Towards sharing one FPGA SoC for both low-level PHY and high-level AI/ML computing at the edge

Ioannis Stratakos*, Elissaios Alexios Papatheofanous†, Dimitrios Danopoulos*,
George Lentaris*, Dionysios Reisis†, Dimitrios Soudris*
*School of Electrical and Computer Engineering, National Technical University of Athens, Greece
{istratak, dimdano, glentaris, dsoudris}@microlab.ntua.gr
†Electronics Lab, Physics Dpt, National and Kapodistrian University of Athens, Greece
{eapapatheo, dreisis}@phys.uoa.gr

***Abstract*—Beyond 5G networks are expected to distribute the computations from the cloud to multiple edge nodes, some of which should process both low-level baseband functions and high-level tasks, most notably AI/ML. These edge computing nodes will demand high-performance, re-programmability, and low-power, especially when located at the far-edges of the network. An attractive solution in meeting these needs is to utilize highly-complex SoC FPGAs, such as the state-of-the-art RFSoC or ACAP devices. As proposed in this work towards minimizing cost and power consumption of nodes, sophisticated SoC programming will enables us to execute in parallel the baseband and AI processes, i.e., to exploit a single device as an accelerator for the full range of tasks executed on a network node. The current paper explores how to integrate RFSoC/ACAP in such architectures, including the high throughput interfaces, the HW/SW co-processing capabilities, the accelerators' deployment, and a preliminary estimation of performance/utilization.**

***Index Terms*—BBU, AI/ML, RFSoC FPGA**

I. Introduction

The upcoming generations of mobile communications have to support an increased number of connected devices as well as novel and/or enhanced high demanding services, while keep the Quality-of-Service (QoS) high. Consequently, the emerging telecommunication technologies have to provide increased signal bandwidth and very low communication latency and to support the beyond 5G applications have to include edge compute nodes that are high-performance, low-power, and flexible/reprogrammable, all at the same time. Moreover, these multi-layer network architectures and the remote edge computing nodes impose the need for processors that are powerful and efficient in executing both low- and high-level functions, e.g., telecom DSP pipelines and AI/ML.

In these novel network generations, a key element that has to meet all the aforementioned needs and requirements is the Baseband Unit (BBU) with a major role of (de)modulating the transmitting/receiving data. To tackle the challenge of designing a BBU meeting the above demands and specifications, this work proposes implementing concurrently such diverse functions on a single COTS device by exploiting the novel capabilities of very complex SoC FPGA, i.e., RFSoC and Versal ACAP. We explore the possibility of dividing logically the SoC in two parts: the low-level telecom implementation and the high-level AI/ML acceleration. The former will utilize hard-IPs for analogue and digital interfaces towards bridging the radio and core parts of the network, as well as CLBs for the radio signal processing. The latter will rely on CPU cores and the remaining CLBs to perform HW/SW co-processing of the AI/ML algorithms. In this work, primarily, we evaluate the overhead of implementing the various interfaces and assess the FPGA space remaining for our high-performance processing; secondarily, we estimate the achievable throughput of real-time DSP telecom chains and representative AI accelerators when placed together in the FPGA. Overall, we explore the trade-offs and programming approaches while tuning function resources and changing architectural configurations.

Work partially supported by EU project AI@EDGE (grant agr. 101015922)

The paper is organized as follows: Section II gives our view for concurrently hosting low-level PHY DSP and AI/ML applications on the same SoC FPGA at the edge and presents key features of these devices. Section III presents an initial implementation of key components for the low-level PHY DSP, while Section IV gives relevant informations for the AI/ML application. Next, in Section V an initial evaluation is given in terms of FPGA resources and performance. Finally, Section VI concludes this paper.

II. Proposed Architecture

The implementation proposed in this paper considers a single FPGA SoC divided in two logical parts, one for low-level real-time DSP functions and one for off-line best-effort AI acceleration. Fig. 1 shows an indicative partitioning of Xilinx RFSoC. The hard-IPs for DAC/ADC and FEC are assigned to partition L, whereas the majority of CPU cores/peripherals are assigned to partition H (one core is still reserved for monitoring/controlling L and implementing auxiliary telecom functions, e.g., the EVM calculation). The FPGA's programmable logic (PL) is shared among L and H partitions, such that L can complete the signal modulation/demodulation in the given deadlines, while H can exploit the remaining PL to maximize the speedup of the given SW functions executing on PS. The L partition will utilize the PL to implement soft-IPs for digital interfaces (25G Ethernet, AXI stream, etc) and VHDL components for DSP (QAM, FFT, etc). The H partition will utilize the PL to implement AI functions (convolutions, activations, etc). The exact ratio

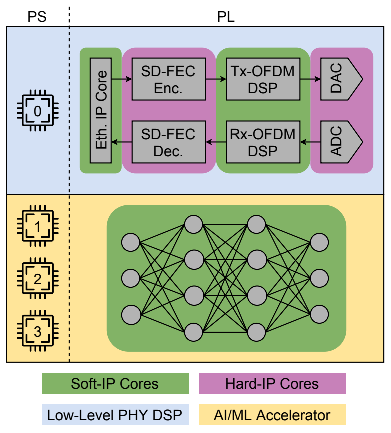


Fig. 1: SoC FPGA partitioning to accommodate low-level PHY processing and AI acceleration with HW/SW co-processing.

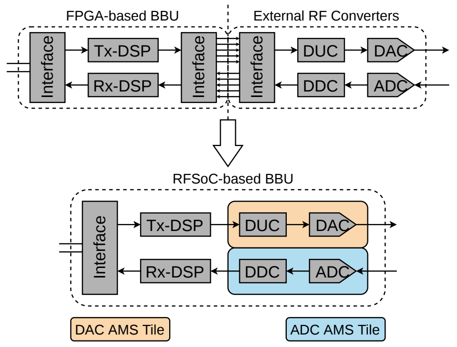


Fig. 2: Evolution of FPGA-based BBUs.

of L to H resources is application dependent: higher telecom throughput specifications will lead to lower AI acceleration. For evaluation purposes in this exploration study, we assume telecom bitrates in the area of 5Gbps (e.g., up to 3Gbps useful data or 6Gbps OFDM data over the channel).

When assuming a device offering different hard-IPs, i.e., AI engines instead of FEC, as in the case of Xilinx Versal (ACAP), then the proposed partitioning changes to accommodate new soft-IPs on the PL side. The aforementioned AI functions will move from the PL to the hard-IP engines of the SoC, and hence, will leave space in the PL to implement soft-IPs for FEC. The trade-off between hard- and soft-IPs is also application dependent: lower RF specifications decrease the need to use hard-IPs for telecom and allow us to increase the performance of higher-level functions via hard-IPs for AI/ML and multi-Gbps Ethernet (to exchange more data with the core network instead of the RF domain).

The remainder of this section discusses those distinctive features of RFSoC, which enable an efficient solution for the BBU implementation, together with AI/ML, at the far edge.

A. Rationale of selecting RFSoC

Until the widespread adoption of 5G standard, the devices must have backward compatibility with the already existing standards (e.g. 4G LTE). FPGAs are attractive for implementing BBUs that support a variety of standards. Until recently FPGAs were used only for basedband signal processing and communication with the core network. Xilinx introduced the Zynq Ultrascale+ RFSoC family of devices, that integrates on the same die an heterogeneous processing platform (CPUs, FPGA fabric, etc.) with direct RF-sampling converters (Fig. 2).

The integration of DACs/ADCs on the same die tackles many design challenges. The interface between the FPGA and the converters becomes significantly simpler. Previous systems had to implement specific protocols for communication (e.g JESD204), that led to significant portion of the FPGA resources used for this purpose. Moreover, the external I/O interfaces consumed significant power to operate. Now, RFSoC devices [1] offer 1) reduced power consumption, 2) reduced clock/data routing complexity on PCB 3) simplified synchronization procedures between FPGAs and RF-converters and 4) the full flexibility of traditional FPGAs.

B. Integrated DACs/ADCs

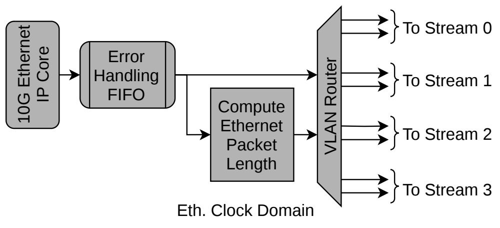
The RFSoC device integrates Analog-Mixed-Signal (AMS) components, i.e., on the same die, it hosts up to 16 Digital-to-Analog (DAC) and 16 Analog-to-Digital (ADC) converters depending on the device generation. The converters have analog bandwidth in the order of GHz. Moreover, each converter has each own dedicated digital datapath that implements some DSP functionalities previously implemented on the FPGA fabric. These functionalities include interpolation/decimation filters, digital mixers to up-/down-convert from/to baseband, signal compensation blocks, as well as signal detection capabilities. Moreover, the programmability of the AMS blocks offers 1) different number of parallel samples that ease the operating frequency on the FPGA, 2) ability to process real or complex signals and 3) multi-band operation.

C. Ethernet Connectivity

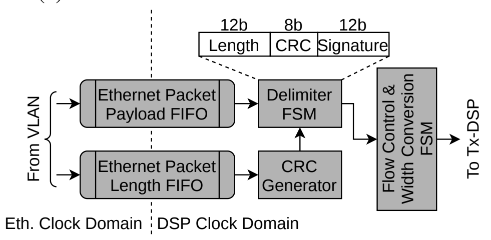
RFSoC has increased capabilities in terms of Gigabit Ethernet connectivity enabling the Ethernet-to-RF applications. On the PS side the designer can opt to use the equipment of the single Quad of PS-GTR transceivers as well as the four triple speed Ethernet MACs that provide support for IEEE Std 802.3 with 10/100/1000Mbps rates and a set of capabilities enabling the VLAN tagging, Precision Time Protocol (PTP) support, etc. More importantly, on the PL side, each of the GTY transceivers can support up to 25G Ethernet with the MAC and PCS implemented on the FPGA fabric. 100G Ethernet can also be supported with the Integrated Block for 100G Ethernet that includes both MAC and PCS logic. The above features/equipment/IPs can enable applications where connectivity to RRHs or network backhaul is required.

D. Embedded Components (hard-IPs)

Next to the FPGA fabric, RFSoC integrates a quad-core Cortex-A53 APU and a dual-core Cortex-R5 RPU. Utilizing the embedded CPUs allow real-time monitoring and control of the BBU operation. Automatic calibration of the DSP



(a) VLAN-based ethernet traffic scheduler.



(b) Ethernet packet protection and control flow.

Fig. 3: L2 architecture for Ethernet to Tx-DSP mapping.

functions can be accomplished through data monitoring and computation of link statistics (e.g EVM, noise variance, etc.). The user will also be able to connect remotely to the device to perform parameterization of the BBU, debugging in case of error reporting, perform maintenance and update supported functionalities. A single CPU core is sufficient to perform these tasks. The rest of the CPU cores can be used to coordinate the proposed AI/ML partition of the RFSoC FPGA.

RFSoC integrates Soft-Decision FEC (SD-FEC) hard-IP blocks. The FEC algorithms are very computationally intensive and would require huge amounts of resources (BRAMs, DSPs) on the programmable FPGA fabric (PL), especially for decoding, as well as increased dynamic power consumption. Certain RFSoC devices integrate up to 8 such SD-FEC blocks with their own dedicated clock network (not limited by the remaining DSP chain). These blocks are configurable and can support different coding schemes, such as Turbo and LDPC. Pre-configured standards are supported (5G NR, LTE, 802.11, DOCSIS 3.1), as well as custom LDPC codes can be used.

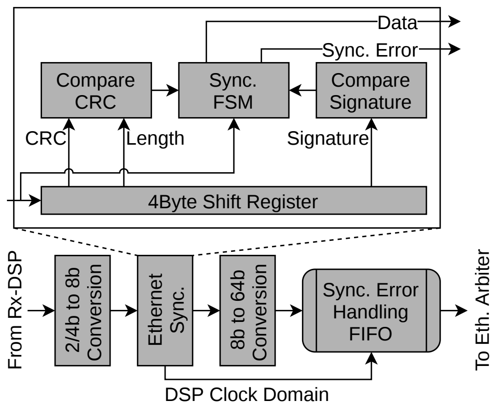
III. Proposed BBU Functionality on RFSoC

A. *Ethernet connectivity*

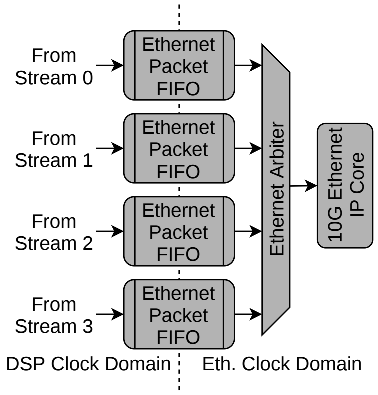
For the current utilization of the RFSoC as a BBU, the Ethernet interfacing is realized using the 10G/25G High Speed Ethernet Subsystem Xilinx IP Core, configured for 10GE. In order to map incoming Ethernet traffic to the DSP chains of the baseband processing and also recover Ethernet frames from demodulated OFDM symbols, we design and develop custom L2 functionalities in VHDL. Our VHDL includes basic error handling, flow control, frame encapsulation and allocation of DSP resources based on VLAN tags, all programmable at runtime between QPSK and QAM16.

The architecture of the L2 functionalities for the Ethernet receive (or RF transmit) side is depicted in Fig. 3. First, the incoming Ethernet packet is checked for errors and it will be dropped if the 10GE IP Core reports errors. Then the VLAN Route components routes the packet according to its VLAN tag to one of the 4 corresponding DSP chains that the architecture was designed to include (Fig. 3a). We note here that the VLAN tagging is part of the SDN functionality of bandwidth allocation and it is implemented by a network controller external to the FPGA. The length of the above incoming packet is calculated on the fly and both the packet and its length value are buffered to FIFOs performing CDC. The Delimiter FSM is responsible for appending the delimiter of Fig. 3b as a header to the Ethernet packet. The design choice for the delimiter is 32-bits wide to induce the minimum overhead possible to the air transmission while still maintaining an increased chance for the Ethernet packet to be correctly recovered at the RF receive side. It consists of a static 12-bit signature field with a hard coded value as well as a 12-bit length field indicating the Ethernet packet's length. The length value is protected by a CRC8 field in order to minimize the chances of erroneous detection of the delimiter at the opposite side. Finally, the Flow Control FSM converts the incoming 64-bit words into either 2-bit or 4-bit words based on the modulation order (QPSK/QAM16) and forwards them to the physical layer. In the absence of Ethernet traffic, the Flow Control FSM will indicate this input status to the L2 of the RF receive side by inserting a null delimiter with a length value of zero. In this case, idle symbols which are defined as x00, will be forwarded to the physical layer following the null delimiter until a new Ethernet packet shows up.

The L2 functionalities architecture for the RF receive (or Ethernet transmit) side is shown in Fig. 4. Depending on the modulation order, 4-bit or 2-bit values are concatenated to bytes and forwarded to the Ethernet Synchronization component. This component is responsible for performing synchronization of Ethernet frames by correctly recovering them based on the received 32-bit delimiter. Its internal structure is also depicted in Fig. 4a Whenever a new byte is received a new 32-bit word is formed by means of a 4-byte shifting register. The 32-bit word is then processed based on the expected delimiter structure, the CRC8 value of the length field is calculated and compared with the received CRC8 and the expected signature value is compared to the received corresponding field, all concurrently. The outputs of the comparison are forwarded to the Synchronization FSM, which decides whether a null or actual delimiter is received at the expected time (after the end of the previous frame). If a delimiter is detected at an incorrect time, either while recovering a previous frame or not exactly after it has been recovered, the Synchronization FSM will signal the Error Handling FIFO to drop the frame at this FIFO's output. Such errors during recovery stop frames from being forwarded to the upper layers. The recovered packets are stored in a CDC FIFO ready to be transmitted via Ethernet. Finally, the Ethernet Arbiter component aggregates traffic from the 4 independent DSP chains one-by-one and in a round robin fashion. Upon availability, traffic is forwarded as Ethernet packets to the 10GE IP Core.



(a) Ethernet frame synchronization.



(b) Ethernet traffic arbiter.

Fig. 4: L2 architecture for Rx-DSP to Ethernet mapping.

B. Analog-Mixed-Signal Functionalities

RFSoC AMS blocks consist of two main parts: a) the digital datapath and b) the actual RF-converters. The digital datapath is able to perform different DSP functions that used to be implemented on the FPGA fabric or by external components.

First of all, in the boundary between the FPGA fabric and the AMS blocks exist configurable CDC FIFOs. These FIFOs regulate the data traffic and provide a flexible interface from/to custom DSP function on the FPGA fabric. The next sub-components, implemented on the digital path of AMS blocks, are the interpolation/decimation filters. The number of filters enabled is configurable and offer interpolation/decimation of signals up to x8 for the 1^{st} and 2^{nd} generation of RFSoCs, while the 3^{rd} generation further expands this capability with up to x40 factors. However, the filter coefficients are not programmable. Next, are the Digital Complex Mixers (DCM), which support real or complex output signals. The DCMs support three modes of operation: a) no mixing, b) coarse and c) fine. When configured to do coarse mixing only a few number of Intermediate Frequencies (IFs) are supported, which are submultiples of the sampling frequency. When doing fine mixing, the user is able to program the intended IF to the embedded Numeric Control Oscillator (NCO) of the DCM during the design phase or even create appropriate logic (hardware or software) to adapt the IF in real-time based on requirements at the moment. Another, common sub-component of the AMS blocks, is the Quadrature Modulator Correction (QMC) block. When the RFSoC are interfaced with external analog quadrature mixer devices, due to unpredictable events, imbalances to the RF signals occur. These imbalances can lead to performance degradation of the BBU. The QMC blocks offer a convenient way to correct these imbalances, but the process of detecting the imbalances is still left to the designer, who must design the algorithm. One more feature of both DACs and ADCs in RFSoC is their capabilities on generating multi-band signal. Specifically, multiple digital datapaths can be combined and drive a single DAC for generating multi-band signals. Accordingly, the ADCs can distribute a multi-band signal to a different number of digital datapaths able to process the data of a single band. But, as with the previous blocks, there some limitations. When processing real signals up to four band can be combined, while for complex signals up to 2. The DSP functions on the FPGA process two data streams independently producing a complex signal and feed two distinct AMS blocks. After interpolation and up-conversion to their respective IF (Digital Up-Conversion - DUC), the two bands are combined and drive a single DAC. The DAC of the AMS block not used is powered down in this operation mode leading to reduce power consumption. Until now, the blocks described are common for both DACs and ADCs. However, there are some features distinct to DAC and ADC AMS blocks. The ADCs have built-in signal detection circuit that is able to report the strength of the received signal. The primary use of such feature is the realization of an Automatic Gain Control (AGC) mechanism. AGC is used so that the full dynamic input range of an ADC is used, by responding to varying signal amplitudes. The RFSoC devices provide only an indication of the signal strength and is left to the system designer to implement the appropriate algorithm to compute the required gain and apply the correction. For the DACs, there exists another filter before the digital to analog conversion implementing an Inverse Sinc operation. Thus, the DACs are able to have a flat response in a wider bandwidth.

IV. Proposed AI Acceleration on RFSoC

Hardware devices such as FPGAs have been introduced to tackle the high computational demands in the edge domain where power efficiency and low latency poses a critical issue. Our representative AI solution relies both on the PS and PL side of the SoC. The FPGA communicates through PCIe to feed the data under a 16-lane endpoint configuration. This section will describe the varying levels of abstraction available for FPGA design regarding AI applications, from the hardware description languages (HDL) that operate on the circuit level and demand RTL expertise to the generalized computation engines that can be application agnostic and require little programming effort.

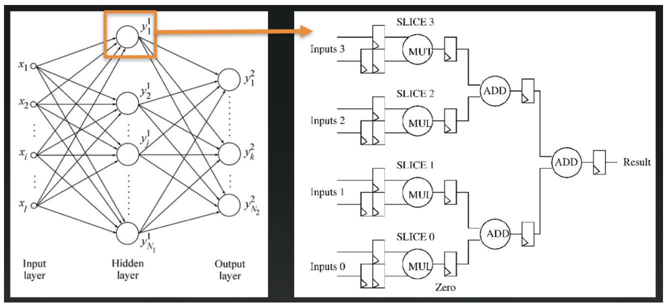


Fig. 5: An abstract model of an FPGA implementation in circuit level derived from a neural network topology [5]

A. HDL design

Register-transfer-level (RTL) abstraction is used in HDL languages (i.e. VHDL or Verilog) to create low-level representations of an AI model, from which a mixture of registers and boolean equations is derived. Unlike in software compiler design, RTL takes as input the register transfer level representation and involves constructs such as cells, functions, and multi-bit registers. In order to accelerate an AI algorithm on hardware several Processing Elements (PEs) that perform multiply–accumulate operations are hand-written in Verilog or VHDL code that are designed and optimized based on the constraints of the target platform (i.e. the number of multipliers, etc). Mapping an AI model's computational operations to matrix-matrix or matrix-vector multiplication modules has been widely applied in prior studies [2]–[4]. Also, usually tiling and ping-pong double buffers techniques are then employed to improve the throughput along with other optimizations on the RTL level. Thus, this method can make the design of an AI application strenuous as opposed to software programming but the high reconfigurability of HDL can often be a key requirement especially in the AI market which is constantly evolving.

B. HLS design

Employing High Level Synthesis for FPGA design enables a fast development process and high flexibility. Although using HLS provides a software-like tool flow (i.e. Xilinx Vitis or Intel Quartus Prime), the developer must still learn hardware-centric concepts, such as pipelining and routing, that they may have not been exposed to in writing C-code for traditional processors. Also, studies have shown that HLS can simulate faster the effects of data type precision and other hardware approximations which are often utilized on AI applications [6], [7]. HLS automatically generates an RTL testbench which is driven by vectors generated by the original C++ code which is essential for simulating AI inference with bit-accurate precision. Also, in the case of Neural Networks, an HLS compiler can synthesize each layer as a separate module where every layer usually feeds its output to the next in a dataflow architecture with streaming transfers [8]. Additionally, each layer's precision can be controlled seamlessly by changing the weights or biases bitwidth using the HLS fixed-point arbitrary precision data types. Last, previous work has shown that HLS designs can exploit significant parallelism compared with RTL approaches and achieve similar latency improvements [9].

C. Dedicated DPU kernels

Responding to the extreme demand by the new emerging AI algorithms, HW companies introduced AI specific engines which have higher compute density and lower power requirements than processing on their traditional processing elements. For example, Xilinx has developed Vitis AI which is a complete development stack for AI inference targeting Xilinx hardware platforms, including both edge and cloud devices. Vitis AI uses an optimized IP named DPU (Deep Learning Processing Unit) which is a programmable engine optimized for convolutional neural networks. The Vitis AI tool converts the AI models into a graph-based intermediate representation which is then compiled for the DPU IP by generating the instruction stream along with coordinating data transfers.

V. Preliminary Evaluation

This section presents the software and hardware environments used to perform exploration and testing. FPGA resource utilization is estimated for both low-level PHY DSP and AI/ML accelerator, as well as their performance. Additionally, we performed an exploration of different AI/ML and low-level DSP configurations to determine the feasibility of our proposed architecture. Our target platform is ZCU111 evaluation board hosting XCZU28DR RF-SoC FPGA.

A. Software, Device setup, Benchmarks

The low-level PHY DSP evaluation we performed on Xilinx Vivado Design Suite with VHDL coding and IP cores, while for the AI/ML we focused on evaluating several DNNs using Xilinx Vitis AI development stack which is developed specifically for AI inference on Xilinx FPGAs. This framework consists of optimized IP, tools, libraries and example designs that supported many DNN topologies that we tested which were trained in popular Deep Learning frameworks such as Tensorflow or Pytorch. The models were also quantized into 8-bit precision using the Vitis AI quantizer and were compiled for the Xilinx DPU-B4096 architecture.

For AI/ML we utilized 1–3 B4096-DPU IPs for our exploration purposes. This configuration achieves 4096 operations/per clock for each DPU. The achieved frequency of the DPU IP was 300 MHz which translated to 2.56 TOPs of achievable performance. To use the DPU, we prepared the instructions and input image data in the specific memory address that DPU can access. The DPU operation also required the application processing unit (APU) to service interrupts to coordinate data transfer.

For the low-level PHY benchmarking, we assumed a Tx-Rx chain relying on OFDM and QAM. We estimated the FPGA resources by considering representative DSP function implementations and including margins based on various configuration/optimization levels. Next to the DSP chain, we used the hard-IP FEC encoder/decoders. We assumed datapath

TABLE I: Accuracy and performance benchmark accross different DNNs and datasets

	Model specifications					FPGA results		
Category	***Model***	***GOPs***	***Img Size***	***Dataset***	***Float acc.***	***DPU acc.***	***Latency***	***FPS***
Image Classification	ResNet-50v.1	6.97	224x224	ImageNet	75.2%	74.8%	12ms	158
Image Classification	Mobilenet_v2	0.6	224x224	ImageNet	70.1%	67.7%	4ms	575.5
Object Detection	yolov3_voc	65.6	416x416	voc07_test	78.5%	77.4%	72.7ms	27.6
Semantic Segmentation	ERFNet	54	512x1024	Cityscapes	53%	51.7%	145ms	23.2

TABLE II: Resource utilization on XCZU28DR[1] RFSoC

Name	*FF*	*LUT*	*BRAM*	*DSP*
B4096-DPU	105008	53540	257	562
Tx-Rx DSP chain	50K–100K	10K–20K	6–14	180–370
PL PCIe	20201	8183	22	0
10G Eth. Core	22004	51929	3	0
10G L2 Tx+Rx	912	1581	8	0
Misc. (AXI, etc.)	7156	9599	13	0

[1] Total resources: FF = 850K, LUT = 425K, BRAM = 1K, DSP = 4.2K

TABLE III: Performance-Utilization trade-off on XCZU28DR

Configuration	Throughput		Resources		
	Gbps	TOP/s	LUT	BRAM	DSP
A+4·B+E	2.5	0.5	39%–48%	30%–33%	30%–48%
A+4·B+4·F+E	2.5	0.5	51%–61%	30%–33%	78%–96%
2·A+4·B+E	2.5	1.12	51%–61%	54%–57%	43%–61%
2·A+4·B+2·F+E	2.5	1.12	58%–67%	54%–57%	67%–85%
2·A+2·B+2·F+E	1.25	1.12	53%–58%	53%–55%	59%–68%
3·A+2·B+E	1.25	1.34	59%–64%	77%–78%	48%–57%
3·A+2·B+2·F+E	1.25	1.34	65%–70%	77%–78%	72%–81%

A=DPU, B=Tx+Rx DSP, F=FIR, E={10G Eth., L2 Tx+Rx, PL PCIe, Misc.}

pipelining and parallelization (multiple chains) to increase the overall telecom throughput. The operating frequencies of these configurations varied in the area of 250MHz.

B. Results

First, we measured the accuracy and performance of distinct neural networks for distinct application domains. Table I summarizes the results obtained with Vitis AI 1.3 using the B4096-DPU configuration along with INT8 bit quantization on DNNs trained on Tensorflow. The top-1 accuracy for float and quantized DPU models (or mAP for object detection models) is also reported in addition to the performance metrics of latency and Frames per Sec (FPS).

Table II gives the resource utilization of the basic parts of our proposed architecture. The AI/ML partition and part of the low-level PHY DSP (Ethernet core, L2 Tx/Rx) partition and support logic for both partitions utilize around 19% FFs, 30% LUTs, 28% BRAMs and 13% DSPs. The resources used to implement one actual Tx-Rx DSP chain (row 3) vary due to our aforementioned estimation approach covering a range of possible implementations.

Lastly, we consider various combinations of the two main functionalities implemented on a single RFSoC device to explore the capabilities of our proposed architecture based on performance metrics and feasible utilization ratios for FFs/LUT/BRAM/DSP. Table III provides the utilization and the useful data rates of the most representative configurations. The more telecom throughput is required, the more low-level PHY DSP functionality will be implemented and the less AI/ML (eg. Table III rows 3, 4). On the other hand if more AI/ML acceleration is needed, more DPUs will be used and less low-level PHY DSP. Either case, from the table we observe that the resource consumption constrains are met. We note that the use of FIR filters for the low-level PHY DSP is application-specific and is not considered for all DSP chains in Table III (also, in certain cases/scenarios, they could be shared among chains). The most challenging configurations are those utilizing 3 DPUs and 4 low-level PHY DSP chains, together with the whole support logic and with/without the FIR filters, which start pushing our target device to its limits. Even so, with more aggressive optimizations during the design phase, all resource requirements of Table III can be met on RFSoC.

VI. Conclusions

This paper presented a system architecture, based on state-of-the-art SoC FPGA device, to accommodate low-level DSP functions required in a BBU along with AI/ML acceleration on the same device. Our initial evaluation shows that such combination is feasible in terms of resource consumption and performance metrics. Thus, future edge devices will benefit from sharing a common device to accelerate diverse functionalities from low-level DSP to high-level AI/ML.

References

[1] Xilinx, "White Paper:An Adaptable Direct RF-Sampling Solution," Tech. Rep., February 20 2019.

[2] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775, 2018.

[3] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 152–159.

[4] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto customized hardware," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 24–29.

[5] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. Berlin, Heidelberg: Springer-Verlag, 2006.

[6] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[7] M. Wess, S. M. P D, and A. Jantsch, "Neural network based ecg anomaly detection on fpga and trade-off analysis," 05 2017, pp. 1–4.

[8] D. Danopoulos, C. Kachris, and D. Soudris, "Utilizing cloud fpgas towards the open neural network standard," *Sustainable Computing: Informatics and Systems*, vol. 30, p. 100520, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537921000135

[9] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, pp. 1–1, 12 2015.