Zero-Copy AI-augmented Signal Processing Pipeline on Heterogeneous Satellite Processors

Michael Petry*,†, *Student Member, IEEE,* Andreas Koch*,†, *Student Member, IEEE,* Martin Werner†
*Airbus Defence and Space GmbH, †Technical University of Munich

***Abstract*—We present a novel signal processing concept that natively integrates artificial intelligence (AI) processing capabilities into satellite platforms. The processing workflow on high-end, resource-constrained platforms is typically Field Programmable Gate Array (FPGA)-dominated which renders the efficient integration of Machine Learning-based components a challenge. We address this by leveraging a novel System-on-Chip (SoC)-based heterogeneous compute platform and propose a deployment concept that allows for a flexible and dynamic arrangement of conventional and AI-based processing steps. After a comprehensive review of the hardware components we identify efficient inter-component data streaming as the primary implementation challenge, which we address using a zero-copy strategy. The complete deployment process is outlined for an AI-augmented telecommunications processing scenario. A succinct performance analysis of the critical inter-component data flow concludes this work, identifying potential bottlenecks and discussing avenues for future enhancements.**

***Index Terms*—Satellite processing, artificial intelligence, heterogeneous computing, zero-copy, wireless communication**

I. Introduction

Artificial intelligence (AI) has become ubiquitous in terrestrial applications, offering innovative solutions for rapid and efficient problem solving across various domains, including image and speech recognition, behavioral observations, autonomous reactions, and many more. While AI has seen widespread adoption on earth, its potential for space-bound systems, particularly satellites, has just recently begun to receive significant attention [1].

Possible benefits of on-board AI affect the whole ecosystem within and ”outside” of satellites. This includes removing the necessity of human-based intervention in spacecraft control by utilizing self-supervision and -recovery mechanisms through AI-powered behavior analysis [2], alleviating bandwidth-intensive raw data transfers for computer vision-based earth observation (EO) applications by processing data directly on-board [3], and increasing overall system performance by exploiting the strengths of Machine Learning (ML) in digital signal processing (DSP)-intensive tasks, such as telecommunications [4]. Ultimately, AI can completely transform how satellites interact with terrestrial systems, transforming them from mere data collectors with relay functionality to autonomous datacenters in space.

This work is supported by the German Aerospace Center (DLR) under project Machine Learning on Telecommunications Satellites (MaLeTeSa) with Grant number 50 YB 2103. Corresponding author: Michael Petry (e-mail: michael.petry@airbus.com).

Key enabler is the integration of generic AI-centric processing capabilities into the satellite platform and ensuring a native integration into the conventional processing flow. While terrestrial systems benefit from a wide landscape of specialized hardware, such as Graphics Processing Units (GPU), Tensor Processing Units (TPU), Digital Signal Processors, etc., which can efficiently work in concert due to standardized communication protocols and drivers, space-bound solutions lack this diversity. Amongst multiple restrictions, such as the limitation in available hardware technology due to harmful radiation, a tight power budget due to solar energy sources as well as thermal management, or weak general computational power due to heritage-proofed processors, the major challenge is the extreme reliance on Field Programmable Gate Array (FPGA)-devices, especially for high-performance satellites, which renders the seamless integration of AI into the processing chain cumbersome.

This challenge has initiated a race amongst both established companies as well as newly formed start-ups. Amongst many proposed solutions, a straightforward approach is to utilize standalone modules designed for AI computation, such as NVIDIA’s embedded GPU module series [5] and Intel’s Movidius™ Myriad™ X Vision Processing Unit, or neuromorphic architectures, such as Brainchip’s akida IP [6] and Intel’s Loihi chip [7], accepting the penalty of missing radiation compliance and its associated problems of runtime faults and a shortened device lifespan. A different approach is the concept of General Purpose Computing on Graphics Processing Units (GPGPU) using FPGA-accelerated soft-GPU Intellectual Property (IP)-cores, effectively realizing a GPU on the FPGA [8]. Furthermore, purely software-based solutions that target execution on Central Processing Units (CPU) and microcontrollers are in active development, such as Klepsydra AI’s high efficiency inference framework [9].

However, the prevailing approach, among most solutions, is to treat the AI application as a separate entity by either managing it as an isolated software application or separating it into a distinct hardware component. The lack of native integration and the resulting suboptimal interfaces impede the effectiveness for high-performance applications.

This paper takes a different approach by integrating, and more importantly, interconnecting AI capabilities within the heart of the satellite. Its main contributions are the following: Based on a detailed component-level analysis of our next-generation satellite processor, we derive a deployment strategy that prioritizes efficient data flow and effective hardware utilization based on a future reference scenario from the telecommunications domain, proposed by Petry *et al* [10]. To highlight relationships between both works, a coherent color-scheme is used throughout this work.

The rest of this paper is structured as follows: Sec. II introduces the telecommunications satellite processor design and reviews the reference scenario to be implemented. Sec. III proposes a deployment concept after analyzing the relevant hardware resources and their interconnectivity capabilities and describes the mapping procedure from algorithm to hardware. A brief performance analysis of the data flow is performed in Sec. IV, followed by an analysis of the system's behaviour and a discussion of possible improvements. Lastly, Sec. V concludes this work. This is followed by an outlook on the remaining challenges towards achieving powerful AI integration.

II. AI-Augmentation of the Telecom Processor

In this section we describe the processor design and its main components towards AI integration, and introduce the reference scenario which is later implemented on this processor.

A. Overview Telecom Processor Architecture

The next-generation telecommunications processor architecture, developed as part of the European Space Agency (ESA)-funded Theia project, consists of multiple functional components on dedicated hardware boards, also known as slices, interconnected by space-grade communication links such as Space-Wire and optical links. Fig. 1 provides an overview of these slices. Among a control unit that supervises the spacecraft's health, and a stand-alone FPGA that enables in-flight re-programmable classic signal processing capabilities, the key slices w.r.t. telecommunications-related applications are the RF frontend, the signal regeneration (Re-Gen), and the AI-& DSP-Accelerator slices. While the RF slice typically functions as simple interface to the radio spectrum (although comprising certain integrated static processing features), the Re-Gen slice offers application-specific integrated circuit (ASIC)-based processing capabilities, such as protocol-specific (de-)coding and (de-)modulation. The main innovation is the first-of-its-kind AI- & DSP-Accelerator slice built around AMD-Xilinx's Versal series [11], as explained in subsection III-B. This slice is designed to serve as the main workhorse for tightly integrated AI-augmented processing scenarios.

B. Reference Scenario: AI-augmented Communications System with Deep Learning-based Signal Synchronization

Machine Learning (ML) has expanded into various aspects of wireless communication, including Integrated Sensing and Communications (ISAC) [12], differentiable Ray-Tracing-based digital-twin networks [13], [14], physical layer transceiver design [15]–[19], and more. The telecommunication satellite industry has recognized the potential of ML in its DSP-heavy applications, including beamforming and beam management, network orchestration and inter-satellite routing,

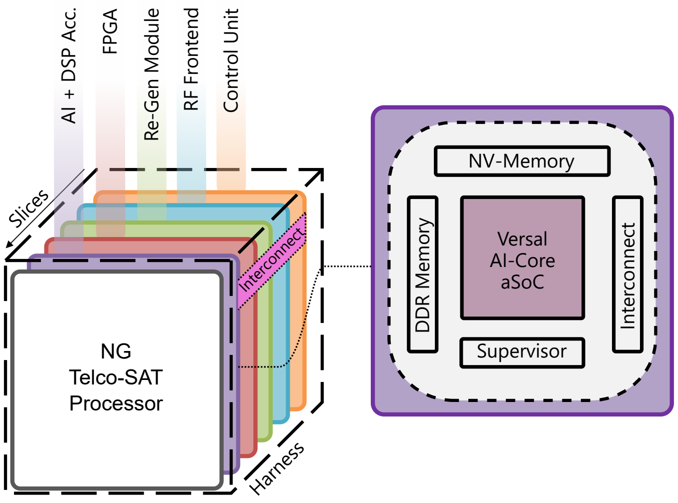

Fig. 1. Overview of the telecommunications processor design and its components. Detailed look into the AI- & DSP-Accelerator slice.

and radio access network (RAN) processing. Additionally, it has acknowledged the necessity of ML in future communication networks, such as 5G Non-Terrestrial-Network (NTN), as indicated by most 3GPP study items [20], [21], on satellite functionality.

To illustrate the presented deployment concept we select a reference scenario from the telecommunications domain that represents the key concept of future AI-augmented processing chains well: Combining conventional (classical) and AI-based processing blocks in a hybrid processing strategy. We realize a novel AI-augmented RF transceiver design from the receiver's perspective. The receiver integrates an auto-regressive signal synchronization procedure that estimates and compensates for coarse and fine center frequency offset (CFO) and sample time offset (STO) solely from the received IQ-samples. The "estimate-and-transform" strategy re-uses the concept of a Radio Transformer Network [22], utilizing mostly ML-based blocks to estimate certain signal parameters, followed by classical blocks to perform compensating signal transformations based on those estimations.

Fig. 2 visualizes the processing pipelines of the transmitter and receiver in analogous colors as originally presented in [10]. Small solid-border boxes denote the individual processing steps (turquoise: ML-based, gray: classic, red: RTN) and their surrounding shaded dotted-border boxes indicate their respective functionality. By focusing only on the receiver algorithm, it is evident that a complex computational graph which interleaves ML- and classically-based operations is utilized. Without delving too deeply into the functionality of the individual components, two RTNs are employed. The lower one utilizes a CNN to estimate synchronization-related properties, such as the phase offset from the RF signal, followed by the corresponding signal transformation using classic operations.

The utilization of a mixture of classic and ML-based processing is essential for high-performance and high-throughput

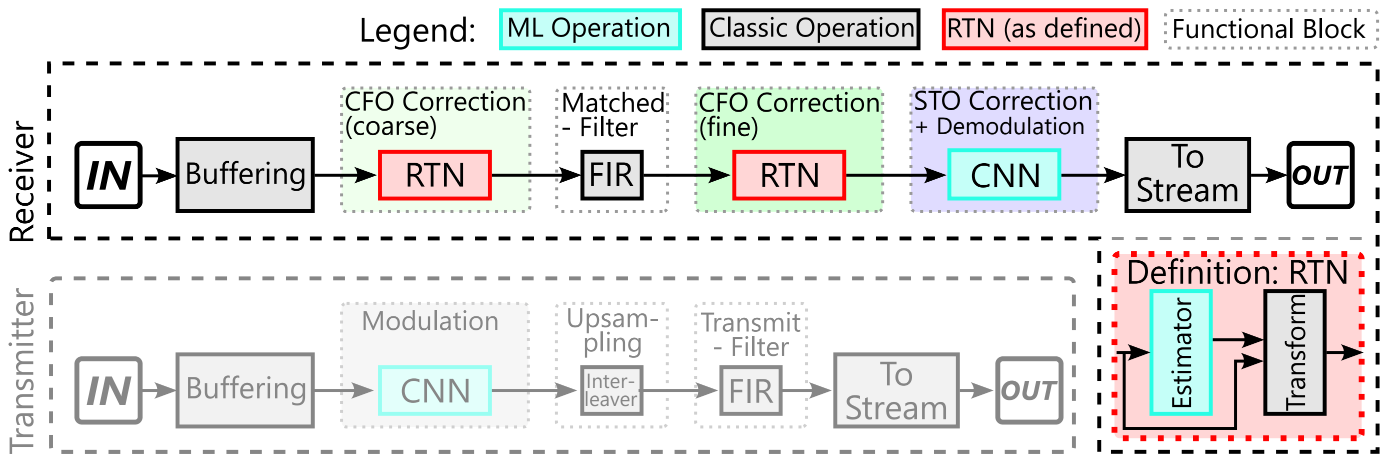

Fig. 2. Visualization of the transceiver processing chain. Machine Learning-based, classical, and RTN-based operations are shaded with turquoise, gray, and red colors, respectively. The surrounding dotted-border boxes indicate the processing blocks functionality.

DSP applications. Both types offer distinct advantages over each other in terms of functional performance and computational complexity depending on the specific task. A hybrid architecture that can not only support both ML and classical operations, but also interconnects them in a highly flexible manner to allow a complex data flow, is required to realize such AI-augmented concepts effectively. Therefore, the chosen application serves as a challenging deployment scenario which is detailed in the next section.

III. Deployment Strategy and Implementation

In this section, we outline the deployment strategy for the reference scenario on the AI- & DSP-Accelerator slice. We begin by examining the challenges associated with transitioning from a single-component to a multi-component processing system. Subsequently, we explore the capabilities of each component and their interconnectivity. From this we derive a mapping strategy that associates algorithm parts with hardware, and define a corresponding control strategy.

A. *Transitioning to a Modular Computational Architecture*

Porting an algorithm from a single-component to a multi-component compute platform bears a huge potential towards overall system performance, including throughput performance, power efficiency, scalability, resource optimization, and fault tolerance. The downside is typically an increased system complexity. New issues emerge, such as workload distribution and balancing, data buffering and synchronization, inter-component communication, supervision and control mechanisms, among others. These aspects are crucial and must be carefully addressed to achieve maximum effectiveness.

Since addressing all of these issues in this work is not feasible, in the remainder of this work we want to focus on the most crucial two, which are workload distribution and device intercommunication.

a) Workload distribution: Splitting up an algorithm and distributing its parts to various components requires consideration of the following:

- Matching computational requirements with native hardware capabilities. For instance, performing matrix multiplications on optimized rather than general purpose hardware, while maintaining sufficient numerical precision (e.g., fix- or floating-point format, bit-width).
- Ensuring sufficient local data cache to avoid slow external buffering.
- Maintaining equal execution times to allow efficient superscalar execution (refer to the Pipeline-Problem [23]).

In summary, the primary goal is to distribute the parts to the most suitable hardware components while ensuring rapid data transfers and interconnectivity. This ensures that the primary bottleneck of the system is compute power rather than secondary effects like memory size and data access limitations.

b) Inter-device connectivity: Efficient communication between 'neighboring' distributed parts is essential and directly impacts the overall system throughput when neglected. Especially in data-heavy applications, such as image processing and raw radio signal processing, connectivity has shown to often limit the system performance [24], [25]. However, one needs to distinguish between global and local connectivity. Global refers to the external interface that lets the device communicate to the other devices, for instance, using a system-bus or in case of the Versal the Network-on-Chip (NoC). Local refers to the internal capabilities of the device to store and move data between its low-level processing elements, such as the FPGA's Block-RAM or the AI-Engines' (see Sub-sec. III-B) local memory.

To address both of these subjects accurately, a deep understanding of the involved components is paramount. Therefore, the following subsection comprises an individual component-wise analysis before a hardware mapping can be performed.

B. *Overview and Analysis of Processing Resources*

In the following, we will summarize the essential components on the AI- & DSP-Accelerator slice, whose core component is the Versal AI-Core adaptive System-on-Chip [11]. Integrated inside are the Network-on-Chip (NoC), the

Processing System (PS), an FPGA (here also referred to as DSP-Accelerator), and an AI-Engine (AIE) array (part of the AI-Accelerator).

1) Network-on-Chip: The Network-on-Chip (NoC) serves as a multi-terabit interconnect between the different compute resources, memory (e.g., DDR4, HBM), and peripheral interfaces (e.g., PCIe, 100G Multirate Ethernet). Data enters the NoC through either a memory-mapped or streaming Advanced eXtensible Interface (AXI) port, which is an interface protocol defined in ARMs Advanced Microcontroller Bus Architecture (AMBA) standard. The NoC allows for flexible and dynamic access from any point within the SoC, servicing all major data generating and consuming components. The NoC is the central interface to the DDR memory controllers, external memory, on-chip memory (OCM), and Direct-Memory-Access (DMA) unit.

2) Processing System: The processing system is based on a dual-core ARM Cortex-A72 application processor and a dual-core ARM Cortex-R5F real-time processor with a primary intention for control and supervision tasks. It hosts an operating system (here: AMD-Xilinx's Petalinux) containing a Hardware Software Interface (HSI) that provides the necessary infrastructure (drivers, interfaces, configurations) to communicate with the SoC's components, such as the FPGA and the AI-engine array. The PS directly integrates an exhaustive number of peripherals, such as Gigabit Ethernet, Serial Peripheral Interface (SPI), Controller Area Network Flexible Data-Rate (CAN-FD), etc., and is furthermore connected to the NoC.

3) AI-Accelerator: The AI-Accelerator is a hardware-software co-design based on AMD-Xilinx's Deep-Learning Processor Unit (DPU) IP-core. It implements a configurable computation engine optimized for deep neural networks (DNN) using a synthesized code-block on the FPGA with a series of C++ programs deployed on the AI-Engines. The DPU is a generic ML accelerator that loads pre-compiled DNN models dynamically at run-time and executes them as needed. It currently only supports 8-bit fixed-point quantization of weights and features. For a more detailed introduction, please see [26].

Although the DPU utilizes multiple computational and memory resources internally, it can be perceived as a holistic component from a global perspective, whose main data interface (input and output) is the RAM.

4) DSP-Accelerator: For this hardware platform, the DSP-Accelerator consists solely of the SoC's FPGA. This terminology is used to ensure the generality of our proposed deployment concept and avoid being limited by resource-specific details. DSP components, such as filtering, FFT, or custom operations like transformation blocks of RTNs (see Sub-sec. II-B), are implemented on the FPGA using pre-built IP cores or RTL-level code. The available local storage is limited by the FPGA's internal RAM blocks (i.e., Block-RAM, Ultra-RAM, and distributed RAM), which total to around 200 MB for the largest chip.

5) Random Access Memory: Random Access Memory (RAM) is the only component which is not integrated inside the SoC, but placed on the AI- & DSP-Accelerator slice as a dedicated component. It is physically connected to the SoC's DDR memory controller, and made available to the other resources through the NoC, as mentioned above. The RAM is an especially crucial component, since it serves as a central element for data sharing between all mentioned components. The hereby used memory is an 8 Gb PC4-3200 DDR4 module with a maximum bandwidth of 25.6 GB/s [27].

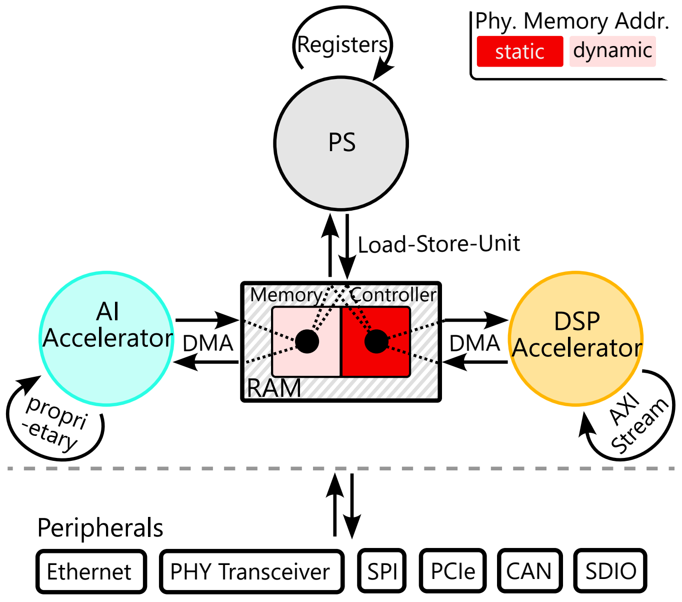

Fig. 3. Overview of the processing resources and their inter-communication using RAM. The PS utilizes its Load-Store Unit for memory access, while the AI- and DSP-Accelerator rely on DMAs, although their memory regions differ. Peripherals, such as Ethernet, Physical Transceivers, PCIe, etc., are available through one of the main processing resources.

After having introduced all relevant components, the following section elaborates on how they can be used in concert to enable a complex computation and data flow.

C. Inter-Component Communication

The interplay between various components is mainly defined by their ability to communicate with each other. On the one hand, this refers to the exchange of input, output, and intermediate data. On the other hand, this also refers to the ability to perform control and synchronization signalling to ensure smooth processing. For our further analysis we restrict our focus to data exchange only, since signalling does not present a significant bottleneck in our data-heavy scenario and is briefly handled in Sub-sec. III-E. Fig. 3 offers a glimpse into the main processing elements' data flow to and from one another. The RAM serves as a centralized storage hub, where all components obtain and store their processed data. This arrangement is inherently shaped by the centralized communication facilitated by the NoC and the individual components' lack of sufficient local storage. It therefore becomes clear, that the ability to exchange data between components is facilitated by the RAM and its access mechanisms.

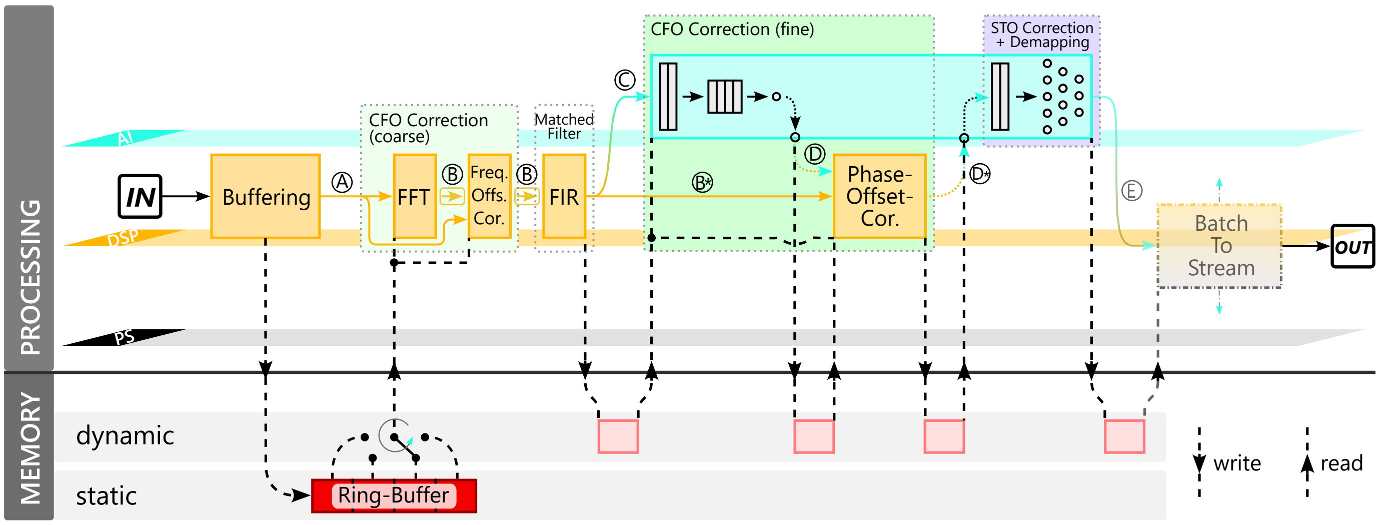

Fig. 4. Visualization of the hardware-mapping of the reference scenario's processing components and their respective data flow across the devices. Gray, yellow, and turquoise shaded colors refer to a mapping to the PS, DSP-Accelerator, and AI-Accelerator, respectively. Their memory access patterns are displayed on the bottom half of the figure, showing read and write operations into dynamic (light-red) and static (dark-red) buffer regions.

The PS has a rudimentary access mechanism solely through its Load-Store Unit and is therefore clearly not intended to actively participate in any data-intensive subtasks. This limitation extends to all peripherals that route data through the PS. Additionally, any RAM access, e.g., data copying between different regions (e.g., static and dynamic allocated buffers), is to be avoided. Since there exists no RAM-to-RAM DMA, this operation can sometimes be necessary, hence its performance is evaluated in sec. IV.

The AI-Accelerator uses separate buffers in the RAM for loading input data and storing the final result. Its access mechanism employs DMA for both directions. Due to the implementation of the underlying DPU, it is restricted to dynamically allocated buffers (i.e., accessible via virtual address), only. The internal mechanism to store intermediate results is proprietary and likely uses both RAM and FPGA-RAM.

Lastly, the DSP-Accelerator shares a similar interface like the AI-Accelerator, but it employs static memory-mapped input and output buffers configured during the operating system's build phase. AXI-Stream facilitates the transport of intermediate on-device data transfers, especially in scenarios involving buffering or a series of sequentially interconnected DSP blocks.

With the computational capabilities and data handling capacity of the components described, we are now ready to implement the reference scenario on hardware, as detailed in the next subsection.

D. Software-Hardware Mapping and Data Exchange

1) General Strategy and Data Fusion Approach: As of now, the software-hardware mapping is performed in a fully manual manner, since it enables the highest level of flexibility for exploring different design decisions. However, the mapping generally follows an intuitive and logical approach: ML-based blocks are always mapped to the AI-Accelerator if the operation is natively supported. Alternatively, the preferred fallback option is the DSP-Accelerator, followed by the PS. Classical processing blocks are preferably mapped to the DSP-Accelerator, with the PS as fallback option. It is crucial to understand that a mapping to the PS shall be avoided, even for low processing complexity steps, since its primary bottleneck is memory bandwidth, not computational power.

Equally important is to efficiently route the data between various processing components and domains. Key to efficient and fast data exchange is a zero-copy strategy. Zero-copy implies, that adjacent operations share the same memory interface, i.e., data buffer. In practise this requires aligning the memory addresses between adjacent input-output and output-input relationships. For instance, as visualized in Fig. 2, the Matched-Filter's output buffer address must equal the (fine) CFO Correction's input buffer address. Although a simple concept, aligning static (defined when building the operating system) and dynamic (defined during run-time) buffers in different memory regions and mapping virtual to physical memory address can be inconvenient, depending on the underlying drivers used.

2) Mapping of the AI-augmented Receiver Algorithm: As described in Sub-sec. II-B and shown in Fig. 2, the receiver algorithm consists of multiple interleaved classical- and ML-based processing blocks. The hardware mapping is visualized in Fig. 4. The processing flow is shown in the top half, the memory access patterns are shown in the bottom half. The same processing blocks as introduced in Fig. 2 are shown, with gray, yellow, and turquoise colors denoting their mapping to PS, DSP-, and AI-Accelerator, respectively.

It is evident, that the mixture of processing domains results in a variety of interconnections, which are labelled using letters surrounded with a circle. Implementing them efficiently is the major challenge in a multi-component system. These interconnections are now described in detail:

- **A**: Data transfer between peripheral and DSP-Accelerator. Continuous data is buffered in a statically allocated ring buffer by the peripheral's driver, whose segments are then read by the DSP-Accelerator using its DMA.
- **B**: Data transfer between two sequential DSP-blocks. Data is directly streamed between both blocks using AXI-Stream. No significant local data storage is required, hence, RAM access is not necessary.
- **B***, **D**, and **D***: See below.
- **C**: Data transfer between DSP- and AI-Accelerator: The output of the DPS-Accelerator is stored via DMA in the dynamic input buffer of the AI-accelerator, which is then read via DMA by the AI-Accelerator.
- **E**: Data transfer between AI-Accelerator and DSP-Accelerator or PS: The output of the AI-Accelerator is stored via DMA in a dynamic buffer, which can directly be accessed by the DSP-Accelerator via DMA or by the PS using its Load-Store unit.

A significant complication is introduced within the AI-Accelerator, which features an RTN block whose transformation component (Phase-Offset-Correction) is classical, and hence deployed on the DSP-Accelerator. This effectively leads to a classical component enclosed within two ML components. While in general it is possible to deploy multiple separate neural networks (NN) on the AI-Accelerator to address such an issue, the approach of a single NN is pursued here. This is made possible by utilizing the DPU's Custom-Layer functionality, which allows to execute arbitrary code, and hence even a block on the DSP-Accelerator, in-between two layers of the NN. In our case, the following interconnections are required:

- **B***: Data transfer between a stand-alone and an interleaved DSP-block: Since the data cannot be directly stream between both DSP-blocks due to additional data dependencies on the AI-Accelerator, the input buffer is used as temporary storage, and simply read via DMA when required.
- **D**: Data transfer of an intermediate result between AI- and DSP-Accelerator: The intermediate result is automatically stored in a dynamic buffer by the AI-Accelerator and is then read via the DSP-Accelerator's DMA.
- **D***: Same as D, but in reverse direction and order.

Finally, after defining the hardware mapping, interconnections, and memory access patterns, we define a control procedure that supervises the complete computation process.

E. Process Control Procedure

The computation process is controlled by a Python application running on the PS. For simplicity, computation is executed in a one-shot, non-pipelined manner. Pipelining is discussed in Sec. V. The application has the following tasks:

- Wait asynchronously for a ring buffer segment to be filled with data from the RF frontend slice.
- Start FFT, frequency-offset correction, and FIR filter component on DMA-Accelerator, wait until completion.
- Start AI-Accelerator. Upon reaching the phase-offset correction block, the DPU's Custom-Layer API executes a custom script and supplies its dynamic input and result buffer addresses, which triggers the DSP-Accelerator's DMA for both dynamic buffers. Upon completion of the block, the result is written via DMA into the dynamic output buffer supplied by the Custom-Layer API, and control is returned back to the AI-Accelerator.
- Upon completion of the AI-Accelerators execution, the final result is available in the AI-Accelerators dynamic output buffer and can be further routed to its destination.

With the procedure defined, the reference scenario is fully implemented. We can now begin to evaluate the performance of this implementation, which is detailed in the next section.

IV. Performance Evaluation

This goal of this section is to explore the practical performance of our deployment concept. Since the AI-Accelerator's performance has already been sufficiently analyzed [25], [26], [28], [29], and the DSP-Accelerator's performance is highly predictable due to its reliance on the FPGA, we focus our analysis on the components' inter-communication capabilities. After reviewing the measurement methodology and system configuration, we summarize our findings and provide a critical discussion on possible improvements.

A. Measurement methodology and system configuration

As observed in Sub-sec. III-D, a complex computation graph necessitates various communication interfaces between the involved components. Only a single inefficient interface can have a significant impact on the overall system performance, which is why this section evaluates the achievable data transfer speeds between all involved components. In the following we list the related interfaces, calculate their theoretical maximum performance, and detail our measurement strategy to evaluate their realistic performance. For all measurements we use a buffer size of 64 MiB.

a) AI-Accelerator - RAM: The AI-Accelerator utilizes DMA to read and write its input and output data. Since the underlying DPU is a proprietary and encrypted hardware-software co-design, its realistic bandwidth to memory can only be estimated. Since running an "empty" model with zero layers (to remove any computation burden and hence time) is not supported, we can only measure the total inference time for valid neural networks. Therefore, we compile and run the execution time of multiple models that each contain the exact same layer (here: convolutional layer with resolution 512x512, kernel 7x7, 128 input- and output-channels, same padding) a different amount of times, which lets us determine the isolated layer run-time, and therefore the raw data transmission time by subtracting the layer execution time. Note, that the model loading time is (correctly) not included in this measurement, as model loading is performed prior to execution. This method

TABLE I
PERFORMANCE ANALYSIS OF THE PS, AI-, AND DSP-ACCELERATOR'S THEORETIC AND MEASURED COMMUNICATION CAPABILITIES WITH THE MEMORY. THE EVALUATED METRIC IS THE DATA RATE.

Operation	Route			Throughput [GiB/s]		
	Source	via	Destination	Forward	Backward	Theoretic Max.
Data Feed	AI-Accelerator	DMA	RAM *(dynamic)*	3.19		4.47 - 26.82
Data Feed	DSP-Accelerator	DMA	RAM *(dynamic)*	14.59	14.91	17.88
DSP Pipeline	DSP-Accelerator	AXI-Stream	DSP-Accelerator	-	-	17.88
Copy Buffer	RAM	PS Load-Store-Unit	RAM	4.34	4.29	8.94

computes the average transfer rate for reading and writing, the separate values cannot be identified.

The theoretical maximum rate can be computed based on the bus-width of the corresponding input and output data AXI-port[1] ($128 \cdot$ batch-size), for a batch-size up to six. For a clock frequency, which is typically set to 300 MHz, this results in a maximum theoretic data rate of 4.47 GiB/s per Batch, totalling to 26.82 GiB/s for multi-batch execution. The configuration analyzed here utilizes single-batch execution only.

b) DSP-Accelerator - RAM: The DSP-Accelerator utilizes DMA to access the RAM. As a first-order approximation, the speed depends on the DMA's frequency and bus-width and can be calculated as

$$\text{Transfer Speed} = f_{\text{DMA}} \cdot \text{bus-width}_{\text{DMA}}. \quad (1)$$

Although f_{DMA} can reach values up to 500 MHz [29], a popular setting that avoids timing problems with most IP-cores (including the DPU) is 300 MHz, which is also used here. Although the speed is expected to be similar in forward (reading from the RAM) and backward (writing to the RAM) direction, it is measured in both directions by implementing an "empty" receiving IP-core on the FPGA (to avoid transfer stalls caused by computation) when reading data, and a constant data source when writing data to the RAM. The DMAs are controlled via the python py-uio library [2].

c) DSP-Accelerator - DSP-Accelerator: Communication between two sequential DSP-blocks is implemented via AXI-Stream, which has an intrinsic deterministic performance. Its performance is calculated as

$$\text{Transfer Speed} = f_{\text{AXI-Stream}} \cdot \text{bus-width}_{\text{AXI}}. \quad (2)$$

With $f_{\text{AXI-Stream}} = 300\,\text{MHz}$ and AXI-buswidth $= 512$, the interface speed results to $17.88\,\text{GiB/s}$. No measurement has been performed, since the calculated speed is expected to be identical with the practical speed.

d) PS - RAM: The PS accesses the RAM via its Load-Store Unit, and hence can be easily evaluated by timing the corresponding system-native memory copy functions. Since our control program utilizes python, we use *ctypes*[3] system-native *memmove* method to evaluate the transfer speeds from a static to a dynamically allocated buffer and in reverse direction. On a first-order approximation, the theoretical limit is defined by the max. CPU clock frequency ($f_{\text{CPU,max}} = 1.2\,\text{GHz}$) and the memory bus-width (64 bits), resulting to 8.94 GiB/s.

B. Measured Performance Results

Table I summarizes the benchmarked operations and their results. The PS-based buffer copy mechanism achieves a realistic rate of 4.3 GiB/s which is rougly identical in both directions, realizing 48% of its theoretical speed. The interface between AI-Accelerator and RAM achieves a 71% utilization with 3.19 GiB/s transfer speed. The DSP-Accelerator is able to access the memory with a much higher rate of 14.59 and 14.91 GiB/s in forward and backward direction, respectively, realizing 82 % of its maximum potential. Lastly, as mentioned before, the on-device data transfer of the DSP-Accelerator achieves 17.88 GiB/s.

C. Discussion and Future Improvements

The above results are promising, indicating efficient utilization of the overall system through our deployment approach. The AI- and DSP-Accelerators demonstrate effective coordination in exchanging process data. Notably, the DSP-Accelerator's utilization of multiple sequential processing blocks proves highly effective, achieving data transfers at theoretical maximum speeds.

Measurements confirm that the PS's access inefficiency to memory poses a potential performance bottleneck if utilized, for instance, when performing buffer copy operations. By utilizing cache coherent memory access (not used in this work), an increase in bandwidth is expected. Another reasons for the gap between theoretical and measured communication speed for the other components is the non-optimized configuration of DMA frequencies and bus widths.

Not analyzed in this work is the additional overhead incurred by data conversion or formatting (e.g., from INT8 to FLOAT32) between different processing components, which is expected to have a minimal impact on the overall performance. These conversions would typically be deployed on the DSP-Accelerator.

V. Conclusion and Future Work

In conclusion, this paper has demonstrated the successful integration of artificial intelligence (AI) processing capabilities into high-performance satellite platforms, specifically within the context of telecommunications signal processing. By interleaving AI-based computation blocks within an FPGA-based

[1]https://docs.xilinx.com/r/en-US/pg389-dpucvdx8g/Interface-Ports ref. to port *Mxx_IMG_AXI*.

[2]py-uio library website: https://github.com/mvduin/py-uio

[3]ctypes python library: https://docs.python.org/3/library/ctypes.html

processing framework, we have shown the feasibility of enhancing traditional data processing workflows with advanced AI techniques. Through a thorough evaluation of hardware components and communication interfaces, we have achieved a seamless deployment of an AI-augmented signal processing pipeline on state-of-the-art telecommunications processor hardware. Our results highlight the potential for improving satellite platform performance and efficiency through the intelligent use of AI technologies.

a) Future work: Moving forward, there are several avenues for future research and development in this area. Firstly, further optimization of communication interfaces and hardware configurations could lead to enhanced system performance and efficiency. Moreover, pipelining the data flow bears significant overall performance potential, ideally keeping the DSP- and AI-Accelerator occupied at all times and removing the idle periods when waiting for prior computation to finish. This would require the introduction of additional buffer memory between each pipelined computation stage and a modification to the zero-copy strategy to preserve optimal communication. A further extension is to realize a multi-DPU-based AI-Accelerator to run multiple different neural networks simultaneously. Additionally, a highly relevant future feature could be to implement real-time adaptivity to the AI-related processing components, such as enabling in run-time weights update or neural network training. Lastly, the topic of redundancy, system behaviour in case of component faults, and options to restore service should be addressed.

References

[1] M. Ghiglione and V. Serra, "Opportunities and challenges of AI on satellite processing units," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, ser. CF '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 221–224. [Online]. Available: https://doi.org/10.1145/3528416.3530985

[2] A. Koch, A. Krstova, F. Hegwein, M. C. De Lera *et al.*, "On-Board Anomaly Detection on a Flight-Ready System," in *2023 European Data Handling & Data Processing Conference (EDHPC)*, 2023, pp. 1–4.

[3] G. Giuffrida, L. Fanucci, G. Meoni, M. Batič *et al.*, "The -Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022.

[4] G. Zhu, D. Liu, Y. Du, C. You *et al.*, "Toward an Intelligent Edge: Wireless Communication Meets Machine Learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[5] N. Destrycker, W. Benoot, J. Mattias, I. Rodriguez, and D. Steenari, "EDGX-1: A New Frontier in Onboard AI Computing with a Heterogeneous and Neuromorphic Design," in *2023 European Data Handling & Data Processing Conference (EDHPC)*, 2023, pp. 1–6.

[6] Brainchip, "Benchmarking AI Inference at the Edge: Measuring Performance and Efficiency for Real-World Deployments," 2023, https://brainchip.com/wp-content/uploads/2023/01/BrainChip_Benchmarking-Edge-AI-Inference-1.pdf [Accessed: 10. March 2024].

[7] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya *et al.*, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[8] G. Benelli, G. Giuffrida, R. Ciardi, D. Davalle *et al.*, "GPU@SAT, the AI enabling ecosystem for on-board satellite applications," in *2023 European Data Handling & Data Processing Conference (EDHPC)*, 2023, pp. 1–4.

[9] P. Ghiglino, M. Harshe, D. Stenaari, and L. Mansilla, "AI/ML Inference Engine Software for High-Reliability applications on Space Qualifiable Hardware," in *2023 European Data Handling & Data Processing Conference (EDHPC)*, 2023, pp. 1–11.

[10] M. Petry, B. Parlier, A. Koch, and M. Werner, "Auto-Regressive RF Synchronization Using Deep-Learning," in *IEEE International Conference on Machine Learning for Communication and Networking*, 2024, [accepted, pre-print available at https://www.bgd.ed.tum.de/pdf/2024_RFSynchronization_ICMLCN_Petry.pdf].

[11] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, "Xilinx Adaptive Compute Acceleration Platform: VersalTM Architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[12] A. Bourdoux, A. N. Barreto, B. van Liempd, C. Lima *et al.*, "6G White Paper on Localization and Sensing," 06 2020.

[13] J. Hoydis, S. Cammerer, F. A. Aoudia, A. Vem *et al.*, "Sionna: An Open-Source Library for Next-Generation Physical Layer Research," 2023.

[14] A. Alkhateeb, S. Jiang, and G. Charan, "Real-Time Digital Twins: Vision and Research Directions for 6G and Beyond," *IEEE Communications Magazine*, vol. PP, pp. 1–7, 11 2023.

[15] S. Cammerer, F. A. Aoudia, S. Dörner, M. Stark *et al.*, "Trainable Communication Systems: Concepts and Prototype," *IEEE Transactions on Communications*, vol. 68, no. 9, 2020.

[16] M. B. Fischer, S. Dörner, S. Cammerer, T. Shimizu *et al.*, "Adaptive Neural Network-based OFDM Receivers," in *2022 IEEE 23rd International Workshop on Signal Processing Advances in Wireless Communication (SPAWC)*, 2022, pp. 1–5.

[17] S. Dörner, S. Rottacker, M. Gauger, and S. t. Brink, "Bit-wise Autoencoder for Multiple Antenna Systems," in *2021 17th International Symposium on Wireless Communication Systems (ISWCS)*, 2021.

[18] S. Cammerer, J. Hoydis, F. Aoudia, and A. Keller, "Graph Neural Networks for Channel Decoding," 12 2022, pp. 486–491.

[19] W. Xu, Z. Zhong, Y. Be'ery, X. You, and C. Zhang, "Joint Neural Network Equalizer and Decoder," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, 2018, pp. 1–5.

[20] 3GPP, ""3rd Generation Partnership Project; Technical Specification Group Radio Access Network; New SID on AI/ML for NR Air Interface"," 3GPP, Tech. Rep. RP-212708, 2021.

[21] ——, ""3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on Integrated Sensing and Communication (Release 19)"," 3GPP, Tech. Rep. TR 22.837, 2024.

[22] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[23] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[24] N. P. Jouppi, C. Young, N. Patil, D. Patterson *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, jun 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[25] M. Petry, G. Wuwer, A. Koch, P. Gest *et al.*, "Accelerated Deep-Learning Inference on the Versal adaptive SoC in the Space Domain," in *2023 European Data Handling & Data Processing Conference (EDHPC)*, 2023, pp. 1–8.

[26] M. Petry, P. Gest, A. Koch, M. Ghiglione, and M. Werner, "Accelerated Deep-Learning inference on FPGAs in the Space Domain," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 222–228. [Online]. Available: https://doi.org/10.1145/3587135.3592763

[27] M. Technology, "MTA9ADF1G72AZ DDR4 SDRAM VLP UDIMM," 2024, https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/modules/unbuffered_dimm/ddr4/adf9c1gx72az.pdf?rev=4d6f78da73bb4d5f89724be3e7073314 [Accessed: 09. March 2024].

[28] A. Al-Zoubi, G. Martino, F. H. Bahnsen, J. Zhu *et al.*, "CNN Implementation and Analysis on Xilinx Versal ACAP at European XFEL," in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, 2022, pp. 1–6.

[29] M. Wierse, "Bachelor Thesis: Evaluation of Xilinx Versal Device," 2023, https://doi.org/10.3929/ethz-b-000600880.