updates per second. For each of these frequencies, one first needs to read the specific amplitudes and phases and determine the binary value they are transmitting.

We shall call this circuit the judging circuit and it is located inside the Data Encoder and Decoder. This needs to be determined in less than 4ns per sample to keep up with the FFT. However, our best implementation of the judging circuit was found to require at least 20ns per sample. To solve this, the judging circuit was replicated five times and a task distributor was added to send each sample to a different judge in a round robin approach. A collector was also added that would sample the outputs from each judge also in a round robin approach and output those values in a single data line. The end result is that this larger circuit has a latency of 28ns[4] but a throughput of 250 million samples per second (i.e. it accepts a new sample every 4ns and outputs a result every 4ns); allowing it to keep up with the FFT without stalling.

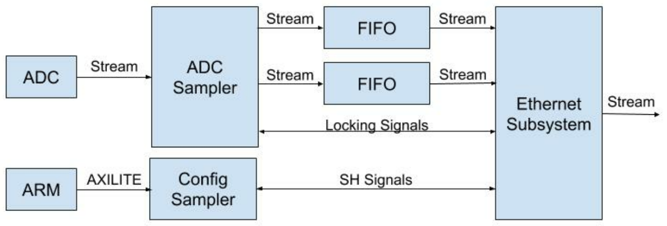

Fig. 5. FIFO shift register subsystem

A FIFO[5] was then used to store the outgoing data. This subsystem was built as shown in Fig.5. The sampler, whose behaviour is controlled from the software layer, will send pulses which tell the Ethernet Subsystem to read the data on the FIFO buffer and transmit it to the BBU via an Ethernet message. The frequency of the timer interrupt and which frequencies are transmitted can be controlled by the BBU using ReCPRI messages, and allows for an application layer such as SDN to control the data bit rate depending on the current usage requirements.

Independent to this, the sampler has a constant stream of incoming data from the ADC. To prevent overlapping, two FIFO devices as well as a locking scheme was implemented (see Fig.5). At every update, the sample alternates which FIFO it writes to, first erasing the data found there. When the packager reads the data from the FIFO, it reads it from the FIFO the sampler is not using. As a result the two systems work independently and no stall states are required.

C. Resource Utilization

This project is implemented and tested on the Zynq UltraScale+ SideWinder100 board featuring a Zynq Ultrascale+ FPGA (Xilinx FPGA model xczu19eg-ffvc1760-2-i) and a quad core ARM Cortex A53 co-processor [21]. The commercial cost of this prototype is under CAD$7000. Fig.6 shows the resource utilization of this project on the FPGA. Across all resource types, this project consumes less than a quarter of the available resources. The remaining three quarters are available

[4]20ns + 8ns for distributor and collector

[5]A FIFO buffer is an ultra high speed shift register memory without addressing where the first item written is the first item read

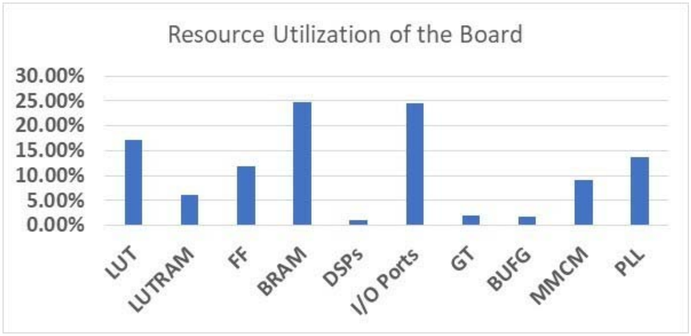

Fig. 6. ReCPRI FPGA resource utilization

for additional features that the telecommunications company may wish to add, leaving room for full customization of the hardware.

V. Software implementation

The radio tower consists of both the FPGA processor and an ARM processor. The processor's purpose is to handle all ReCPRI messages that the FPGA circuit is unable to handle, to compile and run user provided scripts, to configure the hardware (i.e. set or change parameters), and to update the FPGA bitstream when necessary.

A. Reprogramming the FPGA

A complication in trying to use the co-processor to reprogram parts of the FPGA is that FPGAs are usually reprogrammed using a separate JTAG connection. This connection processes the bitstream and sends it to the FPGA. Utilizing this JTAG, however, requires that an operator travels to the FPGA site for all hardware changes and re-flashes the FPGA. Xilinx FPGAs, however, can also be programmed using PCAP, which allows FPGA reprogramming from the software layer. The co-processor sends an AXI-Stream signal containing the new bitstream to the FPGA's AXI-PCAP bridge (located in the hardened portion of the FPGA) which then uses that bitstream to reprogram the re-configurable portion of the FPGA. As future work, Xilinx FPGAs also enable partial reconfiguration over PCAP, and we hope to extend our system to do this. In this mode, we can target a smaller bitstream to modify only one sector of the re-configurable portion of the FPGA, and the AXI-PCAP bridge would reconfigure that sector without requiring that any other sectors to stop running [22]. This is useful when the user wants to add a new hardware feature to the radio tower without stopping or delaying the flow of IQ data. Using ReCPRI, a programmer or a software app can use the same data lines to send a new bit stream to a radio tower and can, with a single command, change the hardware layer remotely.

B. Send and Run Script messages

In the same way, the software layer can also be reconfigured using ReCPRI messages. The Send and Run Script message types were added to allow the user to send a new C++ script and to remotely control when that script is run. This script will run in the same process space as the ReCPRI control and has full access to the memory system and messaging system allowing the application layer to add new functionality to the radio tower. To run this new script within the same process space of the function calling it, we compile the script not as a script but as a library. We can then use dynamic loading to load this library in real time. When the execute message is received, the software layer creates a new thread and runs the starting function of this library. As a result, the script is run in the same process space allowing it access to the same memory region and providing all the ports required to interface with the network and with the hardware providing this script with full control over the system. As a test, we added an environmental monitor to the radio tower, connecting its output to one of the hardware pins. We then used ReCPRI to send and run a script that would at fixed time intervals poll the environmental monitor and relay a warning to the BBU when the values were outside of the safe range. This functionality was added without requiring any tower downtime or requiring IQ data to stop.

C. Hardware to Software communication

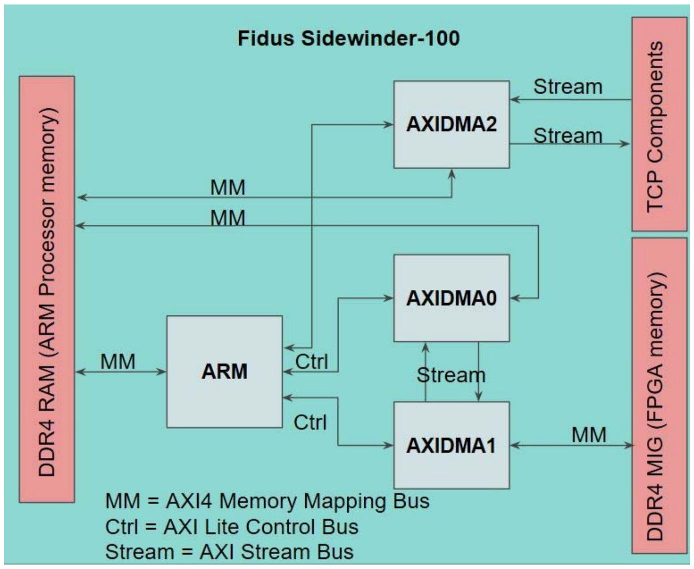

Fig. 7. Memory Interface

Hardware to Software communication, as shown in Fig.7, consists of two parts. Firstly, it is desired that the co-processor has access to the FPGA's memory and that the FPGA has access to the co-processor's memory. Similarly, a pipe should also exist between the two devices to allow the hardware to forward packets to the co-processors, and also to allow the co-processor to send out a packet into the network via the hardware.

The Xilinx AXI DMA IP Core [23] acts as a bridge between the DDR memory and the FPGA programmable logic. It performs the conversion between AXI stream signal and memory map signal, giving hardware the capability to perform direct memory access (DMA). Both types of Hardware to Software communication leveraged the Xilinx AXI DMA Driver [24]. This driver abstracts the control flow of the Xilinx AXI DMA IP Core in the kernel level and exposes high level APIs at the user level. This allows designers to have user level applications control the hardware DMA engines.

For FPGA and co-processor memory sharing, 2 DMA cores are placed in the hardware. One of the cores has its Memory Map Bus connected to the DDR memory on the co-processor side, the other one is connected to the DDR on the hardware side. The 2 DMA cores are also interconnected using 2 high speed AXI Stream serial lines for bidirectional data transfers. Finally, the AXI Lite Bus of both cores, used to send control signals, is multiplexed and connected to both the co-processor and the FPGA logic allowing both devices to initiate data transfers between the two memories.

For the TCP pipe, another DMA core is used. The DMA also has a multiplexed AXI Lite Bus connected to the co-processor and the hardware allowing both to initiate a data transfer. The Memory Map Bus is connected to the DDR on the processor side, and AXI Stream Buses are connected to the network components. Whenever a TCP packet belonging to the co-processor enters the hardware, an interrupt and the data is sent to this DMA core and the OS would detect the interrupt and control the DMA core to move the streaming packet to the DDR memory. Likewise, the co-processor could use this OS driver to send a packet from the co-processor via the DMA core to the hardware. The hardware would send this packet to the parser who would forward it to the bridge to be sent out in the network.

VI. Related Work

Guo et al. [25] approach the bandwidth problem by proposing a compression/decompression algorithm for CPRI IQ traffic. The algorithm would actively seek which bits are not useful and would eliminate them. The results, however, showed a small difference in the signal to noise ratio, and some total data reduction, but there was an increase in signal errors as some of the eliminated bits were false positive containing useful data. They concluded acknowledging limited applicability of their algorithm in a general setting.

Monti et al. [26] investigate the division of tasks between the remote radio head and the baseband unit when using eCPRI. Their analysis leads them to the conclusion that the more pre-processing is done in the tower, the lower the required data communication between the tower and the BBU. However, they also find that the more pre-processing is done at the tower, the harder it is to implement more complicated coordination, synchronization, and configuration actions, a problem resolved in ReCPRI by using reconfiguration. They then propose a three stage solution where a server is added between the radio tower and the BBU to perform the pre-processing so that the large bandwidth signal travels a shorter length. This solution, however, increases the network costs as more servers are required, increases the latency as the signal has to be processed in three places, and it does not decrease the traffic between the tower and the first server.

Zeng et al. [27] make use of a DSP processor and four Xilinx Virtex 7 FPGAs to combine the eCPRI data from multiple towers and send them on a single, higher bandwidth, data line. They made use of the new 120Gbps Ethernet lines to converge the data from sixteen radio towers. They would then use another cluster of four FPGAs and a DSP to re-separate the data back into 16 individual channels for the central server to process. For existing infrastructure, this is an expensive solution as it would require purchasing clusters and laying the infrastructure to support this 120Gbps Ethernet line. For new installations, however, this is less expensive than laying individual 10Gbps lines for each tower. This work is agnostic of the underlining protocol and would work in an ReCPRI network. If this is used alongside ReCPRI, the SDN element of ReCPRI could also be used to dynamically allocate the bandwidth for each tower as utilization changes.

Zou et al. [28] created a proof of concept prototype for the Xhaul network based on the model proposed by Olivia et al. [29]. The Xhaul network proposes connecting the radio towers directly to the cloud network. The network would still contain compute nodes performing the various tasks that the BBU would have previously performed, but this network would also enable radio towers to send data among themselves. Xhaul's cloud-based implementation would also help with resource sharing and efficient network routing since each tower can chose from a variety of processing nodes located in the network [29]. This technology is still early in its development, and most implementation questions have not been answered [28]. However, the increased control and visibility of ReCPRI, as well as its data pre-processing, would likely simplify the development of Xhaul and other similar proposals compared to having to implement such a network using eCPRI.

VII. Conclusions

We developed the ReCPRI protocol to replace the existing eCPRI standard in the telecommunications backhaul network. This new protocol allows for gradual implementation, as it is fully compatible with eCPRI. This protocol enables SDN-like control and monitoring, allowing for smarter, more efficient, network traffic. It also simplifies future updates and feature extensions as both the software and the hardware of each radio tower can be modified from a central server without requiring any tower downtime, or requiring an operator to visit the tower. This is all done while reducing the required data traffic between the tower and the central server and increasing the number of users that can be serviced from each tower.

VIII. Acknowledgment

We would like to acknowledge Xilinx and Fidus for tools and hardware support. We would also like to thank Naif Tarafdar and Daniel Ly-Ma for their help with using the Galapagos infrastructure and the TCP/IP core.

References

[1] F. Boccardi, R. Heath, A. Lozano, T. Marzetta, and P. Popovski, "Five disruptive technology directions for 5g," *IEEE Communications Magazine*, pp. 74–80, 2014.

[2] B. Leukert, T. Kubach, C. Eckert, K. Tsutsumi, M. Crawford, and N. Vayssiere, "Iot 2020: Smart and secure iot platform," *IEC White papers*, pp. 1–181, 2016.

[3] Machina Research, "Global m2m market to grow to 27 billion devices, generating usd1.6 trillion revenue in 2024," 2015. [Online]. Available: https://machinaresearch.com/news/global-m2m-market -to-grow-to-27-billion-devices-generating-usd16- trillion-revenue-in-2024/

[4] L. Rusch, "Introduction to c-ran in 5g," *Universit Laval*, 2018.

[5] J. Gomes, P. Chanclou, P. Turnbull, A. Magee, and V. Jungnickel, "Fronthaul evolution: From CPRI to Ethernet," *Optical Fiber Technology*, vol. 26 part A, pp. 50–58, 2015.

[6] R. Boutaba, "Introduction to sdn," *University of Waterloo*, 2018.

[7] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, pp. 16–19, 2013.

[8] D. Kreutz, F. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, 2015.

[9] P. Rost, C. Bernardos, A. Domenico, M. Girolamo, and M. Lalam, "Cloud technologies for flexible 5g radio access networks," *IEEE communications*, pp. 68–76, 2014.

[10] A. Checko, H. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. Berger, and L. Dittmann, "Cloud ran for mobile networks – a technology overview," *IEEE communication surveys and tutorials*, vol. 17, pp. 405–426, 2015.

[11] Ericsson, NEC, Nokia, Huawei, and Actel, "Cpri specification v7.0," pp. 1–128, 2015.

[12] W. COCHRAN, J. Cooley, D. Favin, H. Helms, R. Kaenel, W. Lang, G. Maling, D. Nelson, C. Rader, and P. Welch, "What is the fast fourier transform," *Proceedings of the IEEE*, vol. 55, pp. 1664–1674, 1967.

[13] Ericsson, NEC, Nokia, and Huawei, *eCPRI Specification V1.0*, CPRI group, 2017.

[14] Ericsson, Nokia, NEC, and Huawei, *eCPRI Specification V1.1*, CPRI group, 2018.

[15] D. Sidler, G. Alonso, M. Blott, K. Karras *et al.*, "Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," *FCCM*, 2015.

[16] D. Sidler, Z. Istvan, and G. Alonso, "Low-latency tcp/ip stack for data center applications," *FPL*, 2016.

[17] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network fpga clusters in a heterogeneous cloud data center," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 237–246, 2017.

[18] XILINX, *Fast Fourier Transform v9.0 LogiCORE IP Product Guide*, 2017.

[19] T. Mustala and O. Klein, "Common public radio interface ecpri overview," *IEEE 802.1 Working Group*, 2015. [Online]. Available: http://www.ieee802.org/1/files/public/docs2017/ cm-mustala-eCPRI-Overview-0917.pdf

[20] J. Whittaker, "On the cardinal function of interpolation theory," *Proceedings of Edinburgh Math Society*, pp. 41–46, 1927.

[21] XILINX, "Fidus sidewinder 100 product description," 2018.

[22] C. Kohn, "Partial reconfiguration of a hardware accelerator with vivado design suite for zynq-7000 ap soc processor," *XILINX Application Notes*, 2015.

[23] Xilinx, "Axi dma v7.1 logic core ip product guide," *Vivado Design Suite*, 2018.

[24] B. Perez, "Xilinx axi dma," https://github.com/bperez77/ xilinx_axidma, 2018.

[25] B. Guo, A. Tao, and D. Samardzija, "Lte/lte-a signal compression on the cpri interface," *Bell Labs Technical Journal*, vol. 18, pp. 117–133, 2013.

[26] P. Monti, Y. Li, J. Martensson, M. Fiorani, B. Skubic, Z. Ghebretensae, and L. Wosinska, "A flexible 5g ran architecture with dynamic baseband split distribution and configurable optical transport," *ICTON*, 2017.

[27] H. Zeng, X. Liu, S. Megeed, and F. Effenberger, "Dsp for high-speed wireless convergence," *Optical Society of America*, 2018.

[28] J. Zou, A. Magee, M. Eiselt, A. Straw, T. Edwards, P. Wright, and A. Lord, "Demonstration of x-haul architecture for 5g over converged sdn fiber network," *Optical Society of America*, 2018.

[29] A. Olivia, X. Perez, A. Azcorra, A. Giglio, F. Cavaliere, D. Tiegelbekkers, J. Lessmann, T. Haustein, A. Mourad, and P. Iovanna, "Xhaul: Towards an integrated frontaul/backhaul architecture in 5g networks," *IEEE Wireless Communications*, vol. 22, pp. 32–40, 2015.