Experimental Evaluation of 5G vRAN Function Implementation in an Accelerated Edge Cloud

J.C. Borromeo[(1)], K. Kondepu[(2)], N. Andriolli[(3)], L. Valcarenghi[(1)]

[(1)] Scuola Superiore Sant'Anna,via G. Moruzzi 1,56124,Pisa,Italy, justinecris.borromeo@santannapisa.it
[(2)] Indian Institute of Technology Dharwad, Dharwad, Karnataka, India, 580011
[(3)] National Research Council of Italy (CNR-IEIIT), via Caruso 16, 56122, Pisa, Italy

Abstract *Disaggregated next generation eNodeB (gNB) largely benefits from offloading selected virtualized Distributed Unit (vDU) functions into a field programmable gate array (FPGA). Results show that the FPGA-based IFFT and cyclic-prefix addition processing time is faster than dual-core high-end CPU on longer OFDM symbol size with lower power consumption.*

Introduction

In an integrated fronthaul/backhaul network infrastructure, network function virtualization offers quick and affordable deployment, upgrade, and scaling of network services and functions[1]. Thus, 5G and beyond networks are expected to be based on virtualized functions as well and, recently, solutions for a cloud/virtualized Radio Access Network (vRAN) are emerging[2].

In vRAN, the next generation eNodeB (gNB) is disaggregated in Radio Unit (RU), Distributed Unit (DU), and Central Unit (CU)[3]. Generally, the RU is not virtualized because it performs radio frequency (RF) related functions, such as up/down conversion, filtering, and digital to analog conversion. The DU and the CU, which can be virtualized, are deployed close to the the antenna and in centralized locations, respectively. The RU and DU are connected by the fronthaul and the DU and the CU are connected by the midhaul[4].

Different functional splits are proposed that distribute the several functions between the DU and the CU, resulting in different delay, jitter, and capacity requirements[4]. In some cases, although optical fiber connections can support the required capacity, latency requirements are so strict (e.g., hundreds of μs for split option 7) that current virtualization techniques, such as virtual machines or containers, can hardly satisfy them[5].

Accelerated edge cloud micro data centers[6] feature the integration and interconnection of Central Processing Unit (CPU), Graphical Processing Unit (GPU), Field Programmable Gate Array (FPGA), and the recently proposed Data Processing Unit (DPU)[7]. Thus, accelerated edge cloud micro data centers will play an important role in the 5G and beyond fronthaul/backhaul network infrastructure bringing several advantages, such as latency and power consumption reduction[8]. Such advantages are beneficial not only to end-user application/verticals, but also to the integrated fronthaul/backhaul network infrastructure.

This paper implements and experimentally evaluates the performance of offloading some vDU functions onto an FPGA. Specifically, the offloaded functions are the (Inverse) Fast Fourier Transform (IFFT/FFT) and the Cyclic Prefix (CP) addition/removal in Orthogonal Frequency Division Multiplexing (OFDM) signals. FPGAs can be a very good accelerator for these functions since they offer almost deterministic latency and high processing capacity per Watt.

5G Fronthaul/Midhaul Scenario and Implementation

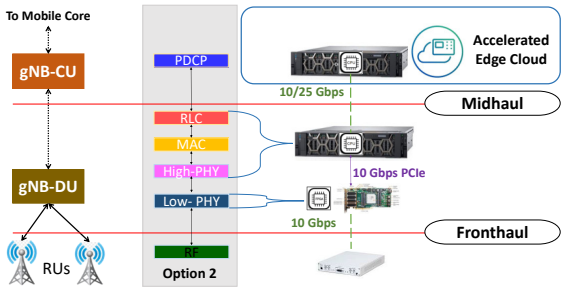

Fig. 1: 5G architecture with Option 2 split

Fig. 1 shows the considered 5G fronthaul/midhaul scenario, where gNB split option 2 is implemented. In the 5G vRAN segment, the RU with RF functions are implemented in a Universal Software Radio Peripheral (USRP). The RU is connected to the DU through a 10 Gb/s optical Ethernet fronthaul. The DU is virtualized and implements functions from Low-PHY up to Radio Link Control (RLC) of the mobile protocol stack. The

978-1-6654-3868-1/21/$31.00 ©2021 IEEE

Low-PHY layer functions, such as IFFT/FFT and CP addition/removal, are offloaded onto a programmable hardware (i.e., FPGA), as shown in Fig. 1. The High-PHY layer up to the RLC layer functions are executed in the gNB-DU CPU. The FPGA and the CPU are interconnected through the Peripheral Component Interconnect express (PCIe) interface. The Packet Data Convergence Protocol (PDCP) functions are implemented in the gNB-CU, which can be deployed in an accelerated edge cloud data center.

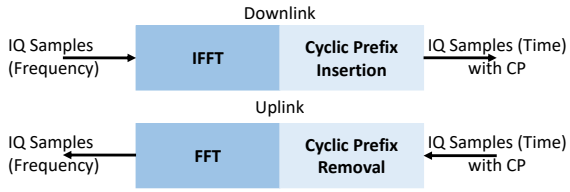

Fig. 2: Low-PHY layer implementation in FPGA

The Low-PHY layer function implementation is based on the Open Computing Language (OpenCL)[9]. The OpenCL is a parallel computing API that provides flexibility to run a single program in multiple platforms (e.g., FPGA, GPU, CPU, and DPU). Thus, it is suitable for the implementation of the considered vDU, where some functions are implemented in the FPGA and others are implemented in the CPU. Fig. 2 shows the details of the Low-PHY implementation in the FPGA. The considered FPGA hardware is a *DE10-Pro* Terasic FPGA board equipped with *Stratix 10 GX* Intel FPGA with 2 banks of DDR4 memory.

Performance Evaluation Results

In the first set of experiments the following performance parameters are considered: the IFFT and CP addition *processing time*, defined as the time required to compute IFFT and CP addition, the FPGA *logic utilization*, defined as the number of utilized logic elements, the utilized *Digital Signal Processing (DSP) blocks*, *memory bits*, *RAM blocks*, and *kernel frequency*.

Five different versions of the Low-PHY implementation are compared that exploit different optimization techniques available in the considered Intel FPGA SDK for OpenCL: (i) version (*V1*) features the implementation of IFFT and CP addition without any optimization; (ii) version (*V2*) uses the loop unrolling method (i.e., the system executes each loop in a parallel manner to speedup the computation by "N" times by using a *pragma unroll (N)* command); (iii) version (*V3*) removes function calls inside the main kernel code (i.e., it avoids creating a different circuit for a specific

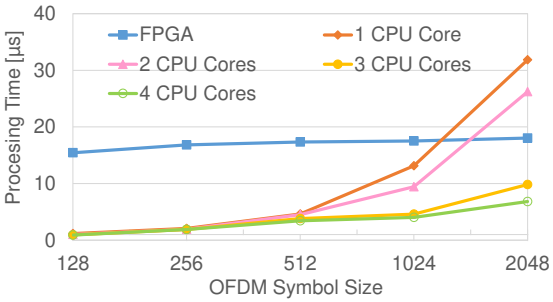

Fig. 3: FPGA vs CPU processing time

function resulting into a lower resource utilization); (iv) version (*V4*) implements a matrix instead of a vector representation of the array to increase the kernel frequency, thus reducing the computation time; (v) and version (*V5*), as V4 but with the kernel code compiled by using Intel FPGA SDK for OpenCL version 20.3 instead of the older 19.1 (used also for V1, V2, and V3).

Table 1 shows the implementation of Low-PHY function with an OFDM symbol size of 128 considering the aforementioned optimization techniques. The processing time decreases from $34.5\mu s$ to $23.5\mu s$ when the loop is fully unrolled (i.e., *V2)* with the trade-off of an increased logic utilization, DSP blocks, and RAM blocks. When function calling is avoided in the implementation (i.e., *V3*), there is a slight decrease in the logic element utilization. By exploiting a matrix representation of the array of components (i.e., *V4*) the processing time and the logic utilization are further decreased and the kernel frequency is increased by a factor of *1.7*. The most optimized version (i.e., *V5*) shows a processing time of $15.43\mu s$.

Considering the most optimized implementation (i.e., *V5*), its hardware performance as a function of the OFDM symbol size from 128 to 2048 is also evaluated and reported in Table 1. Since the design needs more FPGA resources with increasing OFDM symbols, the table shows that the logic utilization, DSP blocks, memory bits, and RAM blocks increase as well. Also the kernel frequency decreases with increasing IFFT points due to increasing IFFT complexity, which is given by $O(NlogN)$ with *N* as the number of OFDM symbols.

Fig. 3 compares the processing time as a function of the OFDM symbol size of a CPU-based and an FPGA-based Low-PHY implementation. For the CPU-based implementation, an Intel i7-7700K CPU@4.20GHz system with four cores is considered. The CPU-based implementation processing time decreases as a function of the utilized CPU cores. Moreover, the processing

Tab. 1: OpenCL Optimization Result Of Different OFDM symbol sizes.

	128								
	V1	*V2*	*V3*	*V4*	*V5*	**256**	**512**	**1024**	**2048**
Processing Time (μs)	34.37	23.5	23.45	21.4	15.43				
Logic Utilization	14%	25%	21%	19%	21%	26%	36%	51%	66%
DSP Blocks	< 1%	5%	4%	3%	3%	3%	8%	14%	14%
Memory Bits	2%	2%	2%	3%	5%	6%	11%	15%	15%
RAM Blocks	4%	6%	6%	7%	11%	13%	16%	23%	23%
Kernel Frequency(MHz)	239.23	366.7	285.63	484.26	484.78	461.68	390.93	299.67	146.26

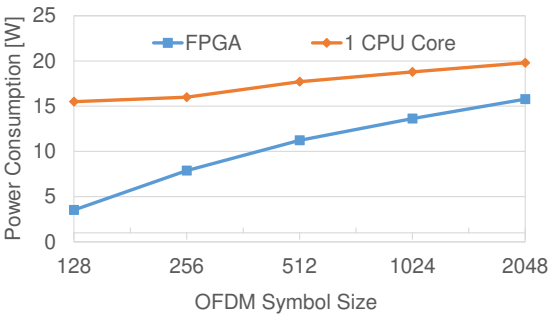

Fig. 4: FPGA vs CPU power consumption

time of both the CPU-based and the FPGA-based implementations increases as a function of the OFDM symbol size. However, the slope of the increase experienced by the FPGA-based implementation ($\approx 1\mu s$ if the OFDM symbol size doubles) is much lower than the one experienced by the single CPU core implementation. At 2048 OFDM symbol size, the FPGA-based implementation has a shorter processing time compared to single and dual core CPU.

Fig. 4 compares the performance in terms of instantaneous power consumption during the IFFT computation and CP addition. The power consumption increases as a function of the considered OFDM symbol size, but the FPGA power consumption is always lower than the single core CPU one.

Conclusions

This paper presented how the performance of virtualized Distributed Unit (vDU) Low-PHY functions can benefit from their offloading onto an FPGA. The results showed that an optimized IFFT/FFT and Cyclic-prefix addition/removal implementation not only outperforms a dual CPU implementation in terms of processing time for large OFDM symbol size, but it requires also less instantaneous power.

Acknowledgements

This work received funding from the ECSEL JU grant agreement No 876967. The JU receives support from the EU Horizon 2020 research and innovation programme and the Italian Ministry of Education, University, and Research (MIUR). We would like to thank “Intel University Program and Terasic Inc” for donating the FPGA hardware.

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges”, *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262,

[2] T. Murakami, Y. Kishi, K. Ishibashi, K. Kasai, H. Shinbo, M. Tamai, K. Tsuda, M. Nakazawa, Y. Tsukamoto, H. Yokoyama, Y. Fujii, Y. Seki, S. Nanba, T. Hara, F. Adachi, and T. Sotoyama, “Research project to realize various high-reliability communications in advanced 5g network”, in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1–8. DOI: 10.1109/WCNC45663.2020.9120477.

[3] L. M. P. Larsen, A. Checko, and H. Christiansen, “A Survey of the Functional Splits Proposed for 5G Mobile Crosshaul Networks”, in *IEEE Communications Survey and Tutorials*, 2019.

[4] ITU-T, “Transport network support of IMT-2020/5G”, GSTR-TN5G, Feb. 2018, Version 1.0.

[5] F. Giannone, K. Kondepu, H. Gupta, F. Civerchia, P. Castoldi, A. Franklin, and L. Valcarenghi, “Impact of Virtualisation Technologies on Virtualised RAN Midhaul Latency Budget: A Quantitative Experimental Evaluation”, *IEEE Communications Letters*, 2019.

[6] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, “Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers”, *Computer Networks*, vol. 130, pp. 94–120, 2018, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2017.10.002.

[7] *https://www.nvidia.com/en-us/networking/products/data-processing-unit/*, last accessed Jun. 1st, 2021.

[8] ETSI GS NVF-IFA 001, *Network Functions Virtualization; Acceleration Technologies; Report on Acceleration Technologies and Used Cases, v1.1.1*, Dec. 2015.

[9] A. Munshi, B. R. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg, *OpenCL: Programming Guide*. Addison-Wesley, July 2011.