Civerchia, F., Pelcat, M., Maggiani, L., Kondepu, K., Castoldi, P., Valcarenghi, L. (2020). Is OpenCL Driven Reconfigurable Hardware Suitable for Virtualising 5G Infrastructure?. *IEEE Transactions on Network and Service Management*, 17(2), pp. 849 - 863.

DOI: https://doi.org/10.1109/TNSM.2020.2964392

# Is OpenCL Driven Reconfigurable Hardware Suitable for Virtualising 5G Infrastructure?

F. Civerchia, M. Pelcat, L. Maggiani, K. Kondepu, P. Castoldi, L. Valcarenghi

***Abstract*—The Open Computing Language (OpenCL) is increasingly adopted for programming processors with reconfigurable hardware acceleration. The 5G telecommunication infrastructure, imposing strong latency constraints on the managed communications, may benefit from OpenCL-designed accelerated processing. This paper presents the first study to evaluate OpenCL hardware acceleration in the context of a 5G base station physical layer. The implementation and optimization process to accelerate the Orthogonal Frequency Division Multiplexing (OFDM) part of the 5G downlink is conducted on a high-end Field Programmable Gate Array (FPGA). We show that the proposed OpenCL implementation complies with the 5G processing timing requirements since the computation time is consistent with the present 5G deployment. However, to be suitable for 5G, the OpenCL platform must improve the data latency transfer between hardware and software. Moreover, a further enhancement for the OpenCL implementation is to improve the code by means of OpenCL optimization techniques. In this way, the performance can be further improved with respect to optimized software on vectorized high-end processors.**

***Index Terms*—5G, Hardware Acceleration, Reconfigurable Computing, OpenCL**

## I. Introduction

In general, processing can be performed by exploiting two paradigms: hardware and software. While the boundary between these paradigms is currently being blurred by methods such as High-Level Synthesis (HLS) and High-Level Design (HLD) [1], hardware is still today a costly solution in terms of design and optimization efforts, but can lead to orders of magnitude higher performance than software thanks to specialization. When inflexible Application Specific Integrated Circuit (ASIC) implementations are hindered by upgradability requirements, reconfigurable hardware is an option of choice for accelerating processing, especially when Application Programming Interfaces (APIs), such as OpenCL, simplify hardware/software communication and co-processing.

High-level software languages with extensive open libraries (e.g., Java, .NET frameworks) [2] [3] have made software design flexible and scalable by abstracting most of hardware resources. However, the utilized level of abstraction and compilation sub-optimality might negatively impact the achievable performance.

F. Civerchia, K. Kondepu, P. Castoldi, L. Valcarenghi, are with Scuola Superiore Sant'Anna, Pisa, Italy

M. Pelcat is with IETR, UMR CNRS 6164, Institut Pascal, UMR CNRS 6602, Univ Rennes, INSA Rennes, France

L. Maggiani is with SmaRTy sas, Aubière, France

Manuscript received January 3, 2020; revised ...

In this scenario, FPGAs are increasingly being used because they offer a third approach. Like ASICs, FPGAs are hardware circuits dedicated to a specific task but they can be reprogrammed a large number of times and for several applications [4]. Moreover, a FPGA distributes computing spatially, potentially exploiting large application parallelism, while software allocates computing sequentially.

However, whilst FPGAs combine the benefits of hardware and software, creating applications optimized for FPGAs is still difficult and usually requires to learn a Hardware Description Language (HDL) such as Verilog or VHDL.

OpenCL represents a more versatile solution, as this language can be compiled for several architectures, and not only for FPGAs, provided that a Host+Accelerator structure is used. A single OpenCL program is thus portable to both parallel and sequential computing devices. However, optimizations are required to obtain good performance on the target platforms [5]. Moreover, an additional advantage of OpenCL is its dynamic and seamless reconfigurability. That is, a computer program written in OpenCL can seamlessly run in hardware or software based on the hardware resource status.

The 5G telecommunication standard is expected to dramatically improve performance with respect to its predecessors by providing both very high bandwidth and low latency. A strong challenge is to comply with latency constraints while supporting large data rates. Hardware acceleration may help to reduce processing latency. For these reasons, OpenCL is a good candidate for accelerating 5G tasks. Moreover, the decision of offloading virtualized function computation to hardware might also depend on compute resources status and availability. For example, offloading computation to hardware can free time slots on compute resource that can then be exploited for improving the performance of already deployed 5G services or deploying additional ones.

The goal of this paper is to assess the suitability of the utilization of OpenCL driven reconfigurable hardware in the context of 5G virtualized base stations, specifically virtualized next generation eNB (gNB) Distributed Unit (DU). To the best of our knowledge, it represents the first study considering the implementation of the 5G DU offloaded to the FPGA by means of OpenCL approach. In particular, the implementation is based on the OFDM acceleration and several performance parameters characterizing the considered implementation are taken into account. One of them is the hardware resource usage considered to offload 5G downlink processing (i.e., the data stream from base station to users). Using a high-end FPGA and a state-of-the-art toolchain for OpenCL, we study how many hardware resources are needed and how many 5G base

station functions can be hosted into the FPGA. Moreover, a resource comparison between OpenCL and HDL is reported to understand the differences between these two approaches.
The work also addresses 5G processing performance in terms of execution time, that results in 5G function processing latency. A comparison between purely software approach (i.e., Single Instruction Multiple Data (SIMD)), purely hardware approach (i.e., HDL), and OpenCL is reported to better understand the OpenCL performance trend. Moreover, a model for both OpenCL and SIMD approach is evaluated to predict the processing delay for bigger OFDM symbol size, to understand the best implementations (hardware and software) in case of a future deployment.
We also evaluate the computational load needed by the Central Processing Unit (CPU) to execute the SIMD and OpenCL. Indeed, OpenCL model is based on software APIs to interact with the hardware, thus, an evaluation of the software processing is reported, highlighting the differences between purely software (i.e., SIMD on a high-end processor) and co-design (i.e., OpenCL) models over hardware. Furthermore, a manager capable to automatically switch the computation from software to hardware is deployed. Such a manager is utilized to offload the processing into the hardware when the CPU suffers from excessive computational load. This manager is a first attempt to provide dynamic reconfiguration between hardware and software, which can be considered for the 5G context.
Finally, we discuss design productivity to assess the complexity of creating and compiling OpenCL kernels to the FPGA in a standard context.

## II. 5G Infrastructure

The fifth generation of mobile communications (5G) must provide high level of efficiency, flexibility, and scalability in the Radio Access Network (RAN). Indeed, applications such as enhanced mobile broadband (eMBB), massive machine type communications (mMTC), and ultra reliable low latency communications (URLLC) [6] demand requirements that the previous generation (4G) mobile communications cannot satisfy. Thus, the RAN has to be re-thought.
In 4G, the evolved Node B (eNB) represents the hardware and software located at the antenna site, enabling the wireless access of many User Equipments (UEs) to the mobile network infrastructure. Here, the protocol stack and signal processing, which handle the data from mobile devices, are resolved. This approach has many limitations, in particular for what concerns flexibility and scalability, since the eNB is structured to provide connectivity for a certain number of devices and it is not able to adjust the power consumption (e.g., putting in standby some functionalities) in case of less users connected. On the contrary, if an area requires a higher number of users to be served, another eNB has to be deployed.
An important novelty of 5G is splitting the eNB into two different entities: the Central Unit (CU) deployed in central locations and the DU deployed near the antenna. This architecture is called Next Generation RAN (NG-RAN) or C-RAN, where the "C" stands for both centralized and cloud. Indeed, CU and DU can be even virtualized and deployed in the cloud [7]. Likewise, the eNB in the 4G infrastructure evolves in gNB. Centralizing the CU leads to several benefits including economy of scale, reduction of the maintenance for the cell towers, and performance improvement due to better coordination between antennas. Economy of scale refers to the possibility of deploying less hardware devices compared to the actual 4G infrastructure. For instance, a single large router for network access can serve the CU rather than many little routers serving a single eNB. In this way, the CU can be designed to be able to move router ports from under-utilized CUs to over-utilized CUs. The maintenance can be reduced by upgrading the software of centralized CUs. Finally, the performance in terms of lower call drop rates and downlink data rates are 30% improved with the centralized approach [8].

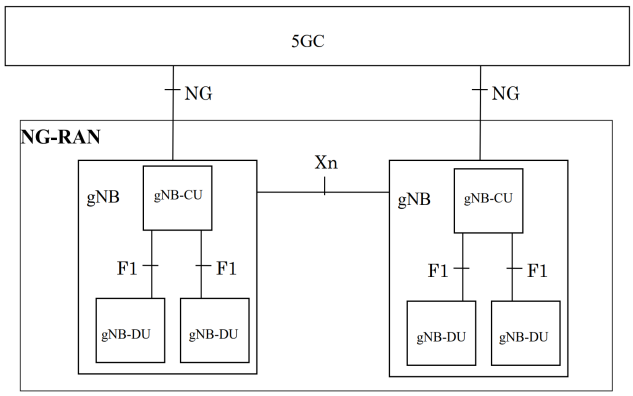



Fig. 1: 5G architecture

The NG-RAN architecture is depicted in Figure 1. Here, many gNBs are connected to the 5G Core (5GC) which manages the UE communication with the Data Network (DN). The link between the gNBs and 5GC is the NG interface, which corresponds to the *backhaul*. Each gNB consists of a CU, gNB-CU in Figure 1, connected to many DUs, gNB-DU in Figure 1, by means of the F1 interfaces, which are the *midhaul* interfaces. Often, based on the DU location, an additional element, including only the Radio Frequency (RF) functions (e.g., filtering, mixing, upconversion/downconversion, Digital to Analog Conversion (DAC)) called RRU (Remote Radio Unit), is deployed and it is connected to the DU by the, so called, *fronthaul* interface. Many protocols have been defined to transmit the data in the fronthaul link, such as Common Public Radio Interface (CPRI), ethernet-based CPRI (eCPRI) and Next Generation Fronthaul Interface (NGFI). An analysis of their performance is presented in [9].
As the gNB is splitted into CU and DU, so are the protocol stack functionalities. This is an important novelty of 5G called *functional split* where the functionalities can be splitted according to the scheme in Figure 2. The 3GPP association established the requirements, in terms of both latency and bandwidth, reported in Table I [10], for the communication network connecting DU and CU as a function of the considered functional split. As reported in Table I, when most of the functions is implemented in the CU, the requirements are stricter.

| Protocol Split Option | Required bandwidth | | Max allowed one-way latency |
|---|---|---|---|
| Option 1 | DL: 4Gb/s | UL: 3Gb/s | 10ms |
| Option 2 | DL: 4016Mb/s | UL:3024 Mb/s | 1.5∼10ms |
| Option 3 | lower than option 2 for UL/DL | | 1.5∼10ms |
| Option 4 | DL:4000Mb/s | UL:3000Mb/s | approximate 100$\mu$s |
| Option 5 | DL: 4000Mb/s | UL: 3000 Mb/s | hundreds of microseconds |
| Option 6 | DL: 4133Mb/s | UL:5640 Mb/s | 250$\mu$s |
| Option 7-1 | DL:10.1∼22.2Gb/s | UL:16.6∼21.6Gb/s | 250$\mu$s |
| Option 7-2 | DL:37.8∼86.1Gb/s | UL:53.8∼86.1 Gb/s | 250$\mu$s |
| Option 7-3 | DL:10.1∼22.2Gb/s | UL:53.8∼86.1Gb/s | 250$\mu$s |
| Option 8 | DL:157.3Gb/s | UL: 157.3Gb/s | 250$\mu$s |

TABLE I: Bandwidth and latency requirements for each functional split

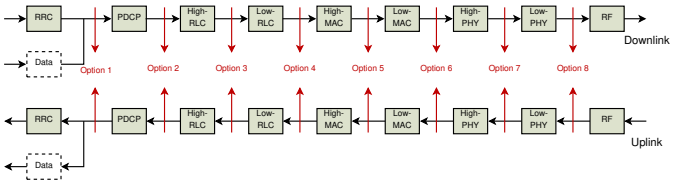



Fig. 2: 5G functional split

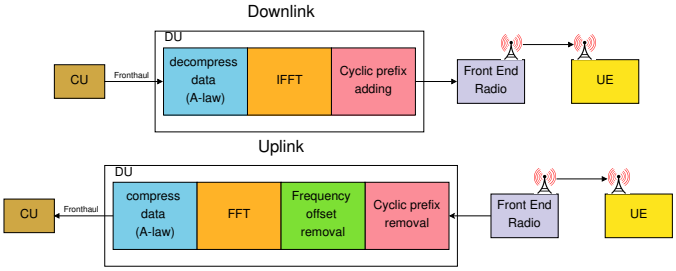



Fig. 3: DU processing with Option 7-1 functional split

This paper focuses on the Option 7-1 functional split as case study, which corresponds to implement the low PHY level functionalities at the DU while the other ones are implemented at the CU. Such functional split is implemented by the OpenAir Interface (OAI) experimental framework used to establish a 5G communication [11]. Many other 5G tasks can be also considered as good candidates to be accelerated into the hardware, up to the entire layers of the 5G protocol stack. However, offloading a lot of processing into the hardware may result in a huge area required by the FPGA, as well as a costly solution has to be adopted. Thus, only the tasks that require strong computation capabilities with latency critical constraints shall be deployed in hardware, exploiting the reconfigurable computing approach where the hardware represents the acceleration part, handled by the software.

Figure 3 shows the functionalities implemented in the DU both in downlink and uplink, considering option 7-1 split based on the OAI implementation. In downlink direction, the DU receives compressed the In-phase/Quadrature (I/Q) samples from the midhaul in the frequency domain. The data in the midhaul is compressed to save capacity and the DU applies the decompression algorithm based on the A-law [12]. Each I/Q sample consist of 32 bits divided into 16 bits that represent the real part and 16 bits for the complex part. Then, the DU performs the inverse Fast Fourier Transform (iFFT) to convert the samples in the time domain and it adds the cyclic prefix, which is a guard interval to avoid inter-block interference between successive symbols. As last step, the DU sends the data to the radio front end, which has the role to perform the DAC and it handles the communication with the UE, which is a generic end-user device.

In the uplink direction, dual operations than the downlink direction are performed with the addition of the frequency offset removal. Indeed, such an offset is considered in baseband receiver to address the non-idealities problems such as sampling clock offset, IQ imbalance, power amplifier, phase noise and carrier frequency offset non-linearity. The processing described above is a part of the OFDM modulation/demodulation that is utilized for 5G communications. As specified in Table I the communication network between DU and CU present strict latency requirements.

## III. Acceleration with Reconfigurable Hardware through OpenCL

### A. OpenCL generalities

OpenCL is a framework for writing programs that can be executed across heterogeneous platforms such as CPUs, General Purpose computing on Graphics Processing Units (GPGPUs), Digital Signal Processors (DSPs) and FPGAs. OpenCL specifies a programming language (based on C99) for programming these devices and API to control the platform and execute programs from the host side (i.e., computational unit on which the host program runs to control the OpenCL device). The main advantage of this framework is the possibility to write a single program that can run on heterogeneous platforms seamlessly, even if many optimizations must be done for each platform to reach the best performance. Moreover, it exploits the parallel computing approach which enhances the application performance. Two main parallel programming models are possible with OpenCL: task parallelism and data parallelism. In the first one, the application can be decomposed in several tasks with different computation loads. Each task can be mapped onto Processing Elements (PEs), that can run concurrently. However, it suffers of load balancing issue since the processing finishes only when the last PE finishes its task. Thus, an unbalanced loading brings to a loss of performance. In data parallelism model, the application can be seen as

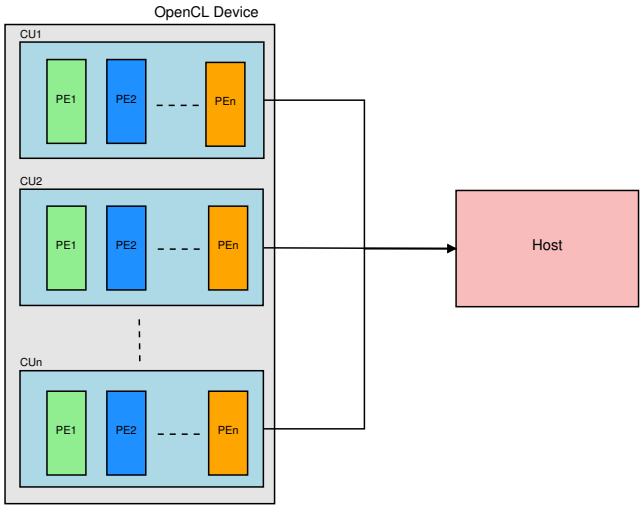



Fig. 4: OpenCL platform model

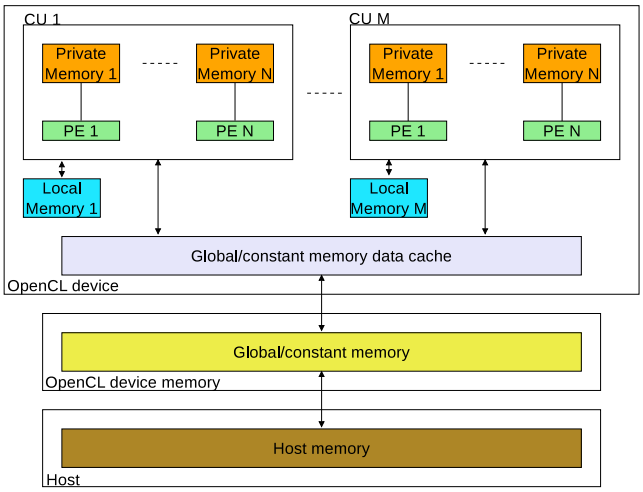



Fig. 5: OpenCL memory model

collection of data elements that can be processed concurrently. This means that a single task can be applied to each data element. This model is possible when a processing on arrays or matrix is needed since each element of the structure can be easily computed by a single task.

The OpenCL platform model is shown if Figure 4. Here, a single *host* acts as a master capable to interact with many OpenCL *devices* which are the components where a stream of instructions execute. Such a stream of instructions is called *kernel* and it represents the program executed by the OpenCL device. Each device is called Compute Devices (CDs) and it can be a CPU, GPGPU, DSP or FPGA. Each CD can be divided in OpenCL Compute Units (OpenCL CUs) which are further divided into PEs. The PEs represent the smallest OpenCL computation units and all the processing run within the PEs.

The host program and one or more kernels must be considered to create an OpenCL application. The host program has the role to initialize and interact with the kernels by using the standard OpenCL API [13]. It establishes the context for the kernels as well as the command queues to be sent to the device. Moreover, it defines the memory objects which are the buffer to be read/written by the kernels. Each kernel needs its command queue. The host places the commands into the command queue and the commands are then scheduled on the associated devices. The commands supported by the OpenCL are: i) kernel execution commands that are necessary to execute a task into a PE of the CD, ii) memory commands that transfer memory objects from/to the device and iii) synchronization commands that refer to constraints on the order of commands execution.

The kernels execute in the CD and they are the processing core of the OpenCL model. Moreover, two types of kernel are defined: pure *OpenCL* and *native* kernels. The first ones are functions written and complied by OpenCL framework. The native kernels are functions created outside the OpenCL environment and it is possible to call them by means of function pointers.

Particular attention must be dedicated to the OpenCL memory model since it enables the communication between the host and the device. Figure 5 shows the memory regions from the host to the devices. Here, five memory regions are defined:

- **Host Memory**. This memory region belongs to the host and it cannot be accessible by any other instance of the OpenCL architecture.
- **Global Memory**. This region enables the communication from the host to the device and vice versa. Indeed, it can be accessible by all PEs. In Figure 5, a further cache level is depicted. This level may be present depending on the capabilities of the device.
- **Constant memory**. This memory has a read-only access for the PEs. During initialization, the host writes this region which remains the same during the kernel execution.
- **Local Memory**. This region is shared by all the PEs belonging to a certain CU and it is used to allocate variables that are commons for all the PEs. Moreover, if the CD has its own memory, it is deployed as a portion of the CD memory, otherwise it is implemented in the global memory. In this case, the access could be slower, depending on the global memory interface bandwidth.
- **Private Memory**. This memory region belongs to the single PE. The variables stored are not visible by any other instance of the OpenCL architecture.

### *B. Specific OpenCL constructs and kernels for reconfigurable hardware*

In this Section, a focus on the kernels and tools considered to exploit the OpenCL model for FPGA is provided. In particular, we describe the relation between the two types of kernels that can be deployed in hardware as well as the the SDK (Software Development Kit) used for this work, which is capable to create a hardware application starting from these kernels.

The OpenCL framework for FPGAs allows the designer to simply create an application by means of HLD approach. For this paper, the Intel FPGA SDK for OpenCL is considered and it supports the OpenCL specification 2.0 [13]. The Intel FPGA SDK for OpenCL Offline Compiler compiles the kernels to an image file used by the host to program and run the kernel on

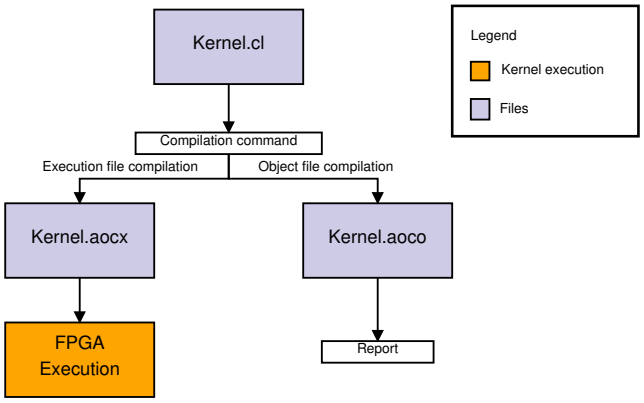



Fig. 6: Intel FPGA SDK for OpenCL workflow

the FPGA. The model previously described remains the same with an important difference: the FPGA exploits spatial implementation of a program, thus the instructions are executed when the data is ready. However, in sequential programming (considered for CPU, GPGPU, DSP), the program counter controls the instructions to run. Indeed, even if the data is available to be computed, the processing is executed one instruction at a time, following the order of the program counter. This method is time-dependent since the instructions are executed on the hardware across time. Designers have to take into account this crucial difference to create an application FPGA-oriented. Indeed, the way the code is written permits to avoid FPGA area overhead or performance degradation.

Figure 6 shows the Intel FPGA SDK for OpenCL kernel compilation flow. The Intel FPGA SDK for OpenCL Offline Compiler compiles the kernel and it creates the hardware configuration in a single step (i.e., kernel.aocx). An intermediate step, that creates Intel FPGA SDK for OpenCL object file (i.e., kernel.aoco), provides the report on area usage and performance bottlenecks, especially in the memory access.

Two types of kernels are supported by the OpenCL for FPGA framework: NDRange and Single Work-Item (SWI) kernels. A NDRange kernel permits to partition the data among the PEs. In particular, the OpenCL runtime system creates an integer index space (i.e., NDRange) where each instance of the kernel is executed. Each instance is called *work-item* and it is identified by its coordinates in the NDRange. This kernel is suitable when there is no data dependency that makes the partitioning easy. The commands run from the host create the collection of the work-item and execute all the work-items in parallel.

Executing NDRange kernels with size (1,1,1) corresponds to run a SWI kernel. SWI kernels permit to control the pipeline since only one work-item is executed. This approach is better when an application has loops or data dependencies. However, the NDRange tries to parallelize the computation with several work-items but the performance could be degraded due to a possible stalls for conflicts with data. Unlike NDRange kernels, single work-item kernels follow sequential programming approach, more similar to C programming. Anyway, the data parallelism is applied on SWI kernel type by pipelining the iterations of loops. Indeed, custom pipeline is possible and data access patterns can be modified using ad-hoc ”pragmas”.

## IV. Related works on OpenCL hardware acceleration

To the best of our knowledge, this study is the first that considers OpenCL acceleration in the context of 5G. A number of recent works relies on the OpenCL language for FPGAs to deploy advanced services such as Convolutional Neural Networks (CNNs), Finite Impulsive Response (FIR) filters or image processing accelerations. In [14], the authors provide an exhaustive analysis about the Time-Domain Finite Impulsive Response (TDFIRs) and Frequency-Domain Finite Impulsive Response (FDFIRs) implemented in the FPGA by means of OpenCL. In particular, they focus on the performance of several FIR implementations on the FPGA as well as performance comparison between FPGA and GPGPU. Moreover, an analysis on the area usage and power consumption is reported.

An optimized implementation of a CNN is proposed in [15], where the authors consider Deep Learning Accelerator (DLA) to overcome the memory bound limit of the CNN deployment into FPGA. They maximize the data reuse and minimize external memory bandwidth. Moreover, Winograd transform is further applied to boost the FPGA performance. The results of such a scenario show that the system is 10x faster of the state-of-the-art implementation and it has comparable performance with GPGPU CNN deployment.

Another approach to improve CNN computation on the FPGA is described in [16]. Here, an analytical performance model regarding area usage is presented and applied to a CNN implementation. Authors show a bottleneck on the on-chip memory bandwidth and provide a 2D interconnection between PEs and local memory to reduce the on-chip memory requirement. Moreover, a 2D dispatcher is further designed to reduce the external memory bandwidth.

Optimization techniques for image processing are presented in [17] where the authors report an optimization review before presenting a spatial-spectral classifier for Hiperspectral Image (HI). This use case is based on the K Nearest Neighbours (KNN) filter deployed into the FPGA by means of OpenCL model. Both SWI and NDRange kernels are tested in terms of area usage and processing time. Moreover, a comparison between the output of KNN filtering and Support Vector Machine (SVM) classification is reported to demonstrate the image processing improvement.

Lastly, the implementation of an efficient 3D Fast Fourier Transform (FFT) is presented in [18]. The core of the work is to improve the pipelines of the FFT providing a valid alternative to HDL deployment. Moreover, the authors report a performance evaluation between OpenCL, HDL, GPU and CPU. The results show that the OpenCL performance are consistent with the HDL ones with the advantage of using fewer resources than IP core design. A significant improvement, in terms of execution time, is highlighted when the processing is performed via OpenCL and HDL with respect to CPU and GPU platforms performance.

## V. Implementation

This section describes the implementation of the 5G DU processing by means of OpenCL, SIMD and HDL, as well as the considered experimental setup for each method. Figure 7 depicts the block diagram of both HDL and SIMD deployments.

Figure 7a shows the block diagram for the SIMD approach. The data coming from midhaul is uncompressed and processed by the iFFT task based on SIMD instructions. In this work, we consider the Intel Advanced Vector Extension 2 (AVX2) SIMD instruction set that performs operations on 256-bit vectorized data. Such instruction set achieves better performance than previous SIMD extensions, in particular with floating point calculation and data organization. The iFFT algorithm is computed by using the forward FFT and considering complex conjugation of the data before and after the forward FFT processing. Three SIMD functions are implemented to address the integer FFT Cooley-Tukey algorithm [20]: i) decomposition of the signal in frequency, which consists in the division of a frequency spectrum composed by N points into N frequency spectra each composed of a single point; ii) the second step considers the conversion of N frequency spectra into N time signals; iii) last step is the recomposition of N time signals into a single time signal. The last step is the most significant one in terms of computation complexity due to totally connected data dependencies to previous steps. Each input sample of the FFT consists on 32 bit integer type where the first 16 bits represent the real part and the second 16 bits the imaginary part. As discussed above, the Intel AVX2 SIMD instructions can process 256 bit at once, thus 8 FFT samples are computed with a single instruction, ensuring a good degree of data parallelism. Since iFFT is considered, all the samples are points of signal in frequency to be converted to a signal in time domain. The output data is computed by the cyclic prefix insertion SIMD function that inserts redundant bytes before iFFT samples in the stream. The number of bytes to be added is summarized in Table II together with the complete OFDM 5G numerology considered for this work [21], [22]. Even if the present value considered as maximum symbol size is 4096 [23], we also evaluate the results up to 8192 symbol size to understand which are the better approaches to address a possible future implementation of this size, since the 5G PHY layer is still under study.

Figure 7b shows the top-level block diagram for the HDL use case. Here, the data in the frequency domain from the midhaul is uncompressed and sent to the FPGA by means of the PCIe interface. Then, the data is processed by the Intel iFFT IP core that returns the samples in the time domain. We consider the streaming iFFT setting that allows continuous processing of input data and it provides continuous complex data stream as output, without halting the dataflow. The streaming iFFT is based on quad output iFFT engine which minimizes the transform time. Quad-output refers to the throughput of the internal iFFT butterfly computing [24]. As described for SIMD, the butterfly represents the last of the three steps necessary to compute the iFFT, according to the Cooley-Tukey algorithm. The engine implementation computes all four radix-4 butterfly complex outputs in a single clock cycle. The output of the iFFT block is sent to the custom IP for cyclic prefix addition. The IP core is synchronized at the output of the iFFT block to add the cyclic prefix bytes without adding clock cycles. The output of the design is a burst of samples in time domain composed by some redundant samples representing the cyclic prefix and the complex samples converted from frequency to time. The number of cyclic prefix bytes depends on the OFDM symbol size considered, as summarized in Table II.

Regarding OpenCL, we choose to implement the 5G OFDM computation inside the DU to evaluate both the FPGA area usage and processing time for OpenCL offloading. Thus, the iFFT and the cyclic prefix addition kernels are deployed in hardware, according to Figure 3. Dealing with the iFFT task, we implement an integer algorithm based on the Cooley-

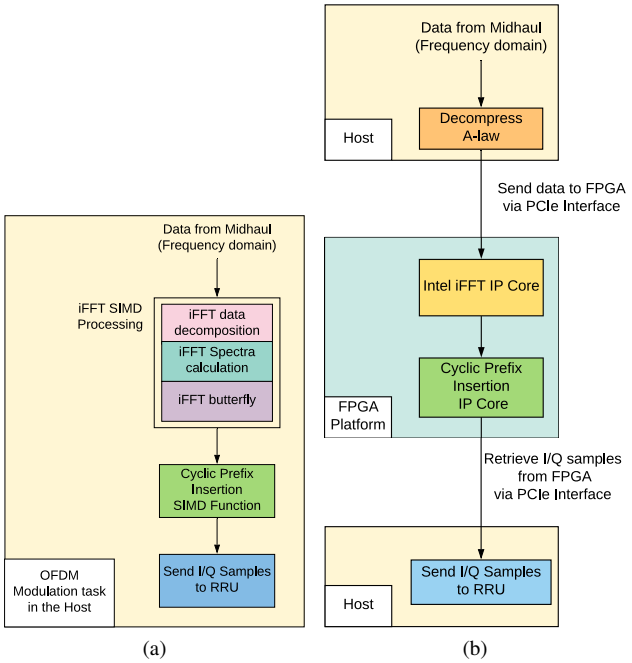



Fig. 7: Architecture descriptions for a) SIMD and b) HDL implemented 5G DU processing

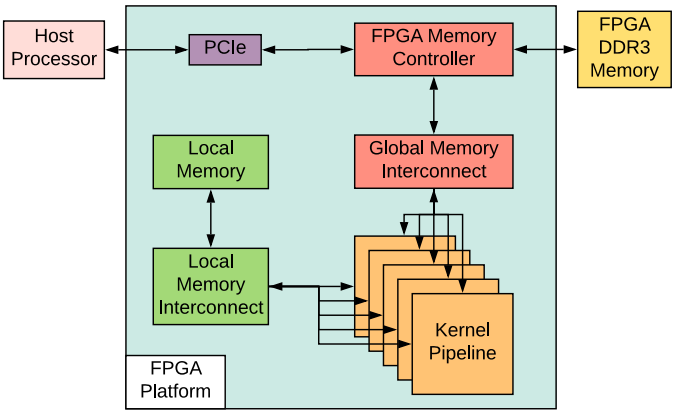



Fig. 8: OpenCL system setup [19]

| OFDM Symbol Size (N) | Cyclic prefix length (M) | Resource Blocks (RBs) | Channel Bandwidth (MHz) |
|---|---|---|---|
| 128 | 10 | 6 | 1.4 |
| 256 | 20 | 15 | 3 |
| 512 | 40 | 25 | 5 |
| 1024 | 80 | 50 | 10 |
| 2048 | 160 | 100 | 20 |
| 4096 | 320 | 275 | up to 200 |
| 8192 | 640 | 550 | up to 900 |

TABLE II: OFDM numerology

Tukey approach and the same steps considered in the SIMD deployment are taken into account. In the OpenCL for FPGA scenario, the implementation of an integer algorithm is preferable to avoid excessive usage of DSP blocks of the FPGA. Indeed the floating point calculations require complex circuits based on adders, subtractors and multipliers that use DSP resources of the FPGA. The algorithm developed for the OpenCL application receives as input a number of samples equals to the OFDM symbol size. To complete the OFDM processing at the DU side, the cyclic prefix addition kernel is also offloaded to hardware. It prevents interferences between OFDM symbols.

Figure 8 shows the setup with the internal connection between host, FPGA and FPGA memory. The communication interface between the host and the FPGA is a PCIe that can directly interact with the global memory by means of a memory hardware controller. The PCIe bandwidth is around 4 GB/s. The maximum symbol size considered for this work is 8192 and each samples is 32 bit based. Thus, the maximum bandwidth for the OFDM used to transfer the data from the host to the FPGA is 262144 bits/s, significantly below the maximum PCIe bandwidth. The Random Access Memory (RAM) of the board represents the global memory of the OpenCL environment and it can be accessible from the kernel pipeline, exploiting the global memory interconnect to interleave data accesses and hide latencies. The intermediate variables necessary for the kernel computation can be stored in both the RAM blocks (local memory) and the FPGA registers (private memory). In case of using RAM blocks, a local memory interconnect is necessary to access the memory itself.

Table III summarizes the characteristics of the Terasic DE5-Net high-end board exploited for this work as OpenCL device. Moreover, the host is based on the the processor Intel Xeon W-2133 equipped with 12 cores with a 3.60GHz clock frequency.

| Board | Terasic DE5-Net |
|---|---|
| FPGA | Intel Stratix V GX |
| Resources | ∼622000 LEs<br>256 DSPs |
| RAM | 2x 4GB DDR3 @933 MHz each |
| Interface to Host | PCI Express (PCIe) x8 lanes |

TABLE III: Terasic DE5-Net board specifications

The host is capable of both compiling and starting the kernel. Once the FPGA is programmed, the host starts the kernel and reads/writes the global memory exploiting the OpenCL API.

The kernel considered for this work is SWI based and it is optimized to reach high global memory throughput and to reduce memory latency transfer. Closing the gap between kernel frequency and memory controller frequency achieves the saturation bandwidth, which is an important parameter to obtain the above optimizations. In particular, memory controllers usually run at $\frac{1}{8}$ of the clock of the external memory for DDR3 memory types. Considering a bus width of 64 bits for DDR3 memory, the memory bandwidth is saturated if the kernel runs at the same operating frequency than the memory controller ($\frac{1}{8}$ of memory frequency). Thus, the saturation bandwidth SB is calculated with Eq.1, where BW is the bus width, $\mathrm{N}_{ddr}$ is the number of DDR banks, MF is the memory frequency and KF is the kernel frequency.

$$\begin{aligned} \mathrm{SB} &= \mathrm{BW} \times \mathrm{N}_{ddr} \times \frac{\mathrm{MF}}{\mathrm{KF}} \\ &= 64\ bit \times 2 \times 8 = 1024\ bits/clock\ cycle \end{aligned} \tag{1}$$

This value is the maximum achievable SB and it can be obtained under the following conditions: i) the kernel has to exactly have one read and one write memory access. Then, ii) no stall shall occur, iii) the memory accesses have to be word aligned and iv) balanced readings and writings have to be done (the same number of bytes have to be read and written from/to the memory). Otherwise, if one or more conditions are not met, a significant overhead of computing cycle occurs and the execution time performance is degraded. These conditions permit to increase the OpenCL kernel frequency up to $\frac{1}{8}$ of the clock of the external memory. As described in Table III, the Terasic DE5-Net board is based on two banks of DDR3 memory, 933 MHz each, with double data-rate. This means that we can consider 1866 MHz (i.e., $2 \times 933$ MHz) as external memory clock and the FPGA memory controller is $\frac{1}{8}$ of this value (i.e., ∼233 MHz). Hence, the saturation bandwidth is achieved if the kernel frequency is equal to 233 MHz (at least, as close as possible). In this work, all the OpenCL kernels developed for the evaluation reach this working frequency, ensuring the maximum global memory throughput.

The main advantage of the OpenCL model is certainly the portability, but each architecture requires its own optimization since specific instructions have to be executed to achieve better performance. For instance, the considered SWI kernel can run in a GPGPU but with very low throughput [25]. Indeed, the GPGPU is equipped with thousands of tens of SIMD cores, each with tens of individual cores capable to run a single work-item. Thus, NDRange kernels are more suitable for GPGPUs. Indeed, their SIMD cores usually require to execute at least 16 to 32 times the same instruction on separate data to reach their peak performance. Such strong data parallelism is the reason of existence of NDRange kernels. Hence, the platform optimization is an important process before compiling an OpenCL application and bad optimizations may lead to a platform inefficiency for OpenCL application. The choice of a SWI kernel is preferable in FPGA to NDRange kernels due to possible conflicts in the memory accesses. Moreover, SWI kernels permit to control the whole pipeline, as well as the memory accesses.

The kernel developed for the application is loop-based.

Each loop inside the SWI kernel is unrolled, which means that the system tries to run each loop iteration in parallel. Hence, by means of a `#pragma unroll N` directive, the compiler is forced to unroll the loop *N* times. This reflects into a speed up of *N* times in the execution performance of the loop (without specifying the *N* factor, the compiler automatically unrolls the whole loop). Thus, to achieve the best performance, it is important to avoid dependencies on the pipeline and concurrent memory accesses. These conditions are fundamental to optimize the SWI kernel, otherwise the OpenCL compiler needs too much FPGA area to efficiently address data dependencies (i.e., implementation of multiple instances of the same variable). To address this optimization, we store the variables inside the FPGA registers to reach lower processing time. Indeed, if we consider on-chip memory (RAM blocks) to store the kernel variables necessary for the processing, several accesses are required to the block RAMs, reducing the processing time. For instance, unrolling loops with large unroll factor can cause memory conflicts on the on-chip memory, if a precise pattern for the accesses is not well-defined. Moreover, the OpenCL compiler tries to optimize the accesses but area usage and latency increases due to the concurrence. Storing the kernel variables as registers permits to achieve higher performance since no memory access is required and no conflicts are possible for concurrent access. As matter of example, we can consider to store an array of 64 elements 16-bit each. The compiler considers 64 registers of 16-bit width. In this scenario, no conflicts are possible since every element of the array has its own register. However, this methods requires higher FPGA resources, in particular Adaptive Look Up Tables (ALUTs) and Flip Flops (FFs). Too many variables stored as registers can rapidly increase the overall logic utilization and the OpenCL design does not fit into the FPGA.

Another optimization is to unroll read and write loops of global memory access to achieve the compile-time *memory coalescing*. This approach is typical to have uniform and regular memory accesses. In particular, an irregular or not aligned access to the memory tends to decrease the performance in FPGA. Furthermore, the memory transfer between the host and the global memory has to be aligned to exploit the Direct Memory Access (DMA) acceleration. Unrolling global memory loops permits to have an efficient access, exploiting the whole communication bandwidth of the memory (as described for Eq.1). On the other hand, choosing big values of *N* correspond to have accesses larger than the external memory bandwidth, thus the best unrolled factor is the one that saturates the memory bandwidth.

A significant optimization on area usage is done by avoiding function calls in the SWI kernel. Indeed, every function call is implemented as a circuit on the FPGA, resulting in a large usage of FPGA resource. Functions call prevents the compiler to create a correct report about FPGA area usage that makes the debugging harder.

Finally, writing OpenCL code for hardware-oriented programming is a best practice to optimize the SWI kernel. This aspect leads to an improvement in terms of both performance and area usage. As a matter of example, we can consider a simple if-then-else directive implemented in both software and hardware. In software (i.e., sequential programming), the if-then-else statement presents an issue in the pipeline. Indeed, there are two possible instructions to be fetched and the CPU has to take a decision based on the condition. Modern CPUs attempt to predict the correct instruction to be executed. Then, the processor fetches the instructions based on such a prediction and, in case of wrong prediction, it discards the partially executed instructions in the pipeline. In this scenario, the pipeline is not stalled, ensuring a smooth execution of the directives. However, in hardware approach (i.e., spatial programming), both ”if” and ”else” instructions are mapped into hardware circuits which lead to a FPGA area overhead since only one circuit is considered at a time. Using conditional statements (i.e., `out = (condition) ? in_1 : in_2`) instead of if-then-else statements can reduce the FPGA resources in many cases. Indeed, the if-then-else construct follows the sequential approach and infers a priority routing network, which needs higher resources. On the other hand, conditional statements are mapped as a unique MUX circuit in hardware and they yields better results in terms of both resource saving and performance since they follow the concurrent approach [26].

## VI. Performance evaluation and results

To evaluate the offloading of 5G computation to reconfigurable hardware, we consider three different performance parameters: FPGA area usage, overall execution time and computational load. All the parameters refers to the implementation of the OFDM downlink processing for Option 7-1 functional split in the DU part of the 5G architecture, as described in Figure 3. The first parameter permits to evaluate the FPGA resources needed to offload the computation. Moreover, we report the resource utilization for both HDL and OpenCL methods to have a comparison between these two different approaches.

The second parameter considers both processing time and memory transfer time performance. In particular, memory transfer time is evaluated for both OpenCL and HDL approaches and it is defined as the time necessary to send the data from the host to the device and vice versa. However, the pure processing is defined as the time to execute the OFDM processing for all the deployments, without considering memory transfers. Processing time analysis is done by comparing the OpenCL platform with HDL and software SIMD implementations. Moreover, a model is developed to predict the pure processing time values for those OFDM symbol sizes that not fit into the FPGA (for OpenCL case). Likewise, the present 5G deployment considers OFDM symbol sizes up to 4096, thus a SIMD model is necessary to estimate the processing time for bigger OFDM symbol size. The SIMD approach is already implemented in the OAI 5G framework, thus it can be considered as a benchmark to evaluate the processing time performance and the correctness of the data. Moreover, it has the best software performance for the iFFT computation [27] which represents the most significant part of

| OFDM Symbol Size | Logic Util (ALMs) | | Registers | | Total Memory bits | | DSPs | |
|---|---|---|---|---|---|---|---|---|
| | HDL | OpenCL | HDL | OpenCL | HDL | OpenCL | HDL | OpenCL |
| 128 | 2% | 29% | 1% | 13% | <1% | 7% | 2% | 24% |
| 256 | 3% | 51% | 1% | 25% | <1% | 7% | 2% | 48% |
| 512 | 5% | 95% | 2% | 49% | <1% | 8% | 2% | 86% |
| 1024 | 9% | No FIT | 3% | No FIT | <1% | No FIT | 2% | No FIT |
| 2048 | 16% | No FIT | 4% | No FIT | 1% | No FIT | 5% | No FIT |
| 4096 | 31% | No FIT | 8% | No FIT | 2% | No FIT | 5% | No FIT |
| 8192 | 60% | No FIT | 15% | No FIT | 3% | No FIT | 5% | No FIT |

TABLE IV: OpenCL and HDL FPGA resource usage

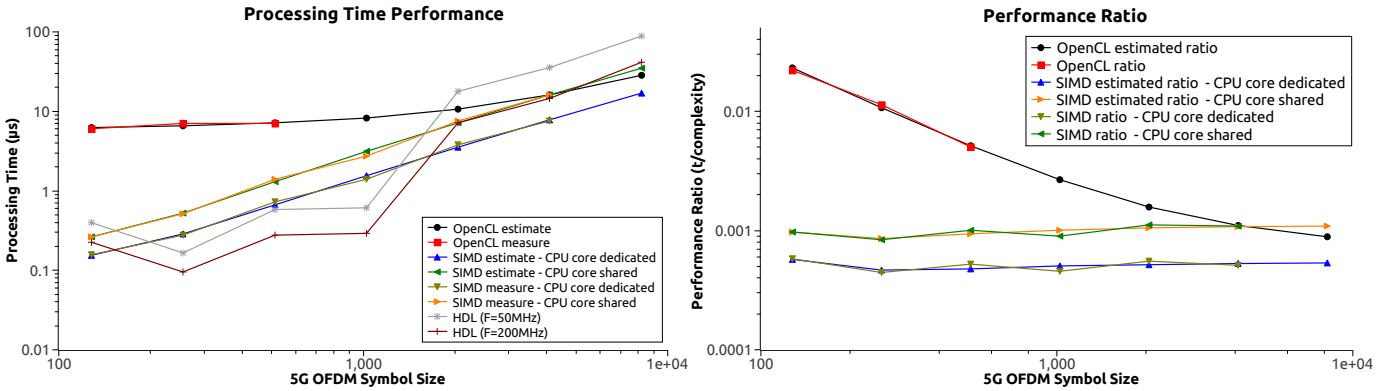
the OFDM processing.
Finally, the computation load parameter is referred to the CPU processing load for both SIMD execution and OpenCL hardware offloading. In this case, the goal is to show if the OpenCL approach permits to save the CPU processing.

### A. FPGA area usage evaluation

To evaluate the area usage in the FPGA, we report the resource utilization for both HDL and OpenCL approaches. Area usage is reported in Table IV where the results are presented for different OFDM symbol sizes. The values refer to the resource occupation, given by the report generated during the OpenCL kernel compilation. Here, it is evident the resource usage differences, which reflect the different model application of HDL and HLS. Indeed, the FPGA resources used for the OpenCL implementation are much higher than the HDL cases. The reason lies in the abstraction layer introduced by the OpenCL. However, the OpenCL resources could be lower since we consider the fully unrolled implementation of the OFDM computation. This means that all the loops inside the OpenCL kernels are unrolled to improve the processing time to meet the 5G requirements, at the cost of significant increase of the area usage. In this scenario, the maximum OpenCL hardware acceleration is implemented for a 512 symbol size OFDM while for higher symbol size the OFDM does not fit.

### B. Overall execution time performance evaluation

To evaluate the overall execution time considered for the OpenCL approach, the pure processing time and the memory transfer time between the host and the OpenCL device are considered. Regarding the OpenCL memory transfer latency, it is calculated by subtracting the execution time of the OFDM kernel under exam with the execution time of a no operation loopback kernel (the data is sent from the host to the kernel and immediately forwarded by the kernel to the host). We measure around 233 µs for all the OFDM symbol sizes, considering a CPU core dedicated to run the OpenCL kernel. This time represents a bottleneck due to the data transfer and synchronization between the host and the device memory, as well as the reading and writing from/to the global memory (which is the DDR3 RAM for the considered board) and the FPGA. We also measure the transfer time in case of OFDM hardware offloading exploiting a single CPU core shared with other concurrent tasks, obtaining a value around 468 µs for all the OFDM symbol sizes. Such a value is around the double of 233 µs, measured in case of ideal condition of CPU dedicated core, which demonstrates that this latency is due to software synchronization with the PCIe interface, since the pure kernel processing time remains the same (the hardware execution does not change).
For these reasons, in a possible deployment of the OpenCL approach for the 5G communication, the PCIe interface between the host and the device represents a bottleneck that has to be avoided. A possible solution is to exploit the *auto-run* kernels and the OpenCL channels. The auto-run kernels permit to execute the processing in hardware without the interaction with the global memory. Indeed, the host starts the auto-run kernel which can process the data acquired by means of the I/O OpenCL channels. For instance, in the 5G context, such a kernel can gather the data to be directly computed exploiting the Ethernet or optic fiber hardware interfaces. Moreover, if more kernels are instantiated, the OpenCL channels enable the data transfer between the kernels without global memory interactions that increase the overall processing time.
Regarding the pure computation time for all the deployments, Figure 9 shows the processing time performance for SIMD, OpenCL and HDL and *performance ratio* trend for SIMD and OpenCL as functions of the OFDM symbol size from 128 up to 8192. Here, the processing time curves are both based on experimental results and estimation, except for the HDL case. As previously discussed, the PHY layer of 5G is still under study, thus the SIMD values are calculated according to the present symbol size which corresponds to 4096. However, according to Table IV, OpenCL kernels does not fit into the hardware for OFDM symbol size bigger than 512. This means that for both OpenCL and SIMD is necessary a model to estimate the processing time up to 8192 OFDM symbol size to understand the behaviour of these deployments.
The model to evaluate the processing time is based on the performance ratio calculated by taking into account the complexity of the OFDM processing. Indeed, in downlink side, the DU have to compute the iFFT and the cyclic prefix addition to complete the OFDM. Since the iFFT is based on the Cooley-Tukey algorithm, the complexity can be calculated by $N \times \log N$, where N is the OFDM symbol size considered, while the complexity of the cyclic prefix addition is M, where M is the number of samples to be added as cyclic prefix. Table V shows the complexity of the both iFFT and cyclic prefix addition. The number of samples considered for the




(a) (b)

Fig. 9: a) OpenCL, SIMD, HDL measured and estimated processing time; b) performance ratio trend obtained by dividing performance time measured by OFDM complexity

| OFDM symbol sizes | iFFT complexity O(N*logN) | Cyclic prefix addition complexity O(M) |
|---|---|---|
| 128 | 269.72 | 10 |
| 256 | 616.51 | 20 |
| 512 | 1387.15 | 40 |
| 1024 | 3082.55 | 80 |
| 2048 | 6781.60 | 160 |
| 4096 | 14796.23 | 320 |
| 8192 | 69049.06 | 640 |

TABLE V: DU OFDM computation complexity for Option 7-1 split

cyclic prefix are the same presented in [28] and summarized in Table II, while we estimate a possible value for 8192 OFDM symbol size. Since the complexity of the cyclic prefix is much smaller than the iFFT complexity, we can approximate the overall complexity equal to the iFFT one. The performance ratio is calculated by dividing the measured processing time by the complexity. The trends are obtained by the interpolation of the performance ratio values calculated for different OFDM symbol sizes. To get the curves for the performance ratio, we use the Matlab curve fitting tool and we choose the equation that maximizes the *R factor*. Such a factor is defined as the square of the correlation between the response values and the predicted response values and it can be a value between 0 and 1. A value close to 1 indicates that a greater proportion of variance is accounted for by the model.

The OpenCL trend follows the exponential law while the SIMD trend can be considered as a second order hyperbola (Eq. 2).

$$\text{OpenCL} => ax^b + c \quad \text{and} \quad \text{SIMD} => a + \frac{b}{x} + \frac{c}{x^2} \qquad (2)$$

Such models permit to estimate the processing time for both SIMD and OpenCL and they are depicted in Figure 9b in logarithmic scale, where it is evident the differences between the two trends. Indeed, OpenCL has a better processing feature when the data size increases due to its pipelined approach. Moreover, the OpenCL performances are not expected to significantly increase since we considered a fully parallelized kernel. This approach results in a more resource usage, as shown in Table IV, but the performance considerably improve. However, the SIMD computation is lower for smaller data processing that reflects into an important distance between the two curve for OFDM symbol size less than 2048. Indeed, the SIMD model follows the sequential computing approach, typical of the software, that increases computation for large data batch. Performance ratio estimated values are calculated starting from the measured processing time, which permits to build part of the performance ratio plot. In particular, the OpenCL ratio trend is also based on 16 and 64 symbol sizes, to have more points for a better performance ratio estimation, even if these are not possible OFDM symbol sizes and they are not shown in Figure 9b.

Regarding the processing time, depicted in Figure 9a in logarithmic scale, the SIMD implementation has constant performance which reflects into a parabolic behaviour in linear scale. Moreover, for SIMD evaluation, two different curves are depicted, which correspond to the computation of the OFDM task exploiting a CPU core dedicated and a CPU core shared with concurrent tasks. The performance time trend for SIMD considering a core dedicated is the best one after 2048 symbol size but it reflects an ideal situation that is difficult to meet in 5G. Indeed, one of the main novelty of 5G is the virtualization that require multiple treads to be executed in parallel, thus, for this analysis, we consider the OFDM SIMD processing along with other parallel tasks.

OpenCL has a almost linear behaviour (negative exponential in linear scale) and the performance are worse up to 2048 symbol size with respect to SIMD. However, the OpenCL trend increases less rapidly than SIMD and they have the same performance at 4096 symbol size, while OpenCL could be a better approach after this value. Furthermore, as discussed at the beginning of this Section, running OpenCL kernel in parallel with other software tasks does not affect the hardware processing time but only the data transfer latency. Thus, the OpenCL processing time under CPU core resources shared is the same of CPU core dedicated, shown in Figure 9a.

The same parameter considered for the OpenCL can be

taken into account for HDL deployment. Indeed, the overall execution time can be divided in the throughput latency (i.e., memory transfer time) and the calculation latency (i.e., processing time). The first one represents the latency due to the data transfer towards the OFDM processing hardware IP and it corresponds to N clock cycles, where N is the OFDM symbol size. The calculation latency represents the clock cycles needed to do the processing. Figure 9a shows HDL curves that only takes into account the calculation latency, which is the delay introduced by the OFDM computation task. Calculation latency is depicted in time domain after multiplying the clock cycles needed for the computation by the inverse of the FPGA operating frequency. Here, 50 MHz and 200 MHz working frequencies are imposed for the evaluation of the HDL curves: 50 MHz constraint as working frequency is respected while the 200 MHz is not met for OFDM symbol size bigger than 1024. Table VI shows the frequencies reported by the Intel Quartus Timing Analyzer according to the Slow 85C Timing model which provides timing delays for the FPGA operating under the conditions of i) slowest silicon for the specific speed grade, ii) low voltage and iii) 85C junction temperature. Such conditions represent a possible worst-case for the device. Indeed, since each FPGA that is cut from a silicon wafer has delay differences, the timing analyzer takes into account the worst performance of a specified speed grade. Moreover, low voltage decreases the transistor switching speed by decreasing electron mobility through the transistors. High temperature also affects transistor speed by changing the characteristics of the silicon material, leakage current, and electron mobility [29].

In Figure 9a, it should be noticed that the HDL curves have better performance of both OpenCL and SIMD for OFDM symbol size less than or equal to 1024. After this values, the HDL computation increased and the SIMD processing time is the best of the three approaches up to 2048 OFDM symbol size. Starting from 8192 OFDM symbol size, both OpenCL and SIMD approaches are expected to have less processing time. The rising processing time after 1024 symbol size is due to the increased number of clock cycles considered to complete the iFFT step in the Intel iFFT IP core.

Finally, a consideration about the technology is necessary. The Intel Xeon W-2133 used for the SIMD processing is produced with a 14 nm production technology while the Intel Stratix V FPGA is based on the 28 nm technology. This means that 14 nm FPGA (e.g., Intel Stratix 10 series) can achieve better performance in terms of both area usage (in particular for OpenCL approach) and working frequency that can significantly reduce the curves reported for the hardware scenario.

### C. Computational load evaluation

For the last evaluation, the performance parameter considered is the CPU load. Such a parameter can be further characterized by three CPU states during the computation: *idle* state, *system processing* state, and *user* state. The first one refers to the time that the CPU is idle and the system does not have outstanding I/O requests. The system processing state represents the CPU utilization that occurred to execute task at the system level (kernel). Moreover, this state refers to the interaction of the system with the external physical peripherals. The last state is the user state that stands for the CPU utilization while executing user level applications.

| OFDM Symbol Size | FPGA Fmax (MHz) | |
|---|---|---|
| | 50 MHz Constrain | 200 MHz Constrain |
| 128 | 131 | 231 |
| 256 | 126 | 222 |
| 512 | 100 | 209 |
| 1024 | 95 | 198 |
| 2048 | 78 | 194 |
| 4096 | 75 | 183 |
| 8192 | 71 | 150 |

TABLE VI: FPGA Timing analyzer frequency report for each OFDM symbol size

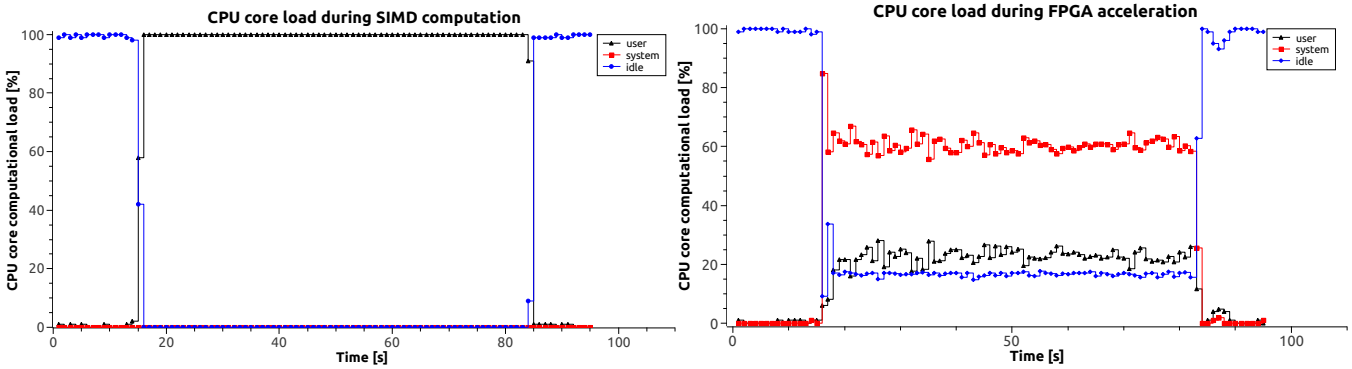
All the tests were performed considering a single core of the Intel Xeon processor. In particular, one core is unhooked from the operative system computation and exclusively dedicated to the processing of the OFDM algorithm.

Figure 10 reports the results of the CPU load during computation of the OFDM via SIMD (purely software) and via OpenCL while Figure 11 shows the performance of the CPU during a switch from software to the hardware processing, triggered by a manager. Such a manager is developed to control the CPU state in terms of processing load. Thus, the hardware offloading is enabled when the CPU computation is heavy for long time. For the manager, we consider a simple algorithm based on a threshold on the CPU load and the hardware acceleration is triggered once this threshold is overcame for a minute. However, more efficient algorithm that can take into account the average energy consumed by the CPU during the processing phase can be developed.

Figure 10a shows the CPU load when the processing is purely software. Here, the CPU core is in idle state only at the beginning and at the end of the test since the computation load required by the SIMD is constant at 100% for the entire duration of the OFDM execution. Moreover, the core is always in user state since no interaction with the external peripheral are required.

Figure 10b depicts the percentage of the CPU core during the OpenCL offload. In this scenario, the computation state is divided between the user state and the system state. Indeed, the user state is due to the host OpenCL application running while the system state reflects the PCIe interactions of the host with the global memory. It should be noted that the user application requires around 25% of the core load while the interaction with the PCIe interface is around 60%. However, the idle state is around 15% which is not found in the pure software use case.

Figure 11 show the transition between hardware to software. Such a transaction is triggered by the manager after around 60 seconds of the CPU core utilization at 100%. Here, the differences between the two approaches are highlighted and the OpenCL approach reduces the computation load since the




(a) (b)

Fig. 10: CPU core computation load for OFDM execution via *a*) SIMD (purely software) and *b*) via OpenCL offload

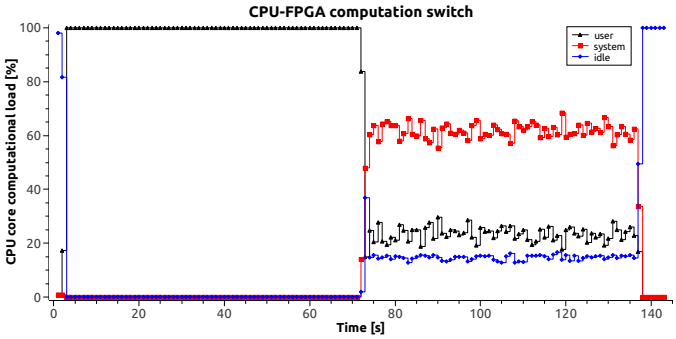



Fig. 11: CPU core computation load during the switch from software to the hardware

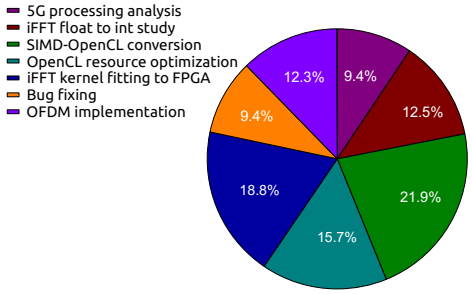



Fig. 12: Design efficiency and implementation chart

core remains in idle state for the 15% of the time. This means that the computational power of the CPU can be reduced by means of hardware offloading and part of the computation load can be reassigned to other tasks.

Finally, a consideration for the OFDM symbol size is necessary. This parameter evaluation is carried out with 256 and 512 OFDM symbol sizes and, in both scenarios, the results for SIMD and OpenCL are the same. Considering SIMD approach, OFDM computation always requires 100% of CPU processing. However, OpenCL approach requires the same CPU computation since all the processing is carried out in the hardware. Indeed, on the software side, the OpenCL application, that controls the FPGA execution, needs the same percentage of CPU resources in terms of both user state (i.e., OpenCL API application) and system state (i.e., interaction with PCIe) for all the OFDM symbol sizes.

## VII. Design productivity analysis with OpenCL acceleration

In this Section, the evaluation of the design productivity of deploying OpenCL kernel is addressed. According to [30], the performance of a HLS approach is a trade-off between a quality system optimization (e.g., resource usage, memory accesses, etc.) and design efficiency. In particular, design efficiency can be expressed with several indicators such as complexity of the design, the cost of the new code, the ”developer friendliness” of programming language, the designer experience and the testability of results. These indicators can be also used ”as is” for the HLD method considered in this work and they reflect into system integration time and design time. The first one concerns the verification and the validation of the the design and it depends on more electronic aspects like analog-to-digital and digital-to-analog conversions or electrostatic discharge that can damage the system. The design time refers to all the tasks necessary to create an application up to the deployment phase. This second parameter depends on the knowledge of the designer about both the system where the application has to be developed and the programming language and tools used to create the application. In this work, the time considered refers to a single developer with experience on the embedded systems but novice in the OpenCL programming model.

Figure 12 reports the percentage of overall time considered (3 months) to finish a precise task in the application development. The work starts with the analysis of the 5G DU processing under the option 7-1 functional split. In particular, the most significant contribution, in terms of complexity, is represented by the iFFT Cooley-Tukey algorithm. Thus, some examples of its implementation in both float and integer representation are considered before starting the implementation (this task requires 9.4% of the development time). Since many OpenCL iFFT float implementations are available off-the-shelf, the second step considers the adaptation of the float version in the

integer version of the iFFT. This step permits to understand if the off-the-shelf iFFT can be suitable for the OFDM stage in the 5G DU processing. It requires 12.5% of the time.
Unfortunately, such a conversion needs a lot of work due to the conversion of each float operation into an integer one and the computation of all the twiddle factors. Indeed, twiddle factors are calculated at runtime in the float version by means of sine and cosine functions. In the integer iFFT considered in the 5G architecture, twiddle factors have to be constant values to avoid FPGA resource overhead or they can be computed at runtime after an offline conversion of the functions sine and cosine into integer lookup tables. Thus, a better way to address the integer iFFT design is to consider an OpenCL kernel created from the SIMD implementation of the integer iFFT. The SIMD version of iFFT is already deployed in the 5G communication framework. The replacement of such a iFFT in the 5G application can be easier thanks to the excellent integration of the OpenCL with the software. The SIMD-iFFT conversion to an OpenCL kernel requires 21.9% of the time.
The next step is related to the optimizations of the iFFT kernel created in terms of area resources. Initially, the iFFT kernel does not fit into the hardware since we consider unrolled loops to improve processing time. Thus, many optimizations have to be performed, especially for the data dependency inside the loops (a complete explanation of the kernel optimizations is reported in Section V). This task takes 15.7% of the overall time.
The next two steps refers to the compilation of the kernel into the FPGA and some bug fixing. Despite these steps may seem faster than the rest of the tasks, they take 28.2% of the overall time due to the Intel FPGA Quartus tool compilation time. Kernel compilation requires from some minutes up to several hours for OpenCL execution file generation, thus, this time is significant it should be taken into account by the designers before starting an OpenCL application.
The last step considers the development of the cyclic prefix addition kernel to be combined with the iFFT kernel to complete the OFDM modulation at the 5G DU side. This task takes 12.3% of the overall time.
Finally, a brief discussion about the productivity design comparison between OpenCL and HDL method is necessary to understand the possible improvements in considering OpenCL for FPGA designs. By using OpenCL, as reported in Figure 12, a good balance between the different steps is obtained. As matter of example, we can consider the SIMD-OpenCL conversion, the OFDM implementation and the co-design implementation steps as an unique coding step which represents the 45% of the overall time. By cutting in half such time, the design time would not be reduced more than a quarter. Thus, the effort of each design phase is more relevant with respect to the obtained benefits. However, the coding time is anyway higher than the other steps due to the inexperience of the developer in both SIMD and OpenCL approaches. For this reason, a good estimation of the overall time to offload 5G OFDM in hardware via OpenCL could be around 2 months, when developed by an expert OpenCL designer. Regarding HDL, an estimate of the development time to complete the 5G OFDM hardware offloading is about twice as much as OpenCL development time, if we consider a novice HDL developer (i.e., 6 months versus 3 months in case of novice OpenCL developer). Indeed, the HDL languages are not programming languages like C/C++ or OpenCL itself, but description languages where timing constraints are entangled with functional constraints. As HDL acronym may suggest, a hardware language describes the FPGA architecture that must be synthesized and implemented by means of ad-hoc Integrated Development Environments (IDEs). Moreover, hardware programming involves concepts like parallel operations, waiting for clock edges, tri-state logic, hardware propagation delays and hierarchical structures that are not taken into account when software programming is considered. This converts design into a deep study of the hardware world that requires more time since the designer has to manually set the concrete hardware architectures.

## VIII. Conclusion

The goal of this paper was to evaluate the viability of OpenCL for implementing some functionalities of a virtualized next generation eNB. To perform such evaluation, the implementation of the OFDM in the 5G DU with Option 7-1 functional split has been considered in a hardware, software and co-design approach exploiting OpenCL model. The performance evaluation showed that OpenCL implementation is useful when virtualized functions shall be dynamically deployed in hardware or software for orchestration purposes. However, OpenCL shows some drawbacks with respect to the other considered methods. Thus, it requires a stronger optimizations and further studies. In particular in terms of resources, OFDM kernels with OFDM symbol size up to 512 can be deployed in hardware by means of OpenCL, but large FPGAs or more pipelined loops have to be considered over this threshold. Furthermore, the pure kernel processing time is compatible with 5G latency performance but memory transfers between host and accelerator are the bottleneck of the system when using PCIe. Around 233 µs have been measured, considering a CPU core dedicated to the processing and, likewise, around 468 µs have been measured in case of CPU core shared with other concurrent tasks. This behaviour is due to data synchronization from the host to FPGA in both directions, almost regardless of the message size. To overcome this drawback, the OpenCL kernels should be developed as auto-run to avoid the interaction with global memory and the PCIe data transfer, otherwise the overall execution time is not compatible with 5G processing. This solution will be further investigated as future work.
Only considering the pure processing time, OpenCL over hardware is more suitable for large data batch processing since results have shown that SIMD and HDL approaches have better performance up to 4096 OFDM symbol size. A model for SIMD and OpenCL performance curves has been evaluated to predict the processing time for bigger OFDM symbol size. Such models have shown that over 4096 OFDM symbol size the OpenCL implementation starts to show less excution time than both SIMD and HDL approaches. Thus, in case of a future implementation of a 8192 OFDM symbol size in the 5G PHY layer (the present implementation reaches 4096), the OpenCL

over hardware with current performance could be considered to implement the DU processing. Moreover, the results are strictly related to the devices and the technologies considered, thus the hardware performance for both OpenCL and HDL can significantly increase with the next FPGA technology.
Regarding the computational load, results have shown that the SIMD execution (purely software) require 100% of the computational load throughout the duration of the OFDM execution. On the other hand, with OpenCL offloading, the CPU core computation is limited to 85% and, for the 15% of the OFDM execution time, the core is in idle state. Moreover, the evidences between the two approaches are also highlighted by the performance measured with the manager developed for the dynamic hardware offloading. These scenarios have been used to demonstrate that the computational power of the CPU shall be reduced and part of the processing can be reassigned for another tasks to deploy more services.
Finally, an OpenCL design productivity evaluation has been reported to understand the efforts necessary to create and deploy the kernels. In particular, we highlighted the difficult steps that a new designer has to take into account for a new OpenCL design on FPGA, and the cost in terms of time to deploy an OpenCL kernel in a standard context.

## ACKNOWLEDGMENT

This work was supported by EC H2020 5GROWTH project (grant agreement no. 856709).

## REFERENCES

[1] Grant Martin and Gary Smith, "High-level synthesis: Past, present, and future," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 18–25, 2009.
[2] Martin Schoeberl, Stephan Korsholm, Tomas Kalibera, and Anders P Ravn, "A hardware abstraction layer in java," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 4, pp. 42, 2011.
[3] Donald Thompson and Colin Miller, "Introducing the .net micro framework," *Microsoft Corporation*, 2007.
[4] Scott Hauck and Andre DeHon, *Reconfigurable computing: the theory and practice of FPGA-based computation*, vol. 1, Elsevier, 2010.
[5] Sean Rul, Hans Vandierendonck, Joris D'Haene, and Koen De Bosschere, "An experimental study on performance portability of opencl kernels," in *2010 Symposium on Application Accelerators in High Performance Computing (SAAHPC'10)*, 2010.
[6] ITUT GSTR-TN5G, "Technical report transport network support of imt-2020/5g, feb 2018," .
[7] F Giannone, K Kondepu, Himank Gupta, F Civerchia, P Castoldi, Antony Franklin, and L Valcarenghi, "Impact of virtualisation technologies on virtualised ran midhaul latency budget: A quantitative experimental evaluation," *IEEE Communications Letters*, 2019.
[8] Sterling Perrin, "Evolving to an open c-ran architecture for 5g," *Fujitsu Heavy reading White Paper*, 2017.
[9] F Civerchia, F Giannone, and S Doddikrinda, "Encapsulation techniques and traffic characterisation of an ethernet-based 5g fronthaul," in *2018 20th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2018, pp. 1–5.
[10] 3GPP, "Study on New Radio Access Technology: Radio Access Architecture and Interfaces (Release 14)," TR 38.801, Mar. 2017.
[11] Navid Nikaein, Mahesh K Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet, "Openairinterface: A flexible platform for 5g research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, 2014.
[12] Recommendation CCITT, "Pulse code modulation (pcm) of voice frequencies," in *Blue Book*. ITU-T, 1988.
[13] Lee Howes and Aaftab Munshi, "The OpenCL Specification," Version 2.0, Khronos OpenCL Working Group, July 2015.
[14] Haomiao Wang, Prabu Thiagaraj, and Oliver Sinnen, "Fpga-based acceleration of ft convolution for pulsar search using opencl," *arXiv preprint arXiv:1805.12280*, 2018.
[15] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C Ling, and Gordon R Chiu, "An opencl deep learning accelerator on arria 10," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 55–64.
[16] Jialiang Zhang and Jing Li, "Improving the performance of opencl-based fpga accelerator for convolutional neural network," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 25–34.
[17] R Domingo, Rubén Salvador, Himar Fabelo, Daniel Madroñal, Samuel Ortega, Raquel Lazcano, Eduardo Juárez, G Callicó, and César Sanz, "High-level design using intel fpga opencl: A hyperspectral imaging spatial-spectral classifier," in *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2017, pp. 1–8.
[18] Ahmed Sanaullah and Martin C Herbordt, "Fpga hpc using opencl: Case study in 3d fft," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. ACM, 2018, p. 7.
[19] Intel Corporation, "Best Practices Guide," Version 18.1, Intel® FPGA SDK for OpenCL™, Sept. 2018.
[20] James W Cooley and John W Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.
[21] Patrick Marsch, Ömer Bulakci, Olav Queseth, and Mauro Boldi, *5G system design: architectural and functional considerations and long term research*, John Wiley & Sons, 2018.
[22] Erik Dahlman, Stefan Parkvall, and Johan Skold, *4G, LTE-advanced Pro and the Road to 5G*, Academic Press, 2016.
[23] David Demmer, Robin Gerzaguet, Jean-Baptiste Doré, and Didier Le Ruyet, "Analytical study of 5g nr embb co-existence," in *2018 25th International Conference on Telecommunications (ICT)*. IEEE, 2018, pp. 186–190.
[24] Intel Corporation, "Intel FFT IP Core User guide," Version 17.1, Intel® Corporation, June 2017.
[25] Haomiao Wang and Oliver Sinnen, "Fpga based acceleration of fdas module for pulsar search," in *2015 International Conference on Field Programmable Technology (FPT)*. IEEE, 2015, pp. 240–243.
[26] M Akif Özkan, Oliver Reiche, Frank Hannig, and Jürgen Teich, "Fpga-based accelerator design from a domain-specific language," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–9.
[27] Paul Rodríguez, "A radix-2 fft algorithm for modern single instruction multiple data (simd) architectures," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 2002, vol. 3, pp. III–3220.
[28] 3GPP, "5G, NR, Physical channels and modulation (Release 15)," TS 38.211, July 2018.
[29] Minh Mac and Chris Wysocki, "Guaranteeing silicon performance with fpga timing models," White paper, Intel® Corporation, Aug. 2010.
[30] Maxime Pelcat, Cédric Bourrasset, Luca Maggiani, and François Berry, "Design productivity of a high level synthesis compiler versus hdl," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. IEEE, 2016, pp. 140–147.