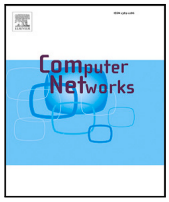
Computer Networks 210 (2022) 108931

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



FPGA-accelerated SmartNIC for supporting 5G virtualized Radio Access Network☆

Justine Cris Borromeo [a,*], Koteswararao Kondepu [b], Nicola Andriolli [c], Luca Valcarenghi [a]

[a] Scuola Superiore Sant'Anna, Via Moruzzi 1, 56124 Pisa, Italy
[b] Indian Institute of Technology Dharwad, Dharwad, Karnataka, India
[c] National Research Council of Italy (CNR-IEIIT), via Caruso 16, 56122, Pisa, Italy

ARTICLE INFO

Keywords:
Hardware Acceleration
Network function virtualization
OpenCL

ABSTRACT

Disaggregated, virtualized, and open next-generation eNodeB (gNB) could bring several benefits to the Next Generation Radio Access Network (NG-RAN) by enabling more market competition and customer choice, lower equipment costs, and improved network performance. This can be achieved through gNB-central unit (CU)-control plane (CP), gNB-CU-user plane (UP) and gNB-distributed unit (DU) separation, CU and DU function virtualization, and zero touch RAN management and control. However, to achieve the performance required by specific foreseen 5G usage scenarios (e.g., Ultra Reliable Low Latency Communications — URLLC), offloading selected disaggregated gNB functions into an accelerated hardware becomes a necessity.

To this aim, this study proposes the implementation of 5G DU Low-PHY layer functions into an FPGA-based SmartNIC exploiting the Open Computing Language (OpenCL) framework to facilitate the integration of accelerated 5G functions within the mobile protocol stack. The proposed implementation is compared against (i) a CPU-based OpenAirInterface implementation, and (ii) a GPU-based implementation of IFFT exploiting *clfft* and *cufft* libraries. Experimental results show that the different optimization techniques implemented in the proposed solution reduce the Low-PHY processing time and the use of FPGA resources. Moreover, the GPU-based implementation of the *cufft* and the proposed FPGA-based implementation have a lower processing time and power consumption compared to a CPU-based implementation for up to two cores. Finally, the implementation in a SmartNIC reduces the delay added by the host-to-device communication through the Peripheral Component Interconnect Express (PCIe) interface, considering both functional split options 2 and 7-1.

1. Introduction

5G architecture implements a very scalable and flexible network technology that provides a resilient cloud-native mobile network with end-to-end support for network slicing. It aims to support new services based on three major usage scenarios, namely: (i) enhanced mobile broadband (eMBB) supporting higher broadband access capabilities, faster connections, and higher resolution; (ii) massive machine-type communications (mMTC) for high density connections of low cost and energy efficient IoT devices; and (iii) ultra-reliable low-latency communications (URLLC), which support mission critical applications requiring very low latency and high reliability [1].

With the constant evolution of 5G networks, Network Function Virtualization (NFV) is explored to provide rapid and cost-effective deployment, upgrade, and scaling of network services and functions in an integrated fronthaul/backhaul network infrastructure [2]. NFV aims to implement the following improvements: (i) decoupling software from hardware, allowing separate timelines and maintenance for software and hardware; (ii) flexible function deployment, where software and hardware can perform different functions at various times; and (iii) dynamic scaling of the Virtualized Network Function (VNF) performance [3,4]. Virtualization prevents network service providers from investing on expensive hardware components. It can also accelerate the installation time, thereby providing faster services to customers.

☆ 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), Italy. Intel University Program and Terasic Inc are gratefully acknowledged for donating the FPGA hardware. This work is also partly supported by DST SERB Startup Research Grant (SRG-2021-001522).

* Corresponding author.
E-mail address: justinecris.borromeo@santannapisa.it (J.C. Borromeo).

https://doi.org/10.1016/j.comnet.2022.108931
Received 7 October 2021; Received in revised form 7 March 2022; Accepted 24 March 2022
Available online 31 March 2022

Table 1
Bandwidth and one-way latency requirements of different functional split options.

Functional Split	Required Downlink Capacity	Required Uplink Capacity	One-way Latency
Option 2	4016 Mb/s	3024 Mb/s	1–10 ms
Option 7-1	9.2 Gb/s	60.4 Gb/s	250 µs
Option 7-2	9.8 Gb/s	15.2 Gb/s	250 µs
Option 8	157.3 Gb/s	157.3 Gb/s	250 µs

The concept of NFV also extends to the deployment of Radio Access Networks (RAN) with the aim of a faster scaling to improve user experience as the network capacity grows, especially for IoT devices where millions of devices are expected to be connected to the 5G network. Open-source mobile platforms such as OpenAirInterface (OAI) [5] and open radio access network architectures like O-RAN [6] have been developed, where mobile networks and equipment are software-driven, virtualized, flexible, and energy-efficient. The Open RAN initiative is also part of the Telecom Infra Project (TIP) [7], which aims to accelerate innovation and commercialization in RAN domain with multi-vendor inter-operable products and solutions that are easy to integrate in the operators' network and are verified for different deployment scenarios. TIP OpenRAN program supports the development of disaggregated and interoperable 2G/3G/4G/5G NR RAN solutions based on service provider requirements.

Thus, the 5G RAN is evolving towards the Next Generation RAN — NG-RAN) where disaggregated Evolved NodeBs (gNBs) are utilized. Each gNB is composed of a Central Unit (CU) that is connected to one or more Distributed Units (DU) through the midhaul interface, and each DU is connected to one or more Radio Units (RU) that implements Radio Frequency (RF) functions using the fronthaul interface [8].

From the data center perspective, accelerated edge cloud micro data centers [9] featuring the integration and interconnection of Central Processing Units (CPUs), Graphical Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), and the recently proposed Data Processing Units (DPUs) [10] are emerging. Thus, accelerated edge cloud micro data centers will play an important role in the disaggregated and virtualized 5G RAN and beyond, bringing several benefits, such as reduced latency and power consumption [4].

This paper proposes the implementation of an FPGA-based SmartNIC to be installed in the edge cloud micro data center, near the antenna site where a DU is hosted or directly in the RU to accelerate the gNB Low-PHY functions. Specifically, the offloaded functions are the Inverse Fast Fourier Transform (IFFT) and the Cyclic Prefix (CP) addition of the Orthogonal Frequency Division Multiplexing (OFDM) in downlink transmission. The considered function implementation is based on the Open Computing Language (OpenCL) framework to ease the integration with other mobile software functions that are not accelerated (e.g., exploiting the CPU).

The FPGA-based implementation is evaluated and compared with the CPU-based Low-PHY of OpenAirInterface (OAI) [5] and GPU-based FFT/IFFT libraries (i.e., clfft [11] and cufft [12]) in terms of processing time and energy consumption. The experimental evaluation shows that for large FFT/IFFT sizes (i.e., ≥ 2048), the FPGA-based implementation outperforms the OAI Low-PHY implementation processed in a high-end single and dual core CPU and GPU-based clfft, but it shows a higher processing time compared to GPU-based cufft. However, cufft energy consumption is high, while FPGA-based Low-PHY experiences the lowest energy consumption. Moreover, the FPGA-based smartNIC overcomes the host-to-device memory transfer bottleneck, thus making FPGA-based accelerators effective in providing deterministic latency and high processing capacity per Watt.

2. Considered 5G RAN Architecture

Different functional split options are currently under investigation for the deployment of NG-RAN that distribute several functions between the RU, DU, and CU, resulting in different delay, jitter, and capacity requirements. The split options that currently received most of the attentions are Option 8, Option 7 (in particular Option 7-1, Option 7-2, and also Option 7.2x in O-RAN), and Option 2. They can be also utilized in combination when RU, DU, and CU are deployed in different devices and interconnected by fronthaul and midhaul interfaces [13–16]. Table 1 shows the requirements of the interfaces (either fronthaul or midhaul) connecting the network elements hosting the functions based on the listed split options. As shown, lower layer functional split options (i.e., option 7 and option 8) have higher fronthaul capacity and stricter one-way latency requirements due to the remote implementation of the Hybrid Automatic Repeat Request (HARQ) [17].

3GPP Release-15 [18,19] also finalized the specification of the 5G New Radio (5G NR), which supports operation with frequency bands ranging from sub-1 GHz up to mmWave. Two operating frequency ranges (FRs) have been defined: FR1: 450 MHz - 6 GHz (commonly referred to as sub-6) and FR2: 24.25 GHz–52.6 GHz (also referred to as millimeter wave). In FR1 and FR2, the maximum bandwidth is 100 MHz and 400 MHz, respectively, both being much larger than the maximum LTE bandwidth of 20 MHz. Moreover, to support a wide range of use-cases and application scenarios, 5G NR features flexible subcarrier spacing, which can be obtained by

$$\Delta f = 2^{\mu} \times 15 \text{ kHz}; \mu \in (-1, 0, 1, 2, 3, 4, 5) \quad (1)$$

Also, the slot duration is scaled by a factor of $T_{slot} = 2^{-\mu}$ from the transmission time interval of the LTE. This means that the slot duration (T_{slot}), the cyclic prefix (CP) length, and the OFDM symbol duration, ($T_{OFDM} = 1/\Delta f$) reduces as the subcarrier spacing increases, as illustrated in Fig. 1 [19]. In this case, the elaboration time needed to perform FFT/IFFT and CP addition/removal is shorter when using subcarrier frequencies higher than 15 kHz, which makes these functions one of the best candidates for hardware acceleration.

The approach proposed in this paper accelerates the 5G gNB stack Low-PHY functions in an FPGA-based SmartNIC. This work focuses on a dual-split scenario where Option 8 is implemented in the fronthaul interface, while two different functional split options are considered in the midhaul (options 2 and 7-1). Fig. 2 shows the proposed hardware offloading setup with two different scenarios distributing Packet Data Convergence Protocol (PDCP) to Low-PHY functions within the DU and CU using option 2 and 7-1 functional splits in the midhaul interface. The RU hosts RF functions only (option 8 fronthaul functional split). The DU and CU components are deployed in the edge cloud. In the accelerated edge, the Low-PHY functions in the DU are offloaded onto and accelerated by an FPGA using the OpenCL Framework.

The considered solution slightly differs from the approach currently adopted for the O-RAN fronthaul, known as option 7.2x split. In that case, as reported in [15], OFDM phase compensation, iFFT, CP addition, and digital beamforming functions in the downlink direction reside in the O-RU. The remaining PHY functions, including resource element mapping, precoding, layer mapping, modulation, scrambling, rate matching and coding reside in the O-DU. By reducing the number of functions hosted in the RU, the approach considered in this paper features most of the advantages listed in [15] for split option 7.2x, such as interoperability, advanced receivers and inter-cell coordination, and future proofness. In addition, it even features lower RU complexity, energy consumption, and cost at the expenses of transport bandwidth scalability. Indeed, it scales with the number of antennas and not with the number of streams, as in split option 7.2x. In addition, user data transfer cannot be optimized to send only Physical Resource Blocks (PRBs) that contain user data for the purpose of reducing transport bandwidth, as in split option 7.2x. However, the proposed FPGA-based implementation can be fully compatible with split option 7.2x because it can be deployed in the RU and accelerate the RU functions listed in split option 7.2x.

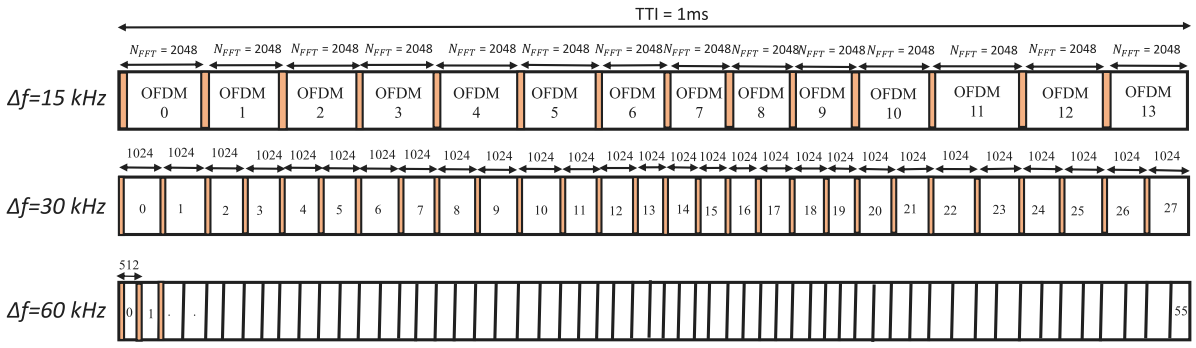


Fig. 1. 5G New Radio Flexible Numerology.

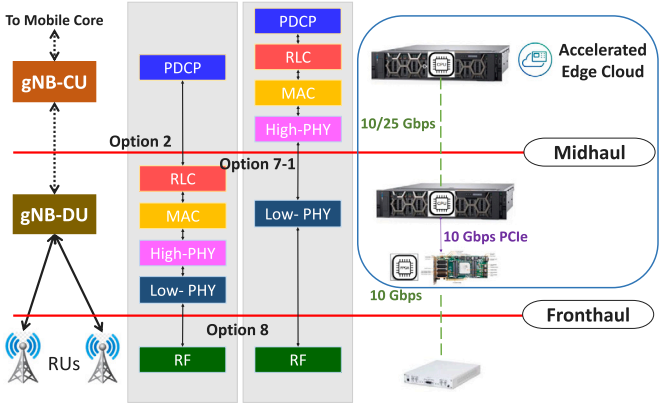


Fig. 2. 5G RAN Architecture with proposed hardware offloading setup using Options 2 and 7-1 functional split.

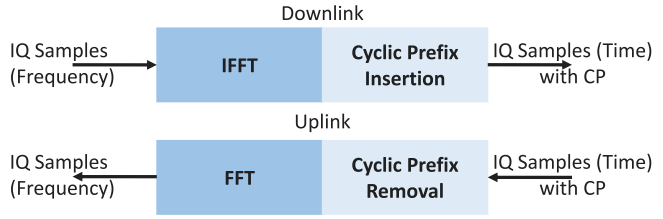


Fig. 3. Low-PHY layer protocol to be implemented in hardware.

3. Considered Low-PHY Implementation

The Low-PHY functions implemented in the FPGA are shown in Fig. 3 for both downlink and uplink directions. In the downlink direction, IQ samples in the frequency domain consisting of 32 bits (16 bits for the real part and 16 bits for the complex part) are received in the FPGA, which then performs the IFFT to convert the samples to the time domain. The cyclic prefix (CP) is then inserted as a guard interval to avoid inter-symbol interference (ISI). In the uplink direction, IQ samples in the time domain are received with CP, then the FPGA performs CP removal to take out the guard interval, and the FFT operation to convert the samples to the frequency domain. The CP insertion (removal) is performed by adding (removing) redundant bytes before each OFDM symbol. The number of added or removed CP samples is shown in Table 2.

The FFT/IFFT implementation is one of the key components, and the most complex and computationally intensive module in Orthogonal Frequency Division Multiplexing (OFDM). The FFT is an optimized computation of the Discrete Fourier Transform (DFT), which can be computed using the formula [20]:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}; k = 0, 1, \ldots, N-1 \quad (2)$$

where $X(k)$ are the samples in frequency domain, $x(n)$ are the samples in time domain, N is the number of FFT/IFFT points, and W_N^{nk} is the twiddle factor. The latter is computed as:

$$W_N^{nk} = e^{-j2nk\pi/N} \quad (3)$$

The computation of the FFT/IFFT can be divided into two parts: the data reordering and the radix butterfly configuration. The order of the samples of the FFT/IFFT input and output are different; the former is in natural order, while the latter is in bit-reversed order. A data reordering step is used to convert the input from the natural order to bit-reversed order and vice versa.

Radix butterfly configuration decomposes the computation of the FFT into different stages. The number of radix butterfly stages needed to compute the FFT/IFFT is given as $log_M(N)$, where N is the FFT/IFFT points, while M represents the radix number. The higher the radix number, the fewer the number of stages needed, balanced by a more complex twiddle factor computation.

Fig. 4 shows a 16-point FFT computed in two ways: (a) 4 stages (16-point) using Radix-2; and (b) 2 stages (16-point) using Radix-4. In computational cost multiplication, the Radix-2 brings twiddle factors at 0° and 180°, while Radix-4 has twiddle factors at angles 0°, 90°, 180°, and 270° [21]. However, as shown in Fig. 4, the former needs 4 stages, while the latter only needs 2 stages to compute the 16-point FFT. In computing the FFT with N-points (higher than 16), a combination of Radix-4 and Radix-2 stages can be considered for implementation.

This paper focuses in implementing up to 2048 FFT/IFFT points; the number of radix butterfly stages in each implementation is shown in Fig. 5. We implemented a Radix-4 butterfly configuration on the first few stages of FFT/IFFT points to achieve fewer computation stages, then we added another Radix-2 butterfly configuration on the last stage of 128, 512 and 2048-point FFT/IFFT.

Table 2
Number of Cyclic Prefix samples with respect to the FFT/IFFT size.

FFT/IFFT size	CP length
128	10
256	20
512	40
1024	80
2048	160

4. OpenCL Framework

Low-PHY functions are implemented by using the OpenCL framework [22], which can execute a kernel on an FPGA platform using

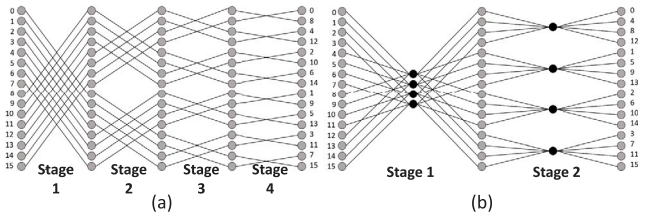


Fig. 4. Radix Butterfly Configuration: (a) 16-point Radix-2 FFT, (b) 16-point Radix-4 FFT.

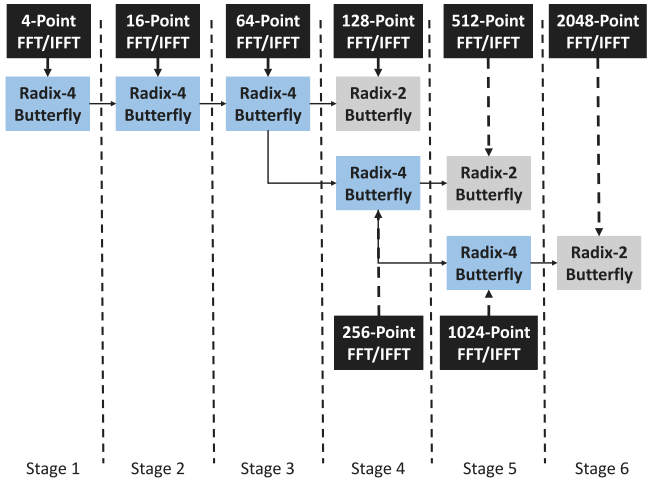


Fig. 5. Radix Butterfly Configuration of different FFT/IFFT points.

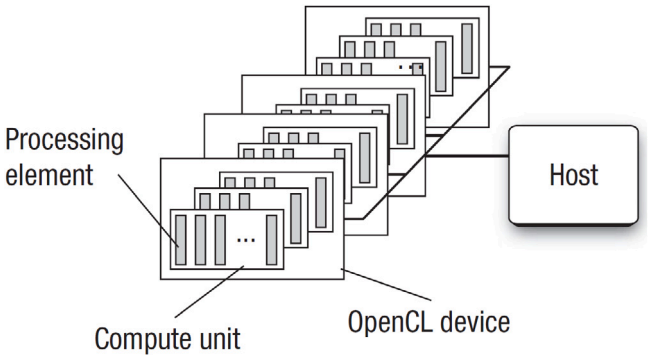


Fig. 6. OpenCL Platform model.

a software development kit (SDK) provided by Intel. This section introduces the OpenCL programming language with some references to the application of OpenCL in different scenarios (e.g., Block Ciphers and Low-PHY). The specific software development kit used to program OpenCL on Intel FPGAs is also discussed.

4.1. OpenCL Programming Language

OpenCL is a parallel computing application programming interface (API) that is capable of executing a kernel on different platforms such as CPUs, GPUs, and FPGAs [22]. It can write a single program and run it on heterogeneous platforms. However, to maximize the performance in each platform, different types of optimization techniques are implemented. It can also support a parallel computing approach to enhance application performance.

As shown in Fig. 6, the OpenCL platform always includes a single host that acts as a master capable of interacting with one or more OpenCL devices. The OpenCL device is where a stream of instructions is executed. Such instructions are called kernel. OpenCL devices are called Compute Devices (CDs) and they can be a CPU, GPU, DSP or FPGA. Thus, OpenCL is suitable for the implementation of the considered DU, where some functions are implemented in the FPGA and some others are implemented in the CPU.

4.2. OpenCL Utilization Survey

Researchers are already exploiting OpenCL framework on FPGAs for different applications, such as image processing (typically based on convolutional neural networks) and cryptographic accelerators.

FPGA-based OpenCL implementations of block ciphers to achieve high throughput and low energy consumption were investigated in [23]. Nine different ISO standard block ciphers were compared with the CPU-based implementations. Results show that the OpenCL implementation in FPGA achieves a higher throughput compared to the CPU in 8 different block ciphers aside from the Advanced Encryption Standard (AES). The authors were also able to achieve an energy improvement by $22.78x$ compared to the pure software implementation of ISO standard block ciphers.

The utilization of hardware acceleration on a virtualized Cloud-RAN is a use case reported in [4] with the aim of leveraging resource utilization for load balancing among different base stations to provide cost reduction, high resource and spectrum utilization, and energy efficient networks. One of the main issues addressed in this paper concerns the computationally intensive signal processing tasks of the physical layer (i.e., channel coding/decoding, FFT/IFFT). This research motivates our work since the authors recommended to implement these tasks in a dedicated CPU processor or on general-purpose layer 1 (L1) accelerators.

The authors from [24] assess the suitability of employing OpenCL-driven reconfigurable hardware in the context of 5G virtualized gNB DU. Using a Terasic DE5-Net Development board with Intel Stratix V GX FPGA, the authors focus on the implementation of the Low-PHY level functionalities at the DU using the Option 7-1 functional split. Results show that OpenCL has a better processing time when data sizes increase (more than 2048 OFDM symbols) due to its pipelined approach. However, the kernel implementation does not fit into the FPGA for OFDM symbol size larger than 512. Another bottleneck presented in this research is the data transfer and synchronization between the host and the device memory, as well as reading and writing from/to global and local memory inside the FPGA.

This paper focuses on a further optimized implementation compared to the one proposed in [24], aimed to deploy kernels with more than 512 OFDM kernels in FPGA through OpenCL framework. Different optimization techniques are utilized to further improve the processing time and the FPGA area overhead. This paper is also an extended version of [25], which includes the processing time of GPU-based IFFT libraries, and proposes the use of kernel autorun through OpenCL channels to reduce the data transfer between the device and the host. With the use of OpenCL channels, IQ samples can either be received from (sent through) one of the FPGA interfaces (QSFP for *DE10-pro* FPGA).

4.3. Intel FPGA SDK for OpenCL

Intel FPGA SDK for OpenCL provides the necessary APIs and runtime library to program the FPGAs attached to the PCIe interfaces, very similarly to a GPU or any kind of hardware accelerators. The IP cores needed for the communication between the FPGA, external DDR memory, and PCIe, alongside with necessary PCIe and DMA drivers for the communication between the host and the FPGA, are also provided by the board manufacturers in a form of a Board Support Package (BSP) [26].

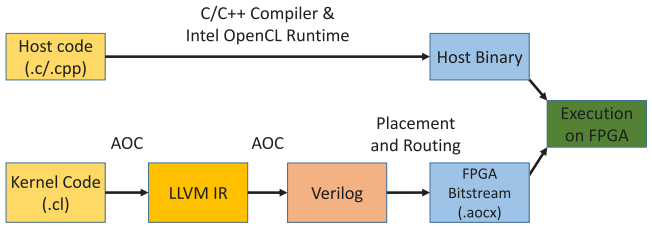


Fig. 7. Intel FPGA SDK for OpenCL flow; AOC is Intel FPGA SDK for OpenCL Offline Compiler.

The flow of Intel FPGA SDK for OpenCL to compile the host code and convert the kernel code to an FPGA bitstream is shown in Fig. 7. The OpenCL kernel needs to be compiled offline using the Intel Altera offline compiler (AOC), unlike CPUs and GPUs, due to long placement and routing time in FPGAs. AOC converts the kernel code into a Verilog code, which is the hardware description language for the FPGA. The Verilog code is converted to an FPGA bitstream after placement and routing. After compilation, the AOC generates the Altera Offline Compiler Object file (*.aoco*) containing kernel and configuration information required at runtime, Altera Offline Compiler Executable file (*.aocx*) (i.e., the hardware configuration file), and a kernel folder or subdirectory that contains the information necessary to create the *.aocx* file, including an area and timing report file for analysis. Note that LLVM is a compiler and tool chain technology that is designed for an intermediate code representation called LLVM Intermediate Representation (IR) [27].

The host side can be programmed using C or $C++$ programming language. The $C/C++$ compiler compiles the host program and links it to the Intel FPGA SDK using OpenCL runtime libraries. After compilation, the host then runs the host application, which programs and executes the hardware image into the FPGA. In our implementation, the host is programmed using $C++$ programming language.

5. Low-PHY Function Optimization using OpenCL

To optimize the OpenCL kernels for FPGA, two main strategies could be considered, namely the improvement of the pipeline throughput and the exploitation of data parallelism. Few procedures to improve the FPGA-based implementation have been discussed in [9], such as replicating a kernel pipeline to increase data parallelism and using sliding windows to improve pipeline throughput. In the following we provide a list of optimization techniques applicable in OpenCL to further reduce the processing time of the Low-PHY function and reduce the resource utilization in the FPGA.

5.1. Loop Unrolling

The loop unrolling command replicates the loop body multiple times executing each loop iteration in a parallel manner [26]. In cases where there are no loop-carried dependencies, the unrolling loops can reduce the processing time of each 'for' loop implementation in the FPGA. However, fully unrolling the loop iteration may also significantly increase the resource utilization of the FPGA. A *pragma unroll* (N) command is used by the compiler before the 'for' loop to unroll the loop, resulting in an N time speedup in the execution performance of the loop.

5.2. Avoid Function Calling

Writing a separate function and calling it inside the main kernel code is implemented as a separate circuit on the FPGA. This would result in an additional use of FPGA resources since more routing resources are needed to connect the function into the main kernel code. To further minimize the use of FPGA resources, function calls should be avoided in favor of using loops inside the kernel code, that are then partially or fully unrolled depending on the availability of the FPGA resources.

5.3. Matrix instead of Vector Representation

When performing arithmetic operations in an FPGA with large number of components in an array, especially when implementing FFT/IFFT with 2048 points, where arithmetic addition and twiddle factor multiplication are computed on 4096 components with 16-bits each. The computational complexity can be reduced by representing these components as a matrix rather than a vector, especially when implementing a nested for loop. Furthermore, lowering the computational complexity of the arithmetic operation increases the kernel operating frequency, resulting in a reduction of the processing time.

5.4. Maximizing Global Memory Bandwidth

The DE10-pro Terasic FPGA Board contains 2 banks of DDR4 memory with a bus width of 64 bits running at 2133 MHz (1066 MHz double data-rate), which provides 34.1 GB/s of external memory bandwidth. Since the memory controller on the FPGA runs at $1/8$ of the clock of the external memory (i.e., 266 MHz), just 128 bytes per clock can saturate the memory bandwidth. Increasing the memory transfer per clock above the memory bandwidth increases the area overhead without decreasing the memory transfer time. For the Low-PHY implementation in FPGA, only 128 bytes of data are sent from the global to the local memory per clock cycle to maximize the global memory bandwidth.

5.5. SmartNIC-based Implementation

The implementation of the DU with the considered split Option 2 in the downlink direction is shown in Fig. 8(a). Here, it is assumed that a Commercial Off-the-Shelf (COTS) server equipped with an FPGA is utilized. The RLC, MAC, and High-PHY functions are implemented in a CPU that receives the data from the CU (through a Network Interface Card – NIC, such as a QSFP) where the PDCP layer is implemented.

After performing the High-PHY function, the IQ samples in the frequency domain are sent from the CPU to the FPGA through the CPU's PCIe interface for Low-PHY elaboration in the FPGA. Finally, IQ samples in the time domain are returned to the CPU, again through the PCIe, and sent to the RU through another NIC (e.g., SFP+). However, the data transfer and synchronization between the host and the device memory becomes a bottleneck in this implementation scenario, due to the contribution of the host-to-device transfer latency. Aside from the host-to-device memory transfer, data transfer between FPGAs' global and local memory also adds to the FPGA-based Low-PHY processing time. Compared to GPUs, where the memory bandwidth offered by GDDR5X or HBM2 is in the order of hundreds of GB/s, FPGA boards usually offer a much lower memory bandwidth (e.g., DDR4 with around 32 GB/s). The proposed solution is shown in Fig. 8(b). This implementation reduces the Low-PHY processing time since data are transferred from the CPU to the FPGA through the PCIe interface only once. This is possible by exploiting auto-run kernels and OpenCL channels. The auto-run kernels allow to execute the processing in hardware without interaction with the host and the global memory. Indeed, the host starts the auto-run kernel that forwards the data to the NIC interface (e.g., QSFP) of the FPGA after the Low-PHY implementation by means of the I/O OpenCL channels.

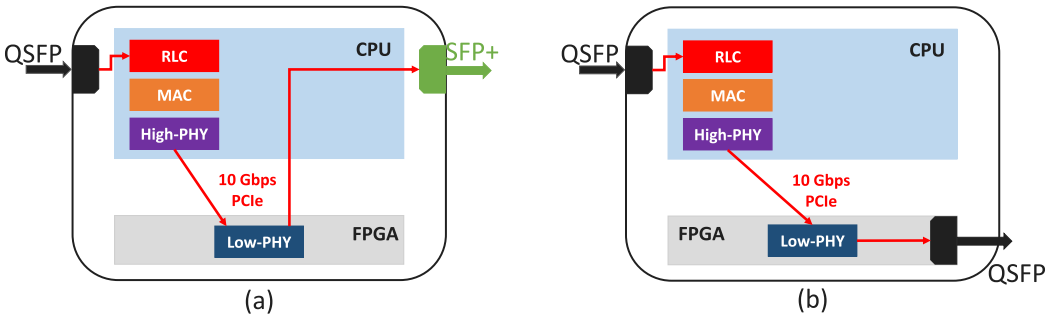


Fig. 8. DU implementation scenarios: (a) COTS server equipped with FPGA; (b) FPGA-based SmartNIC implementation.

6. Performance Evaluation

This section evaluates the impact of the different optimization techniques on the processing time (i.e., the time required to process the considered FFT size), the resource utilization (logic gates, DSP, memory bits, RAM), and the maximum kernel operating frequency of a Low-PHY function in FPGA using OpenCL framework. The best performing version is then assessed in terms of resource utilization and maximum kernel operating frequency by varying the number of IFFT points from 128 to 2048. Results of the FPGA-based Low-PHY in terms of processing time and energy consumption are then compared to a CPU-based Low-PHY implementation with OpenAirInterface, and a GPU-based implementation of *clfft* and *cufft* libraries. The clfft is an OpenCL-based software library containing FFT functions [11]. The cufft is a CUDA-based FFT library developed by NVIDIA [12].

As shown in Fig. 2, the RU hosts just the RF functions implemented in an Ettus X310 Universal Software Radio Peripherals (USRPs) and is connected to the DU through a 10 Gbps optical Ethernet fronthaul. The DU and CU components are deployed in the edge cloud exploiting Dell Poweredge R740 servers. A midhaul link with 10/25 Gbps is used to connect the DU and the CU. In the accelerated edge, the DU Low-PHY functions are offloaded onto a *DE10-pro* development board with Stratix 10 FPGA, two 8 GB DDR4 memory modules and PCIe v3.0 with 16 slots at 32 GB/s bandwidth [26]. The CPU-based implementation is executed in an Intel Core i7-7700K@4.2 GHz and based on Intel Advanced Vector Extension 2 (AVX2), where arithmetic operations are performed on 256-bit vectors to achieve better performance with floating point calculations and data organization. The *clfft* and *cufft* are implemented in an NVIDIA Tesla T4 GPU featuring 320 NVIDIA Turing tensor cores, 16 GB GDDR6 memory modules, and PCIe v3.0 with 16 slots [28]. Host-to-device transfer latency results are also detailed.

6.1. Low-PHY Layer Optimization

Table 3 shows the performance of five implementations of the Low-PHY function of the 5G RAN with 128 IFFT points, considering the different optimization techniques discussed in Section 3.4. The five implementations exploit different optimization techniques available in the considered Intel FPGA SDK for OpenCL: (i) *version 1* features the implementation of IFFT and CP addition without any optimization; (ii) *version 2* uses the loop unrolling method; (iii) *version 3* removes function calls inside the main kernel code; (iv) *version 4* implements a matrix instead of a vector representation of the array to increase the kernel frequency, thus reducing the computation time; (v) and *version 5* is the same as version 4, but with the kernel code compiled by using Intel FPGA SDK for OpenCL version 20.3, instead of the older 19.1 used in the previous cases. The different versions are compared in terms of processing time, utilization of logic gates, DSP utilization, memory utilization, RAM utilization, and kernel operating frequency.

As shown in Table 3, the processing time decreases from 34.37 μs to 23.5 μs when the loop is fully unrolled with the trade-off of an increased utilization of logic gates (14% to 25%), DSP utilization (<1% to 5%), and RAM utilization (4% to 6%). When function calling is avoided in the implementation, there is a decrease of about 4% in the utilization of logic elements. Using matrix representation on array of components further decreases the processing time and the used logic gates, and increases the kernel frequency by a factor of 1.7. The fastest processing time, 15.43 μs, is achieved in implementation version 5 with a utilization of 21% for logic gates, 3% for DSP, 5% for memory, 11% for RAM, and the kernel operates at 484.78 MHz frequency, achieved with Intel FPGA SDK for OpenCL version 20.3.

Table 3
OpenCL optimization result on 128 OFDM symbols.

	Version 1	*Version 2*	*Version 3*	*Version 4*	*Version 5*
Processing time [μs]	34.37	23.5	23.45	21.4	15.43
Logic gate utilization	14%	25%	21%	19%	21%
DSP utilization	<1%	5%	4%	3%	3%
Memory utilization	2%	2%	2%	3%	5%
RAM utilization	4%	6%	6%	7%	11%
Kernel frequency [MHz]	239.23	366.7	285.63	484.26	484.78

Table 4
FPGA resources and kernel operating frequency of Low-PHY layer functions with different IFFT points.

	128	*256*	*512*	*1024*	*2048*
Logic gate utilization	21%	26%	36%	51%	66%
DSP utilization	3%	3%	8%	14%	14%
Memory utilization	5%	6%	11%	15%	15%
RAM utilization	11%	13%	16%	23%	23%
Kernel frequency [MHz]	484.78	461.68	390.93	299.67	146.26

6.2. Hardware Performance

Considering the best performing version in Table 3, namely version 5, 5G Low-PHY functions with 128 up to 2048 IFFT points have been implemented in FPGA using OpenCL framework. Results in terms of utilization of logic gates, DSP utilization, memory utilization and RAM utilization, and kernel operating frequency are shown in Table 4.

It can be noted that the use of hardware resources increases as the IFFT points increase. This is because the size of the array increases and more FPGA resources are needed to parallelize the IFFT and CP addition computation. Also the kernel frequency decreases with increasing IFFT points due to the increasing complexity, given by $O(N log N)$ with N being the number of IFFT points.

6.3. Processing Time

Fig. 9 shows the processing time as a function of the IFFT size of 5G low-PHY in OpenCL exploiting an FPGA, a CPU with a different number of processing cores (from one to four), and in GPU with clfft and cufft libraries. The open-source benchmark package Gearshifft [29] is used to evaluate the processing performance of IFFT libraries in the GPU.

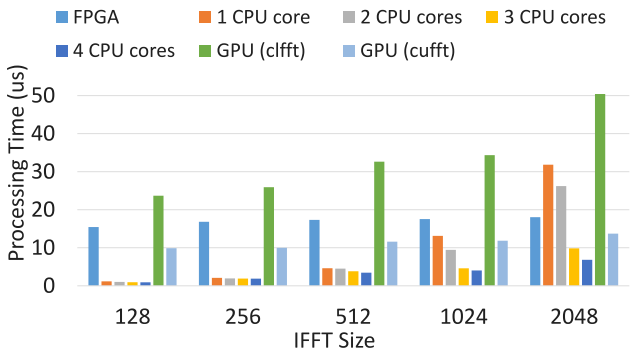


Fig. 9. FPGA vs. CPU vs. GPU processing time.

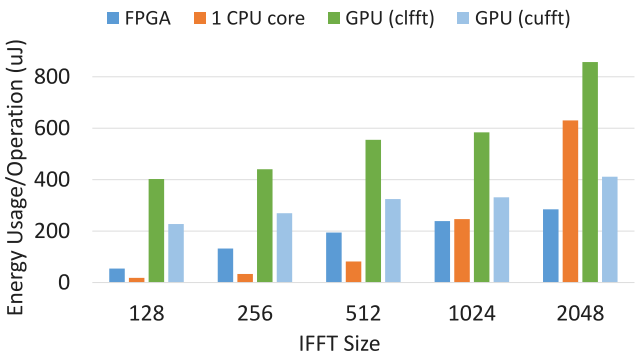


Fig. 10. Energy Usage per Low-PHY operation in FPGA, CPU, and GPU with different IFFT points.

The CPU-based processing time decreases for increasing CPU cores. This is expected since multiple cores allow to run multiple processes at the same time with greater ease compared to a single core, increasing the performance when handling multiple tasks or demanding computations. Such processing time performance cannot be appreciated for lower IFFT sizes (i.e., 128 and 256 IFFT points) due to the lower complexity, which can be handled by a single CPU core. However, a notable performance difference can be highlighted for 2048 IFFT points, where it takes around 6.38 μs only, to run the implementation with 4 CPU cores, while it needs 31.84 μs with 1 CPU core. This is because the computational effort for IFFT and CP addition is shared among 4 CPU cores, resulting in an approximately four times lower processing time.

Moreover, the processing time for the CPU-, GPU-, and FPGA-based implementations increases as a function of the IFFT points. However, just a small increase in the processing time (around 1 μs) occurs with the FPGA-based implementation because of data parallelism with full loop unrolling on the radix butterfly and CP addition computation. The same happens with GPU-based implementation since it has more parallel computing resources (320 Turing Tensor Cores and 2560 CUDA Cores for NVIDIA Tesla T4) compared to a CPU. Also, GPU-based cufft has a faster processing time compared to clfft, since cufft is based in CUDA developed also by NVIDIA where the library is executed, better matching the computing characteristics of the GPU and thus offering a better performance.

In comparing the processing time of different implementations, results show that the GPU-based implementation of the clfft library has the longest processing time, making it less suited to be deployed in the 5G Low-PHY. CPU-based implementations (from 1 to 4 CPU cores) have the shortest processing times up to 512 IFFT size. However, at 2048 IFFT points, the FPGA-based and GPU-based cufft implementations are $1.45x$ and $1.91x$ faster compared to the CPU-based implementation with up to 2 CPU cores, respectively. The CPU-based Low-PHY implementation performs better at smaller IFFT sizes since it operates at a higher clock frequency (i.e., 4.2 GHz) compared to GPU (i.e., 1.59 GHz) and FPGA (i.e., 480 MHz). However, when the computational complexity increases (i.e., P>1024 IFFT points), the data parallelism of FPGA and the parallel computing resources of GPU outperforms CPU-based implementations.

Although the CPU-based implementation with 3 or 4 cores has a shorter processing time compared to FPGA-based Low-PHY and cufft library implementation in Tesla T4 GPU, these results are only possible if there are no other 5G functions implemented in the CPU. With a 5G RAN testbed using a higher layer split (option 2), the DU does not only implement Low-PHY, but also processes the High-PHY, MAC, and RLC functions in the CPU. In this case, CPU resources are shared among four different 5G functions, resulting in a longer processing time. This is why implementing 5G LOW-PHY function in FPGA or offloading the IFFT implementation in GPU using cufft library can improve the overall gNB performance, since they can free some of the CPU cores (e.g., 2) and the free CPU cores can be exploited to perform the remaining RAN functions.

6.4. Energy Consumption

Fig. 10 shows the energy consumption per Low-PHY operation in FPGA, single CPU core, and GPU (for both clfft and cufft). It is measured as the energy consumed per Low-PHY operation, which is obtained by multiplying the power consumption by the Low-PHY processing time in each device. The energy consumption allows a fair comparison among the different implementations because it is the product of power consumption and processing time. Thus, low energy consumption can be achieved by low power consumption or short processing time. The s-tui [30] tool is used to measure the CPU power, while *quartus_pow* (included in the *de10_pro* board support package) is used to estimate the power dissipated in the FPGA, and the $nvidia-smi$ command is used to measure the power consumption in the GPU. As shown in Fig. 10, the energy usage per operation increases as a function of the considered IFFT points when using either FPGA, CPU, or GPU. Results also show that the GPU-based clfft consumes the highest amount of energy per operation, also due to the large processing time, as shown in Fig. 9, making it the least optimal solution to be integrated into the 5G Low-PHY. On the other hand, CPU-based implementation of Low-PHY has the lowest energy consumption up to 512 IFFT size because of the short processing time. However, at higher IFFT sizes, the FPGA-based implementation has the lowest energy consumption ($1.03x$ lower than single-core CPU for 1024 IFFT size and $2.22x$ lower than single-core CPU for 2048 IFFT size) followed by GPU-based cufft implementation. In fact, the long processing time of the FPGA-based implementation (see Fig. 9) is compensated by a low power consumption (see Fig. 11).

Also, implementing Low-PHY with more CPU cores further increases the energy consumption. This means that, as the IFFT size increases, offloading the Low-PHY function into the FPGA becomes more energy efficient compared to processing them into the CPU or GPU (for both clfft and cufft libraries).

The proposed solution is flexible. If the available fronthaul/midhaul latency budget is large, an increased processing time can be traded for a lower power consumption, provided that the fronthaul/midhaul latency constraint is satisfied. Otherwise, the hardware guaranteeing the lowest processing time shall be utilized.

6.5. FPGA-based SmartNIC Scenario

Fig. 12 shows the overall execution time of the Low-PHY implementation when a COTS server is equipped with an FPGA, i.e., the scenario depicted in Fig. 8(a). In this implementation, although the FPGA-based implementation of the 5G Low-PHY layer provides a faster processing for larger IFFT sizes with lower energy consumption, the host-to-device/device-to-host and off-chip to on-chip/on-chip to off-chip memory transfer time is still a bottleneck. Indeed, writing to and reading from memory (i.e., Write and Read in the figure) requires more time than the kernel execution itself (i.e., Kernel in the figure), especially for the 2048 IFFT size. However, the proposed smartNIC

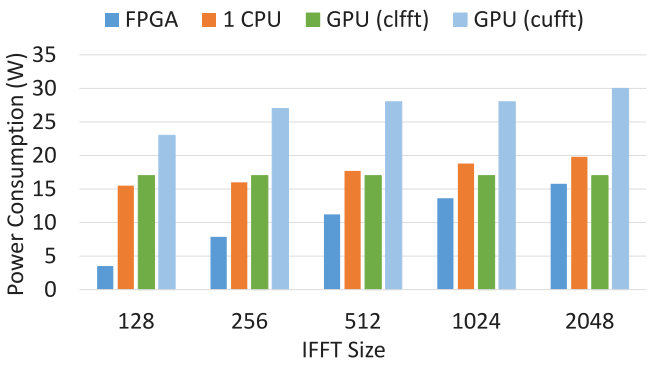


Fig. 11. Power Consumption of Low-PHY operation in FPGA, CPU, and GPU with different IFFT Size.

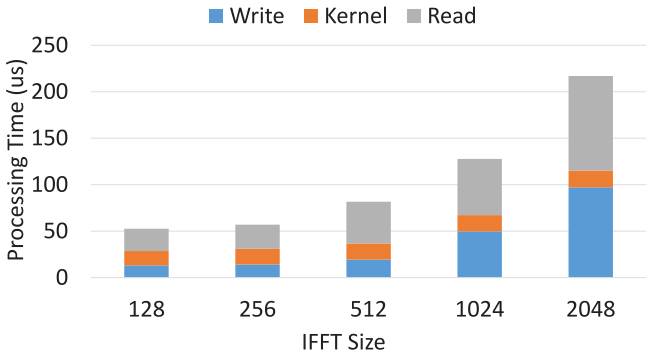


Fig. 12. FPGA overall execution time.

implementation, depicted in Fig. 8(b), mitigates this issue because data are directly sent to the RU through the QSFP interface after the IFFT and CP addition execution, therefore just the data writing to the kernel (i.e., Write) and the kernel processing time (i.e., Kernel) contribute to the overall execution time for option 2 functional split. Thus, the SmartNIC implementation reduces the processing time of about 54.33% for the 128 IFFT size and of 53.08% for the 2048 IFFT size with respect to the one achieved by the COTS server implementation.

Moreover, in case split Option 7-1 is implemented, where only Low-PHY functions are implemented in the DU, while the upper layer functions are implemented in the CU, the computation time can be further reduced. Indeed, in this scenario IQ frequency domain samples coming from the CU can be input directly to the FPGA-based smartNIC, processed there, and the resulting IQ time domain samples can be sent to the RU through another smartNIC QSFP output. Thus, in this scenario also the time required to write the data to the kernel (i.e., Write in Fig. 9) can be saved.

Another bottleneck of implementing OpenCL in FPGA is the writing to and reading from the global memory, which is the DDR4 RAM of the FPGA development board. Since data are sent to the global memory from the host through the PCIe interface, they have to be forwarded first to the local memory inside the FPGA for the IFFT and CP Addition processing. To further reduce the overall processing time of the FPGA-based Low-PHY, OpenCL host pipes [31] can be utilized to have a direct communication between the host and the kernel running in the FPGA. This solution bypasses the latency contributed by the global to local memory transfer inside the FPGA. However, this implementation is left as a future work since host pipes are supported by Arria 10 GX FPGAs only.

7. Conclusion

This paper proposed the implementation of the Low-PHY functions of a disaggregated gNB distributed unit (DU) in an FPGA-accelerated SmartNIC. The proposed Low-PHY implementation has been compared against the CPU-based implementation of Low-PHY utilized by OpenAirInterface running in a CPU with 1 to 4 cores and with clfft and cufft libraries running in a GPU.

Results showed that for low IFFT size the FPGA-based and GPU-based cufft implementations experience a higher processing time compared to the CPU-based implementation. However, at 2048 IFFT points, the FPGA-based Low-PHY function and GPU-based cufft can free up to 2 CPU cores thanks to a $1.45x$ and a $1.91x$ processing time reduction, respectively. The GPU-based implementation showed a higher energy consumption than the FPGA-based one. The FPGA-based implementation also showed the lowest energy usage per operation. Finally, the utilization of the FPGA-based SmartNIC avoided the latency contributed by the host-to-device/device-to-host and off-chip to on-chip/on-chip to off-chip memory transfer.

Although OpenCL provides an easy integration of the Low-PHY with other 5G functions implemented in the CPU, there are still some improvements to be addressed in future works. One is the utilization of host pipes for direct communication between the host and kernel running in the FPGA. The implementation of Low-PHY into the RU using a lower layer split (7.2x used by O-RAN) will also be considered in an SoC FPGA to further reduce the processing time, since the CPU and the FPGA share the same memory. Finally, the FPGA-based Low-PHY will be integrated with the other 5G protocol stack implemented by OpenAirInterface to analyze the overall gNB performance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Ghosh, A. Maeder, M. Baker, D. Chandramouli, 5G evolution: A view on 5G cellular technology beyond 3GPP release 15, IEEE Access 7 (2019) 127639–127651.
[2] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, IEEE Commun. Surv. Tutor. 18 (1) (2016) 236–262.
[3] Technical Specification Group Radio Access Network, Study on New Radio Access Technology; Radio Access Architecture and Interfaces, Technical Report (TR) 38.801, 3GPP, 2017, Version 2.0.0.
[4] Network Functions Virtualization; Acceleration Technologies; Report on Acceleration Technologies & Use Cases, v1.1.1, ETSI GS NVF-IFA 001.
[5] OpenAirInterface | 5G software alliance for democratising wireless innovation, 2021, https://openairinterface.org/ (Last accessed: 2021-07-19).
[6] O-RAN alliance, 2021, https://www.o-ran.org/ (Last accessed: 2021-07-21).
[7] A new approach to building and deploying telecom network infrastructure, 2021, https://telecominfraproject.com/?section=access (Last accessed: 2021-20-04).
[8] F. Civerchia, K. Kondepu, F. Giannone, S. Doddikrinda, P. Castoldi, L. Valcarenghi, Encapsulation techniques and traffic characterisation of an ethernet-based 5G fronthaul, in: 20th International Conference on Transparent Optical Networks (ICTON), 2018, http://dx.doi.org/10.1109/ICTON.2018.8473737.
[9] J.C. Borromeo, K. Kondepu, N. Andriolli, L. Valcarenghi, An overview of hardware acceleration techniques for 5G functions, in: 22nd International Conference on Transparent Optical Networks (ICTON), 2020, http://dx.doi.org/10.1109/ICTON51198.2020.9203242.
[10] NVIDIA DPUS, 2021, https://www.nvidia.com/en-us/networking/products/data-processing-unit/ (Last accessed: 2021-07-19).
[11] Clfft, 2021, https://github.com/clMathLibraries/clFFT (Last accessed: 2021-07-19).
[12] Cufft, 2021, https://docs.nvidia.com/cuda/cufft/index.html (Last accessed: 2021-07-19).
[13] L.M.P. Larsen, A. Checko, H.L. Christiansen, A survey of the functional splits proposed for 5G mobile crosshaul networks, IEEE Commun. Surv. Tutor. 21 (1) (2019) 146–172.
[14] ITU-T, 5G wireless fronthauls requirements in a passive optical network context, 2020, ITU-T G.Sup66 (09/2020).
[15] O-RAN Fronthaul Working Group, Control, User and Synchronization Plane Specification, O-RAN.WG4.CUS.0-v07.00 Version 1.0, 2021.
[16] Huber suhner functional split, 2022, https://www.hubersuhner.com/en/documents-repository/technologies/pdf/fiber-optics-documents/5g-fundamentals-functional-split-overview (Last accessed: 2022-01-10).

[17] Common Public Radio Interface: eCPRI Interface Specification, Version 2.0, 2019.
[18] 3GPP, 3GPP TR 38.912 v15.0.0 (2018-06): Study on new radio (NR) access technology (release 15), (38.912) 3GPP, 2018, Version 15.0.0.
[19] J.K. Chaudhary, A. Kumar, J. Bartelt, G. Fettweis, C-RAN employing xRAN functional split: Complexity analysis for 5G nr remote radio unit, in: 2019 European Conference on Networks and Communications (EuCNC), 2019, pp. 580–585, http://dx.doi.org/10.1109/EuCNC.2019.8801953.
[20] J.W. Cooley, J.W. Tukey, An Algorithm for the Machine Calculation of Complex Fourier Series, in: Mathematic of Computation, 1965, pp. 297–301.
[21] G. Polat, S. Ozturk, M. Yakut, Design and implementation of 256-point radix-4 100 gbit/s fft algorithm into FPGA for high-speed applications, ETRI J. 37 (4) (2015) 667–676.
[22] A. Munshi, B.R. Gaster, T.G. Mattson, J. Fung, D. Ginsburg, OpenCL: Programming Guide, Addison-Wesley, 2011.
[23] A. Barenghi, M. Madaschi, N. Mainardi, G. Pelosi, Opencl HLS based design of FPGA accelerators for cryptographic primitives, in: Intern. Conf.E on High Performance Computing Simulation (HPCS), 2018, pp. 634–641, http://dx.doi.org/10.1109/HPCS.2018.00105.
[24] F. Civerchia, M. Pelcat, L. Maggiani, K. Kondepu, P. Castoldi, L. Valcarenghi, Is opencl driven reconfigurable hardware suitable for virtualising 5g infrastructure? IEEE Trans. Netw. Serv. Manage. 17 (2) (2020) 849–863.
[25] J.C. Borromeo, K. Kondepu, N. Andriolli, L. Valcarenghi, Experimental evaluation of 5G vRAN function implementation in an accelerated edge cloud, in: 2021 European Conference on Optical Communication (ECOC), 2021.
[26] Intel Corporation, Intel FPGA SDK from opencl pro edition: Programming guide, 2021, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf (last accessed 2021-07-21).
[27] The LLVM compiler infrastructure project, 2022, https://llvm.org/ (Accessed: March 7, 2022).
[28] NVIDIA Tesla T4, 2021, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-product-brief.pdf (Last accessed: 2021-07-21).
[29] P. Steinbach, M. Werner, Gearshifft - the FFT benchmark suite for heterogeneous platforms, 2017, CoRR abs/1702.00629 arXiv:1702.00629 URL http://arxiv.org/abs/1702.00629.
[30] The stress terminal UI: s-tui, 2021, https://github.com/amanusk/s-tui (Last accessed: 2021-07-21).
[31] Intel, Opencl host pipe design example, 2021, https://www.intel.com/content/www/us/en/programmable/support/support-resources/design-examples/design-software/opencl/host-pipe.html (Last accessed: 2021-07-21).

Justine Cris Borromeo received his BS Electronics Engineering degree at Mindanao State University- Iligan Institute of Technology in 2015, and the MS Electronics Engineering at Ateneo de Manila University in 2019. He is currently a Ph.D student in Emerging Digital Technologies at Scuola Superiore Sant'Anna, Pisa. His research interests includes radio access networks in 5G technologies, and FPGA and GPU-based hardware acceleration.

Koteswararao Kondepu is an Assistant Professor at India Institute of Technology Dharwad, Dharwad, India. He obtained his Ph.D. degree in Computer Science and Engineering from Institute for Advanced Studies Lucca (IMT), Italy in July 2012. His research interests are 5G, optical networks design, energy-efficient schemes in communication networks, and sparse sensor networks.

Nicola Andriolli received the Laurea degree in telecommunications engineering from the University of Pisa in 2002, and the Diploma and Ph.D. degrees from Scuola Superiore Sant'Anna, Pisa, in 2003 and 2006, respectively. He was a Visiting Student at DTU, Copenhagen, Denmark and a Guest Researcher at NICT, Tokyo, Japan. In 2007-2019 he was an Assistant Professor at Scuola Superiore Sant'Anna. Since 2019 he is a Researcher at CNR-IEIIT.

He has a background in the design and the performance analysis of optical circuit-switched and packet-switched networks and nodes. His research interests have extended to photonic integration technologies for telecom, datacom and computing applications, working in the field of optical processing, optical interconnection network architectures and scheduling. Recently he has been investigating integrated transceivers, frequency comb generators, and architectures and subsystems for photonic neural networks. He authored more than 180 publications in international journals and conferences, contributed to one IETF RFC, and filed 11 patents.

Luca Valcarenghi is an Associate Professor at the Scuola Superiore Sant'Anna of Pisa, Italy, since 2014. He published almost three hundred papers (source Google Scholar, May 2020) in International Journals and Conference Proceedings. Dr. Valcarenghi received a Fulbright Research Scholar Fellowship in 2009 and a JSPS Invitation Fellowship Program for Research in Japan (Long Term) in 2013. His main research interests are optical networks design, analysis, and optimization; communication networks reliability; energy efficiency in communications networks; optical access networks; zero touch network and service management; experiential networked intelligence; 5G technologies and beyond.