Comparative Analysis of Processing Latency and CPU Efficiency in FPGA-Based FEC Acceleration

Aerman Tuerxun
Graduate School of Engineering
The University of Tokyo
Tokyo, Japan
armantursun@g.ecc.u-tokyo.ac.jp

Akihiro Nakao
Graduate School of Engineering
The University of Tokyo
Tokyo, Japan
nakao@nakao-lab.org

***Abstract*—As Radio Access Networks (RANs) evolve towards more intelligent and software-defined paradigms, an increasing number of workloads are being offloaded to hardware accelerators. A significant challenge in this context is to minimize CPU consumption while ensuring low and stable latency in processing acceleration. Tailoring load allocation to meet varying latency requirements in different application scenarios is essential. In this paper, we present an application of RF Network on Chip (RFNoC) technology for FPGA-based acceleration of Low-Density Parity-Check (LDPC) code and Polar code. We conduct a comparative analysis of CPU and FPGA processing performance in OpenAinInterface (OAI) platform, with and without the Data Plane Development Kit (DPDK), to determine optimal load allocation for diverse data requirements. The findings indicate that RFNoC serves as an effective FPGA accelerator for the LDPC process, offering up to a fivefold increase in acceleration and significantly reducing processing delay jitter. Furthermore, the experimental outcomes provide a foundation for future research endeavors, specifically in the realm of efficient load optimization strategies.**

***Index Terms*—Hardware Acceleration, LDPC Code, Polar Code, RFNoC, OpenAirInterface, CPU Consumption**

I. Introduction

The proliferation of softwarization across diverse communication infrastructure equipment, spanning Software-Defined Networking (SDN) like OpenFlow [1], programmable data-planes utilizing technologies like P4 [2] and FLARE [3], and Radio Access Network (RAN) solutions like the OpenAirInterface (OAI), as well as Core networks such as Free5GC, has become a prevailing trend. This widespread adoption is primarily attributed to the compelling advantages it offers, including reductions in Capital Expenditure (CapEx) and the seamless integration of new functionalities. With the democratic development of the O-RAN specification and the emergence of private 5G networks welcoming new stakeholders into the market, softwarization aligns seamlessly with this landscape, given its capacity to deliver both CapEx reduction and operational flexibility. However, a central challenge in this softwarization journey lies in the intricate balance between flexibility and performance. Presently, the prevailing approach involves utilizing general-purpose processors as a foundational element and augmenting them with hardware accelerators to optimize performance.

There are two primary strategies for offloading computational workloads to hardware components. The first strategy is referred to as ”Look-aside Hardware Acceleration,” in which a designated function hardware accelerator operates independently, allowing the CPU to remain unburdened until the accelerator completes its assigned tasks. Look-aside accelerators offer the advantage of allowing Commercial Off-The-Shelf (COTS) servers to make informed decisions regarding which functions to offload onto the accelerator. Intel is a prominent supporter of the Look-aside approach, and it has fully integrated acceleration capabilities into its 4th Generation Intel Xeon Scalable processors through Intel vRAN Boost [4]. The second approach is known as ”In-line Hardware Acceleration,” where certain or all components of the Layer 1 processing pipeline can be offloaded onto a dedicated accelerator card. In high-bandwidth open RAN applications, complete offloading of L1 processing may become necessary, necessitating minimal latency and core utilization. This approach garners significant support from companies such as Marvell, Qualcomm, and Nokia [5]–[7].

Alongside the battle for acceleration dominance, a prevalent myth persists, suggesting that all complex functions should be entirely offloaded to the accelerator. This perspective often overlooks a crucial aspect—the contextual environment in which the accelerator operates. Neglecting this contextual consideration can lead to some significant drawbacks, ultimately negating the cost reduction benefits of softwarization: the inefficient allocation of FPGA resources, the potential over-engineering of FPGA, and the potential for suboptimal performance resulting in increased overhead and costs.

In response to these challenges, this paper posites for the importance of ”contextually selective offloading of RAN functions” through the implementation of various configurations for RAN function offloading via RFNoC (RF Network-on-Chip). Through an extensive analysis of these configurations, this work endeavors to pave the way for a novel optimization paradigm: the contextual optimization of RAN softwarization.

This paper’s principal contributions are summarized as follows:

- A novel RFNoC-based FPGA offloading approach for LDPC and Polar codes is introduced, which achieves a

significant fivefold acceleration compared to traditional CPU processing.
- A comprehensive comparative analysis of latency and CPU consumption is conducted within an open-source RAN testbed system. The analysis underscores the imperative need for tailored acceleration to cater to a spectrum of data requirements.
- Future research paths and methods are discussed, aiming at improving load allocation strategies and fostering more efficient RAN operations.

The rest of the paper is organized as follows. Section II discusss the related work. Section III introduces the RFNoC technology and offloading design of LDPC and Polar with high-level architecture and IP wrapper design. Section IV evaluates the design and analyzes the results of LDPC and polar encoding and decoding compared with software-based processing. Section V discusses the possible future directions of optimal hardware acceleration. Section VI briefly concludes this paper.

II. Related Work

In the realm of offloading techniques for varying operational contexts, several studies stand out for their contributions. Papatheofanous et al. have developed an FPGA-based LDPC decoder architecture, delivering a significant speedup—over 13 times faster than traditional methods for LDPC decoding. However, their LDPC encoder shows divergent performance characteristics [8], [9]. Chance et al. innovate a GPU-based LDPC decoding accelerator, seamlessly integrates into the OpenAirInterface (OAI) [10]. Additionally, Tram et al. introduce an ASIC-based approach for LDPC encoding [11]. It is noteworthy that the aforementioned researches, while pioneering in their respective areas, either do not fully address CPU efficiency in the offloading process or lack integration into RAN systems for comprehensive performance evaluation. This gap underscores the need for holistic studies that consider both computational efficiency and practical applicability in real-world RAN environments.

In addition to hardware-based approaches, there is a segment of research focused on software-based enhancements for LDPC decoding. Notably, in studies such as [12] and [13], the authors employ Advanced Vector Extensions (AVX) Single Instruction, Multiple Data (SIMD) instructions, achieving significant acceleration in decoder performance. Despite these advancements, when compared to hardware accelerators, these software-based solutions exhibit inefficiencies, particularly in processing multiple code blocks and in handling high-throughput scenarios.

Additionally, Wei et al. conduct an analysis of the overhead associated with various physical layer functions within the OAI framework. Their findings reveals that uplink traffic imposes a significantly higher processing cost on the physical layer compared to similar traffic rates in the downlink [14]. Kundu et al. also present a GPU-based practical implementation of inline hardware acceleration for both uplink and downlink [15]. These work have primarily focused on demonstrating the acceleration outcomes of individual functions, with limited attention devoted to a comprehensive analysis of the hardware acceleration requirements for various functions.

III. LDPC Offloading Design

The FPGA component of the system is realized within the USRP X410, featuring the RFSoC XCZU28DR FPGA board and employing commercially available Xilinx LDPC IP cores [16] and Polar IP cores. The interconnection between the FPGA and a workstation, responsible for executing software-based physical layer processing through the default UHD driver, is established via the SFP+ interface. Notably, the UHD software API provides robust support for application development across the varies USRP SDR products. This versatile API enables a seamless transition of applications onto various SDR platforms, thus enhancing flexibility and interoperability in the system.

A. RFNoC

The implementation of the LDPC and Polar IP core within the SDR system is achieved through the utilization of RFNoC technology, as shown in Braun et al. [17]. RFNoC represents an architectural framework that capitalizes on FPGA technology to significantly augment the performance and adaptability of SDR systems. This innovative framework facilitates the bespoke creation of signal processing chains tailored to specific requirements, all within the FPGA environment. RFNoC effectively harnesses FPGA capabilities to establish an interwoven network of processing blocks, thereby establishing an on-chip communication infrastructure. This infrastructure, akin to a dedicated pathway, serves the pivotal role of enabling seamless data streaming and processing directly within the FPGA fabric. It is noteworthy that RFNoC serves as an invaluable tool across a diverse spectrum of SDR applications.

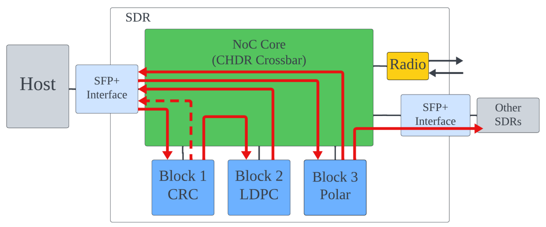


Figure 1: RFNoC Data Flow

Data is encapsulated within a Condensed Hierarchical Datagram for RFNoC (CHDR) [18], and this encapsulated data is routed through the CHDR Crossbar, as illustrated in Figure 1. This routing capability enables the establishment of connections between blocks situated on the same FPGA, between blocks and the host system, or even between blocks on different USRPs via transport adapters. Importantly, the routing configuration within the RFNoC Blocks is dynamic, allowing for adjustments during runtime, all of which can be orchestrated by software control. This dynamic routing not only minimizes data transmission latency but also facilitates the independent implementation and testing of various physical layer functions. This capability proves advantageous for the dynamic switching between the look-aside and in-line offloading.

B. High-level Architecture

The hardware accelerator is integrated into the OpenAirInterface (OAI) testbed. OAI represents an open-source initiative dedicated to the advancement of both software and hardware platforms tailored for wireless communication systems. Its primary focus encompasses the design and prototyping of 4G (LTE) and 5G networks. OAI exhibits a high degree of customizability, rendering it an invaluable resource capable of being tailored to meet specific project requirements. Consequently, it serves as a valuable tool for the rigorous testing and deployment of pioneering wireless solutions.

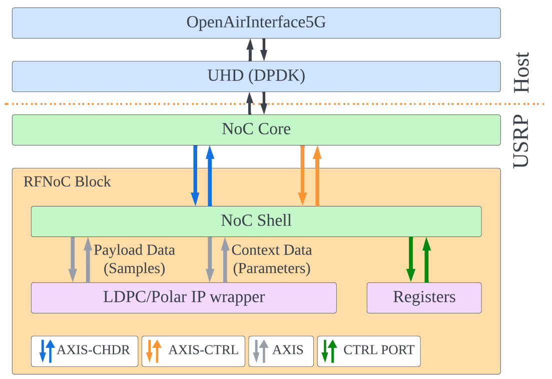


Figure 2: High-level Architecture of LDPC Offloading

The high-level architecture is depicted in Figure 2. In this configuration, OAI operates as the application responsible for invoking LDPC and Polar functions, while the USRP assumes the role of handling LDPC and Polar computations. These two components interface with one another through the utilization of SFP+ Ethernet adapters, thereby facilitating high-speed data streaming between them. To realize this architecture, we opt for the default UHD as the driver of choice, which extends comprehensive APIs to enable seamless integration with the software. Additionally, we leverage DPDK (Data Plane Development Kit) for its polling-based approach, tailored to deliver low-latency streaming capabilities.

The LDPC and Polar RFNoC blocks is interconnected with the NoC Core, which is automatically generated within the framework. This core plays a pivotal role in overseeing the CHDR crossbar and orchestrating routing control among various entities, including inter-block connections and connections between blocks and the host system. Within this architecture, two distinct types of AXI buses are deployed, AXIS-CHDR and AXIS-CTRL. The AXIS-CHDR encompasses data plane functionalities, serving as the conduit for data packets. The width of these data packets is custom-defined, with a 128-packet width employed here for enhanced compatibility with Xilinx IP components. Each data packet accommodates a 64-bit timestamp and user-defined metadata. The metadata capability, an advanced feature of RFNoC, permits packet-specific definitions, which can be used to carry block-based configuration parameters. Additionally, to manage control plane functionalities, AXIS-CTRL is employed. This component facilitates the streaming of control packets for the read and write operations of user registers.

The NoC Shell, an automatically generated component, is an integral part of the RFNoC block and is designed to effectively packetize and de-packetize both data and control packets. Within this design, CHDR packets are segregated into two distinct streams—Payload AXI-Stream and Context AXI-Stream. The Payload AXI-Stream includes sample data, encompassing information bits, padded bits, and parity check bits. Conversely, the Context AXI-Stream is dedicated to CHDR header and metadata transmission. As previously mentioned, metadata encapsulates block-specific configuration parameters essential for LDPC and Polar IP operations. The IP module first configures itself based on these parameters and subsequently processes the sample data from the payload stream. Following channel coding processing, the NoC Shell adeptly reconstitutes all data into a single CHDR packet and dispatches it to the target endpoint. Control packets, on the other hand, are directed to the control port to facilitate user register read/write operations.

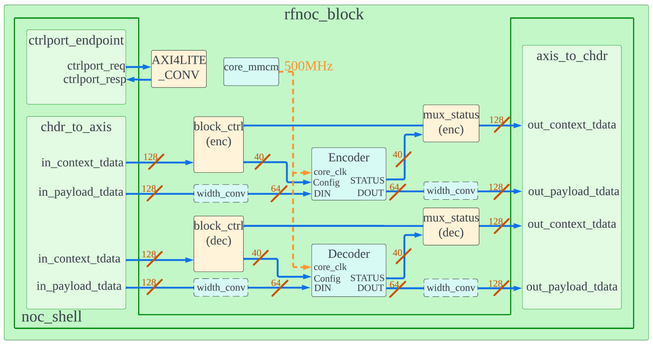


Figure 3: Internal Design of RFNoC LDPC Block

C. LDPC and Polar Block Design

The internal design of the LDPC and RFNoC block is shown in Figure 3, revealing three distinct components custom-developed for LDPC and Polar blocks functionality. First among these is AXI4LITE_CONV, dedicated to facilitating read and write operations targeting Xilinx IP or Polar IP registers. block_ctrl plays a pivotal role in extracting configuration parameters from the metadata, which is streamed through in_context_tdata, and subsequently conveys this essential data to the LDPC or Polar core. Following the configuration phase, the LDPC or Polar IP module proceeds to accept data from in_payload_tdata and executes channel coding operations. Upon exiting the IP core, status data is appended to the context data as metadata by mux_status. Subsequently, noc_shell undertakes the task of packetizing all data and preparing it for transmission back to the host. There is also width conversion processes added to the Polar block in order to support our Polar IP cores. And for better integrating the IP cores into RFNoC framework, we use core_mmcm block to generate demanded clocks.

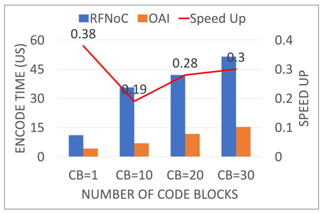
Leveraging the well-established RFNoC framework and readily available LDPC and Polar IPs as commercial, off-the-shelf solutions, our efforts primarily revolves around the management of FPGA registers and parameters. On the host side, we introduce modifications to the UHD APIs to enable the dynamic insertion of parameters into the metadata during runtime, in tandem with the data transmission process. This utilization of RFNoC technology has streamlined the implementation of FPGA-based channel code offloading, requiring minimal development effort.

IV. Evaluation and Analysis

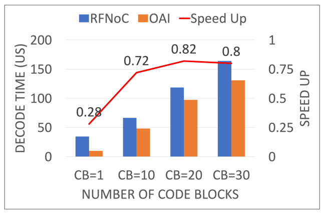
The FPGA system is effectively deployed on the USRP X410 platform, operating at a clock frequency of 500 MHz for LDPC wrapper and 200MHz for Polar wrapper. Meanwhile, OAI is executed on a workstation powered by Ubuntu 20.04, featuring an Intel(R) Core(TM) i9-9900K CPU operating at 3.60 GHz, and equipped with 16GB of RAM.

A. Polar Encoding and Decoding

Polar codes find application in the channel coding of Uplink Control Information (UCI) and Downlink Control Information (DCI). To effectively demonstrate the FPGA accelerator's impact on Polar code processing for both uplink and downlink scenarios, we conducted evaluations focusing on the encoding latency for uplink and the decoding latency for downlink. The hardware acceleration results are presented in Figure 4.



(a) Polar Encoding Latency



(b) Polar Decoding Latency

Figure 4: Polar Coding Latency in FPGA and OAI

The outcomes of our evaluation indicate that both the Polar encoder and Polar decoder exhibit longer processing times in comparison to their software-based counterparts. Notably, when dealing with the pipelined processing of multiple data blocks, while this approach does enhance performance compared to single-data-block processing, it still falls short of the efficiency achieved by software-based implementations.

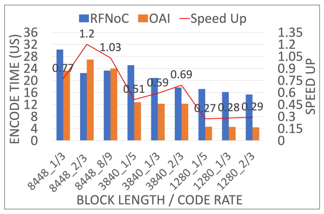
It is noteworthy that Polar code operations typically involve relatively small data blocks, which may not be optimally suited for offloading onto FPGA platforms. Additionally, in both the OAI and FPGA implementations, the Polar function encompasses additional components like Rate-matching and CRC calculation.

B. LDPC Encoding and Decoding

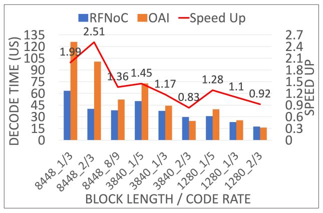
1) Single Block Latency: Figure 5 demonstrates the execution time of the LDPC decoding process, comparing the software-based implementation within OAI and the FPGA-based approach. The evaluation spans testing the execution time across 5 iterations while varying data lengths, coding rates using *ldpctest* in OAI.

In Figure 5a, the observed results demonstrate that for a single code block, software-based encoding exhibits superior efficiency compared to RFNoC offloading. Nevertheless, with the increment in block size, the processing time of RFNoC gradually converges to a level nearly parallel to that of software-based encoding. It is critical to acknowledge that the data encoding outputs from RFNoC are measured in bits, whereas in the OAI framework, each bit corresponds to one byte. Despite this, the additional time incurred for converting from bits to bytes in the software-based method contributes negligible increase to the overall data transfer latency.

As depicted in Figure 5b, the speedup achieved through the utilization of FPGAs for LDPC decoding is notably significant, particularly evident when processing large data blocks, where processing time for a single data block can be reduced by more than twice. However, it's important to note that for relatively small data blocks, the acceleration effect is less pronounced, and in some cases, it even lags behind the performance of software-based processing. This phenomenon is primarily attributable to the substantial time consumed by data transfers between the host and the SDR, thereby amplifying the overall processing latency.



(a) LDPC Encoding Latency



(b) LDPC Decoding Latency

Figure 5: LDPC Single Block Latency in FPGA and OAI

2) ULSCH and DLSCH Processing: To evaluate the efficacy in handling multiple blocks, we employ the *ulschsim* and *dlschsim* utilities provided by the OAI. As depicted in Figure 6, in the DLSCH transmission procedure as implemented in the OAI platform, the TB undergoes a segmentation process into several code blocks, with each block being appended with a Cyclic Redundancy Check (CRC). Subsequently, each segment is subject to Low-LDPC encoding, utilizing a multi-threading technique. This technique is particularly advantageous in our design, as it leverages the inherent capabilities of the OAI thread pool to enhance throughput. This is achieved by distinctly segregating the transmit and receive phases, thereby streamlining the process. Post-LDPC encoding, each code block is processed through rate-matching and interleaving, after which these blocks are concatenated and multiplexed for transmission. Similarly, in the ULSCH transmission process, same method are used for LDPC decoding. This methodology not only addresses the non-thread-safe nature of the RFNoC send and receive APIs but also maximizes the efficiency inherent in the OAI thread pool management.

Such a design approach effectively utilizes the multi-threaded architecture of OAI, thereby optimizing the overall transmission process. This optimization is critical in ensuring that both DLSCH and ULSCH transmissions are conducted with enhanced efficiency and throughput, capitalizing on the advanced capabilities of the OAI platform and the integrated hardware acceleration. In our evaluation, the measured processing latency encompasses the total duration from the initiation of the ULSCH or DLSCH functions, thereby including the time expended on CRC, LDPC processing, rate-matching and interleaving.

Figure 7 and Figure 8 present a comparative analysis of the runtime performance of LDPC encoding and decoding across different code rates, in both RFNoC and OAI environments. The generation of distinct MAC layer TBs is done through variations in Modulation and Coding Scheme (MCS) and Resource Blocks (RB). The results indicate that the multi-threading capable LDPC framework developed in this study demonstrates performance on par with, or surpassing, that of the OAI.

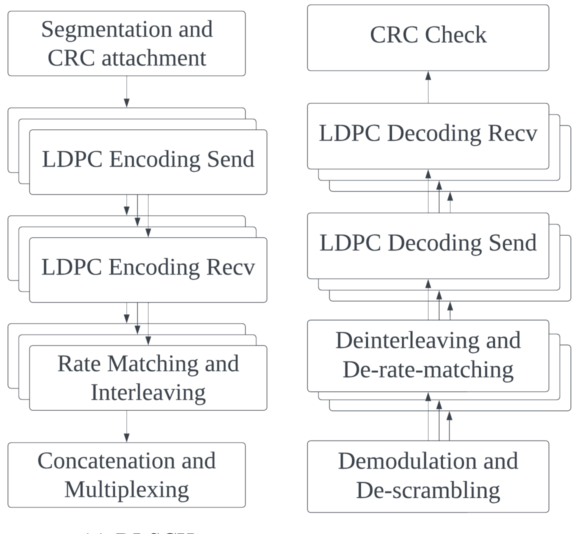

Segmentation and CRC attachment
LDPC Encoding Send
LDPC Encoding Recv
Rate Matching and Interleaving
Concatenation and Multiplexing
(a) DLSCH
CRC Check
LDPC Decoding Recv
LDPC Decoding Send
Deinterleaving and De-rate-matching
Demodulation and De-scrambling
(b) ULSCH

Figure 6: Multi-Blocks Processing in RFNoC Accelerated OAI

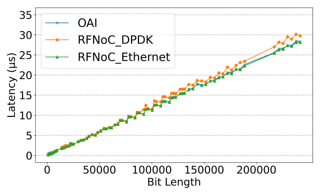
In encoding tests, the RFNoC encoder showcases processing capabilities comparable to OAI, albeit with a slight reduction in efficiency when handling larger terabytes. However, it emerges as a viable alternative to the OAI encoder. Without the utilization of DPDK, RFNoC's performance is approximately equivalent to scenarios where DPDK is employed. It is postulated that the encoded data, being generally smaller in size compared to Log-Likelihood Ratio (LLR) data, diminishes the relative advantage of DPDK in low-throughput data contexts. Nonetheless, it is conjectured that in scenarios involving multiple UEs or Radio Units (RUs), DPDK will significantly enhance throughput. Conversely, it can also be seen that software-processed LDPC encoding adequately fulfills low-latency requirements, obviating the need for hardware acceleration. This insight suggests the feasibility of allocating encoding tasks to the CPU, thereby freeing up FPGA resources for more critical processes, such as decoding.

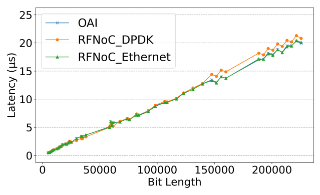
The RFNoC decoder exhibits markedly superior performance compared to the OAI, particularly notable in the acceleration of large TBs. Concurrently, it has been observed that for Base Graph 2 (BG2), specifically for TBS less than 3840, the processing latencies of RFNoC and OAI are comparable. Notably, OAI demonstrates a slight edge over RFNoC in scenarios where the code rate is 1/5. This finding underscores the capability of the CPU to sufficiently meet the highest performance demands of the LDPC decoder in BG2 configurations. Furthermore, it is discerned that when throughput requirements are moderate, the latency objectives can be satisfactorily met without resorting to the utilization of DPDK.

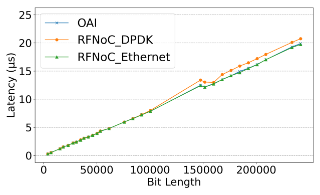
3) CPU Consumption: To evaluate the CPU consumption associated with LDPC operations during the operation of OAI gNB, we employ both Perf and psutil as measurement tools. Perf, a profiling tool, collects statistics about the functions executed most frequently, reporting the percentage of samples corresponding to each function. However, the percentages do not reflect the total CPU time consumed by specific functions. To facilitate comparative analysis across different experiments where total CPU usage may vary, a normalization of Perf data is essential. For each experiment, psutil is utilized to monitor the CPU usage of the processes on a second-by-second basis. This monitoring provides a comprehensive measure of the total CPU resources consumed by OAI. Subsequently, the sample frequency of each function is multiplied by the total CPU usage, as measured by psutil. We run the OAI in *phy-test* mode, in which the UE and gNB generate random uplink and downlink data streams. The results of the CPU consumption are depicted in Figure 9.

As shown in Figure 9a, first, we compare the CPU consumption of LDPC-related functions with and without RFNoC. The results are expected, DPDK takes up a single CPU core, making the overall consumption 108.8%, while without RFNoC there is only 2.044% overall consumption. We then randomly choose half of the data to stay in the CPU for execution and the other half offload into the RFNoC. We find that as long as the DPDK is running, processing the data in both the CPU and FPGA does not additionally increase the CPU's consumption by a significant amount, only 0.15% more than processing it exclusively in the FPGA. This result suggests that we can achieve optimal latency or optimal FPGA utilization by keeping some processing in the CPU.

In Figure 9b, a comparative analysis of CPU consumption is presented for scenarios with and without the use of the DPDK. The findings indicate that while the utilization of the UHD feature results in a marginal increase in CPU consumption, approximately 5%, it significantly lowers the CPU


(a) CodeRate=1/3


(b) CodeRate=2/3


(c) CodeRate=8/9

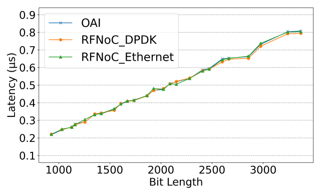
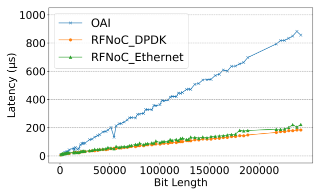
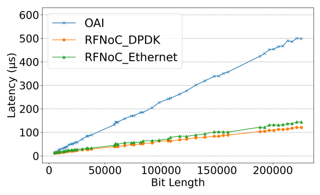
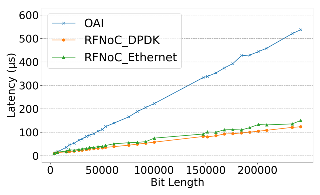

(d) CodeRate=1/5

Figure 7: LDPC Encoding Latency in FPGA and OAI


(a) CodeRate=1/3


(b) CodeRate=2/3


(c) CodeRate=8/9

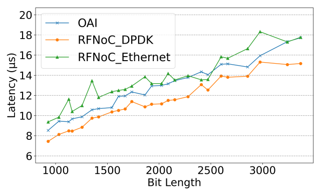
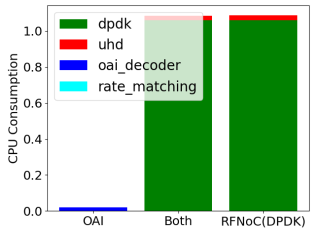

(d) CodeRate=1/5

Figure 8: LDPC Decoding Latency in FPGA and OAI

usage percentage in comparison to the use of DPDK. When considered alongside the previously observed latency metrics, it can be inferred that RFNoC, in the absence of DPDK, is a viable option for achieving low-latency LDPC decoding in contexts where data throughput requirements are moderate. However, it is acknowledged that in scenarios involving a higher number of UEs or in applications demanding high throughput, the implementation of DPDK is likely to yield enhanced performance.


(a)

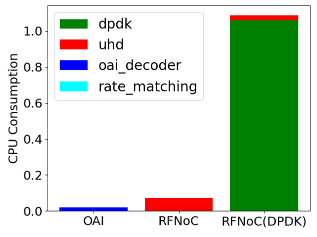

(b)

Figure 9: CPU Consumption Comparison for Different Load Allocation. (a) Comparison with OAI, RFNoC with DPDK and using both. (b) Comparison with OAI, RFNoC with or without DPDK

C. LDPC Comparison to Related Work

At present, there is a limited body of studies focusing on the hardware acceleration of LDPC decoding in RAN, and even fewer studies have conducted tests within actual RAN environments. Additionally, the diversity in computing capabilities across various platforms, coupled with differing methodologies for testing, further complicates direct comparisons. However, as illustrated in Table I, the paper endeavors to present a selection of related studies. This inclusion aims to offer a comparative perspective on performance metrics within this domain, thereby providing valuable context and insights into the current state of art researches in hardware-accelerated LDPC decoding. The configurations are 1/3 for code rate and 8448 for each code block size.

Table I: Comparison of LDPC Decoders

	Platform	***CBs***	***Latency*** (μs)
The Proposal	USRP X410	1	63.35
The Proposal	USRP X410	20	131.58
The Proposal	USRP X410	25	165.95
[10]	NVidia Titan RTX	1	87.00
[10]	NVidia Titan RTX	20	700.00
[9]	Xilinx VC707	1	61.65
[9]	Xilinx VC707	25	608.40
[12]	Xeon Gold 6154	1	31.08
[13]	i7-4770	NA	240

In the study [10], the accelerator is fully integrated with the OAI. For decoding a single CB, the GPU-based accelerator reports a decoding latency of 87μs, whereas the current work achieves a lower latency of 63.35us. For the processing of 20 CBs, the GPU's latency is recorded at 700.00μs, significantly higher than the 131.58μs achieved in this research. In [9], their framework demonstrates marginally superior latency for processing a single CB. However, the performance of our work is notably more efficient when handling 25 CBs. The research in [12] explores a software-based solution, utilizing the AVX SIMD instructions tailored to the architecture. Their approach achieves a latency of 31.08μs using a single core. Furthermore, [13] presents a software-based LDPC decoder and they report a decoding time of 240μs for an unspecified code and number of iterations.

V. Discussion

A comparative analysis of the results for LDPC and Polar code processing underscores the FPGA accelerator's varied performance for different L1 functions. Consequently, the complete offloading of all control channel processes to FPGA may not necessarily prove effective in reducing latency, especially when compared to the more efficient acceleration achieved with LDPC operations. Unique characteristics of each L1 function should guide the decision regarding FPGA offloading for optimal performance improvements.

Moreover, when comparing the acceleration results for LDPC encoding and decoding, it becomes evident that the same offloading framework yields varying results for different stages of the same algorithm. Specifically, while the decoder operation can be effectively accelerated, the encoder lags behind its software-based counterpart. This discrepancy can be attributed to the fact that the LDPC encoder inherently possesses lower algorithmic complexity when compared to the decoder. And in the OAI, extensive optimization efforts have been dedicated to the encoder, enabling it to efficiently process up to eight data blocks simultaneously. These factors collectively contribute to the observed differences in acceleration performance between the LDPC encoder and decoder. Consequently, it is our general belief that a uniform deployment of all LDPC operations in the FPGA for acceleration purposes may not always yield dependable and consistent results. Context-specific considerations, such as code rate and data block length, should be taken into account.

Based on the insights gleaned from our evaluation results and the comprehensive summary thereof, we believe that future research in the field of hardware acceleration will orient towards the following directions:

A. Application Specific

In the evolving landscape of 5G and beyond, the optimization of offloading strategies in gNB hardware acceleration, tailored to the diverse application requirements of UE, emerges as a critical area of research. A key focus is the development of adaptive offloading mechanisms that dynamically adjust to the specific needs of connected UEs. This involves creating intelligent algorithms capable of real-time analysis of UE demands, application types, and network conditions, such as traffic load and channel quality. Such strategies, aligned with Quality of Service (QoS) considerations, are essential for maintaining system efficiency and responsiveness. For instance, UEs engaged in latency-sensitive applications like VR or AR would benefit from prioritized data processing in hardware-accelerated paths, ensuring adherence to stringent QoS requirements.

Furthermore, with the advent of Ultra-Reliable Low Latency Communication (URLLC), there is a pressing need to tailor offloading strategies to meet its demanding latency and reliability criteria. This is especially pertinent in applications such as autonomous driving and industrial automation. Concurrently, the design of scalable hardware acceleration architectures in gNBs is vital. These architectures must efficiently handle varying data throughput requirements, scaling up for high-throughput applications like high-definition video streaming and scaling down for low-throughput, delay-tolerant applications. Additionally, embracing cross-layer optimization techniques offers a promising avenue for offloading in gNB hardware acceleration. By facilitating coordination between the physical, MAC, and network layers, these techniques ensure that offloading strategies not only cater to UE's application needs but also enhance the network's operational efficiency.

B. Network Slicing

The essence of gNB hardware acceleration for network slicing lies in the precision of customization and the intelligence in resource allocation, tailored to meet the specific needs of each slice. Network slices cater to diverse requirements in terms of latency, throughput, and reliability, necessitating an offloading strategy that is highly adaptable and customizable. Incorporating AI and machine learning algorithms into this process enhances the ability to dynamically allocate hardware resources. These algorithms can analyze real-time demands of each slice and adjust resources accordingly, ensuring optimal performance across various network slices.

Moreover, the aspect of energy efficiency is pivotal, especially for slices servicing IoT and other power-sensitive applications. Intelligent offloading should prioritize not just performance optimization but also efficient energy utilization, a critical factor in the sustainability of 5G networks.

In addition, real-time monitoring and adaptation are essential components of effective offloading strategies. Continuous assessment of network conditions and slice performance provides valuable insights, enabling the network to adapt swiftly to the changing landscape of network traffic and the evolving requirements of different slices.

C. AI-Driven

AI's capability for dynamic resource allocation is pivotal in the context of network slicing and application-specific demands. By utilizing real-time network data, AI algorithms can intelligently distribute hardware acceleration resources among different network slices or applications, optimizing based on unique requirements such as latency, bandwidth, and workload characteristics. This ensures that resources are efficiently used and network performance is maximized.

In addition to resource allocation, AI's predictive analytics play a crucial role in proactive network management. By analyzing historical data and current network conditions, AI can forecast future network demands and adjust resources accordingly. This foresight is essential for maintaining consistent performance and reliability, especially during peak usage times or in response to unforeseen network events.

Energy efficiency is another area where AI-driven approaches bring substantial benefits. AI can optimize the energy consumption of hardware accelerators by analyzing usage patterns and implementing energy-saving strategies, such as scaling down resources during periods of low demand. This not only reduces operational costs but also supports the sustainability of network operations.

Load balancing and throughput optimization are further enhanced by AI's ability to analyze and distribute network traffic intelligently. This ensures that high-throughput applications receive adequate resources while maintaining the required performance levels for latency-sensitive applications. The result is a well-balanced network that efficiently caters to a diverse range of application needs.

Lastly, AI significantly contributes to automated configuration and maintenance of gNB hardware accelerators. By continuously monitoring network performance and hardware health, AI can automatically adjust settings for optimal performance and preemptively identify maintenance needs. This automation reduces the need for manual intervention, lowers the risk of downtime, and ensures that the network operates at peak efficiency.

VI. Conclusion

In this paper, we examine the acceleration impact of an FPGA accelerator on various functions and stages, specifically by offloading LDPC and Polar functions from the OAI platform to an RFNoC-based FPGA. Our findings reveal that this approach can yield a substantial acceleration, almost 5 times in speed for diverse data lengths, while incurring only a 5% increase in CPU consumption. Furthermore, the study elucidates how varying data conditions contribute to disparate outcomes in terms of latency and CPU efficiency. This observation underscores the potential avenues for future research in this domain. In essence, this study not only demonstrates the effectiveness of FPGA-based acceleration in the context of OAI but also sets a foundational framework for future advancements in network optimization and efficiency. The insights garnered here are instrumental in guiding subsequent research towards developing more robust, adaptable, and high-performing 5G networks and beyond.

References

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[3] Akihiro Nakao, "Flare: Open deeply programmable network node architecture," in *Stanford Univ. Networking Seminar*, 2012.

[4] Intel, "5g ldpc intel fpga ip user guide," https://www.intel.com/content/www/us/en/docs/programmable/683107/21-1-21-1-0/about-the.html, Accessed: 2023-09-30.

[5] Marvell, "Marvell advances no-compromise 5g open ran with partners at mwc 2022," https://www.marvell.com/company/newsroom/marvell-advances-no-compromise-5g-open-ran-with-partners-at-mwc-2022.html, Accessed: 2023-09-30.

[6] Nokia, "Cloud ran: A guide to acceleration options," https://onestore.nokia.com/asset/213050?ga=2.130704904.642522200.1696335440-132478900.1696335440, Accessed: 2023-09-30.

[7] "Qualcomm x100 5g ran accelerator card," https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/Product Accessed: 2023-09-30.

[8] Elissaios Alexios Papatheofanous, Dionysios Reisis, and Konstantinos Nikitopoulos, "The ldpc challenge in software-based 5g new radio physical layer processing," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2021, pp. 312–317.

[9] Elissaios Alexios Papatheofanous, Dionysios Reisis, and Konstantinos Nikitopoulos, "Ldpc hardware acceleration in 5g open radio access network platforms," *IEEE Access*, vol. 9, pp. 152960–152971, 2021.

[10] Chance Tarver, Matthew Tonnemacher, Hao Chen, Jianzhong Zhang, and Joseph R Cavallaro, "Gpu-based, ldpc decoding for 5g and beyond," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 278–290, 2021.

[11] Tram Thi Bao Nguyen, Tuy Nguyen Tan, and Hanho Lee, "Efficient qc-ldpc encoder for 5g new radio," *Electronics*, vol. 8, no. 6, pp. 668, 2019.

[12] Yi Xu, Wenjin Wang, Zhen Xu, and Xiqi Gao, "Avx-512 based software decoding for 5g ldpc codes," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2019, pp. 54–59.

[13] Wangwang Ji, Zhiwen Wu, Kan Zheng, Long Zhao, and Yang Liu, "Design and implementation of a 5g nr system based on ldpc in open source sdr," in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–6.

[14] Cuidi Wei, Ahan Kak, Nakjung Choi, and Timothy Wood, "5gperf: Profiling open source 5g ran components under different architectural deployments," in *Proceedings of the ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases*, 2022, pp. 43–49.

[15] Lopamudra Kundu, Xingqin Lin, Elena Agostini, and Vikrama Ditya, "Hardware acceleration for open radio access networks: A contemporary overview," submitted, 2023.

[16] "Soft-decision fec integrated block logicore ip product guide (pg256)," https://docs.xilinx.com/r/en-US/pg256-sdfec-integrated-block?tocId=HkDYNxA14Aek XI7Cw61Ow, Accessed: 2023-09-30.

[17] Martin Braun, Jonathan Pendlum, and Matt Ettus, "Rfnoc: Rf network-on-chip," in *Proceedings of the GNU Radio Conference*, 2016, vol. 1.

[18] "Getting started with rfnoc in uhd 4.0," https://kb.ettus.com/Getting-Started-with-RFNoC-in-UHD-4.0, Accessed: 2023-09-30.