Time-Predictable Acceleration of Deep Neural Networks on FPGA SoC Platforms

Francesco Restuccia*† and Alessandro Biondi*†
*TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
†Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy

***Abstract*—This work focuses on the time-predictable execution of Deep Neural Networks (DNNs) accelerated on FPGA System-on-Chips (SoCs). The modern DPU accelerator by Xilinx is considered. An extensive profiling campaign targeting the Zynq Ultrascale+ platform has been performed to study the execution behavior of the DPU when accelerating a set of state-of-the-art DNNs for Advanced Driver Assistance Systems (ADAS). Based on the profiling, an execution model is proposed and then used to derive a response-time analysis. A custom FPGA module named DICTAT is also proposed to improve the predictability of the acceleration of DNNs and tighten the analytical bounds. A rich set of experimental results based on both analytical bounds and measurements from the target platform is finally presented to assess the effectiveness and the performance of the proposed approach on ADAS applications.**

I. Introduction

In recent years, the number of Cyber-Physical Systems (CPS) that make use of Artificial Intelligence (AI) and, in particular, Deep Neural Networks (DNNs) is significantly increased. Among popular AI algorithms, DNNs proved to be particularly effective in performing perception tasks required to implement autonomous/assisted driving functionality for modern vehicles, as well as other mission-specific functionality for advanced robots and factory automation systems. The execution of state-of-the-art DNNs (also called network inference) requires dealing with a complex computing workload that processes a large amount of data with massively-parallel computations [1]. As such, it commonly requires hardware acceleration to be accomplished under stringent timing constraints [2].

To match this need, chip manufacturers started producing heterogeneous embedded computing platforms that integrate multiprocessors with hardware accelerators such as general-purpose graphic processing units (GPUs), field-programmable gate arrays (FPGAs), and digital signal processors (DSPs). Among such platforms, those based on GPUs recently received enormous attention by the real-time systems research community, which achieved a string of interesting results to improve the time predictability of GPU-accelerated tasks. Nevertheless, GPU-based platforms present a set of drawbacks that may not make them the best choice for CPS, at least for those that require hardware acceleration for implementing safety-critical functionalities. Indeed, popular commercial GPUs, e.g., those produced by Nvidia, exhibit poor execution predictability and proved to be unsuitable for critical systems, also due to the fact that they implement an internal scheduling logic that is not publicly documented. In general, very limited information on their behavior is known (interesting reverse engineering efforts notwithstanding [3], [4]). Furthermore, GPUs are characterized by a high energy consumption [5], [6].

FPGA-based platforms represent an attractive alternative for realizing time-predictable embedded computing systems with hardware acceleration. They allow deploying energy-efficient accelerators with competitive performance [6] with respect to GPUs. FPGA accelerators are also more suitable for timing predictability because they are often characterized by a very regular, clock-level behavior and allow for an explicit control of the memory traffic they generate, being the hardware design to be deployed on FPGA under the full control of the designer. Even if commercial FPGA accelerators are typically distributed as closed-source modules, the direct access to the hardware design allows precisely profiling and monitoring the bus traffic they generate, hence achieving an accurate characterization and supervision of their execution behavior that would be simply impossible with other platforms.

Contribution. This paper is focused on FPGA-based acceleration of DNNs by means of the Xilinx DPU accelerator, which to the best of our records is the most mature solution of this kind on the market at the time of writing. An extensive profiling campaign focused on the Xilinx Zynq Ultrascale+ platform has been conducted to study the execution behavior of the DPU by means of a DPU-specific FPGA profiler we developed. Based on the profiling, we designed an execution model for the DPU, which is later used to derive a response-time analysis. We further present a technique to improve the execution predictability of DNNs models using DICTAT, a helper FPGA module we developed to fetch DPU instructions from on-chip memory. All the experimental evaluations presented in the paper are based on state-of-the-art DNNs for Advanced Driver Assistance Systems (ADAS).

Paper structure. The rest of the paper is organized as follows. Section II introduces the system architecture under analysis and the most popular solutions for DNN acceleration on FPGA SoCs. Section III describes the experimental profiling campaign we conducted. Section IV introduces the proposed model for the DPU and the system architecture. Section V presents the proposed worst-case analysis. DICTAT and the corresponding refinements to the analysis are presented in Section VI. Section VII discusses the experimental evaluation, while Section VIII illustrates the related work. A discussion

on the proposed approach, its limitations, and future developments is reported in Section IX. Finally, Section X concludes the paper.

II. Background and System Architecture

A. FPGA SoC architecture

A typical FPGA SoC architecture combines a *Processing System* (PS), including one or more processors (generally ARM-based), with a *Field-Programmable Gate Array* (FPGA) fabric in a single device. The processors in PS execute software tasks (SW-tasks). The FPGA fabric can be programmed to host custom hardware devices such as *hardware accelerators*. Figure 1 illustrates a typical FPGA SoC architecture. Typically, computations are controlled by SW-tasks, which can in turn activate the hardware accelerators when required. The hardware accelerators and the processors can communicate through a shared off-chip DRAM memory or an On-Chip Memory (OCM). The DRAM memory is accessed by a DRAM memory controller, embedded in PS, and shared between the PS and the FPGA subsystems. This is crucial to enable high-performance, asynchronous data communication among hardware accelerators and processors. The communications between the PS and the FPGA subsystems are allowed by two interfaces: the FPGA-PS interface and the PS-FPGA interface. The FPGA-PS interface exports a set of high-throughput ports allowing the access of the hardware accelerators to the devices in PS (e.g., DRAM memory controller, OCM, peripherals). Conversely, the PS-FPGA interface exports a set of ports leveraged by the processors to access and manage the hardware accelerators. The data movement relies on the AMBA AXI bus, which is the de-facto standard for communications in modern SoCs [7]. The bus traffic within the PS subsystem (e.g., originated from the processors or the devices in the FPGA fabric and directed to the DRAM controller and other peripherals in PS) is managed by a multi-level AXI-based PS interconnect.

B. Frameworks for DNN acceleration on FPGA SoCs

To date, most of the development and deployment efforts for DNN models rely on powerful and energy-intensive GPU-based systems. The parameters of such DNN models, such as activations, weights, and biases are typically represented as 16-bit or 32-bit floating-point data. Due to the intrinsic differences in the architecture of GPU and FPGA platforms, most of the common neural networks deployed for execution on GPUs are not compatible out-of-the-box with FPGA SoC platforms. Indeed, while in GPU-based systems neural networks can be executed through calls to parallel computation APIs (e.g., the CUDA API in NVIDIA platforms), FPGA SoC platforms require a specific *hardware accelerator* deployed into the FPGA fabric for the execution of DNN models. The academia and the industry proposed several frameworks to cope with FPGA-based acceleration of DNNs, which combine conversion tools and specialized hardware accelerators for the deployment of DNNs on FPGA SoC platforms [8]–[11]. Among such a variety of frameworks, to the best of our records the most mature one is the Vitis AI framework [11] by Xilinx.

1) The Vitis AI framework: Vitis AI provides a collection of tools, libraries, and hardware accelerator IP cores for the conversion and execution of GPU-like floating-point DNN models upon Xilinx FPGA SoC platforms. Vitis AI supports DNN models deployed through the most mainstream neural network frameworks, such as Caffe, Tensorflow, and Pytorch. The execution of the DNN layers in Vitis AI relies on the *Deep learning Processing Unit* (DPU) core. The DPU is a hardware accelerator to be deployed into the FPGA fabric and optimized for the execution of convolutional DNNs. A brief description of the DPU core is provided in Section II-D. The DPU engine is configured by a control software application running in the PS of the FPGA SoC platform. Control software applications can be developed by leveraging the libraries provided by the Vitis AI framework.

C. DNN working flow with Xilinx Vitis AI

As for any framework for FPGA SoC platforms, a DNN model must undertake some process in Vitis AI before being ready to be executed on the target platform. In particular, in Vitis AI this process involves a quantization phase and a compilation phase. Such phases are performed once offline, typically on a powerful workstation. The output of such steps is a set of instructions and data for the DPU core.

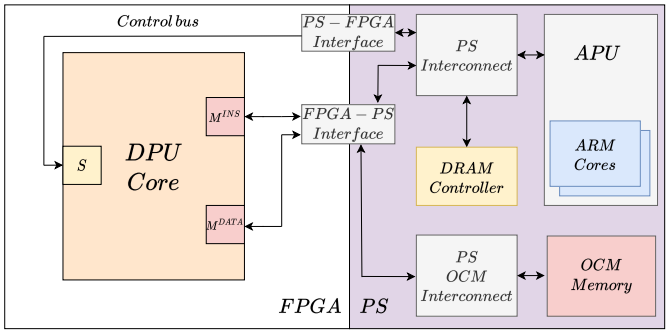

Fig. 1. The sample architecture of a Xilinx FPGA SoC platform deploying the DPU core in the FPGA fabric.

1) Preparation and quantization: The input to the Vitis AI framework is a pre-trained (GPU-like) floating-point neural network model. As of today, the Vitis AI framework supports only convolutional DNN models operating on images. As a first step, the data structure of the DNN model must be made compatible with the features of the DPU core. This means that the whole DNN data structures must be converted from floating-point to 8-bit fixed-point data. This process is called *quantization* and is performed by the Vitis AI quantization tool. Quantization is supported by a calibration phase, which makes use of a subset of the training images to minimize accuracy losses [11]. The output of the quantization process is the quantized DNN model. The quantized DNN model is then provided to the Vitis AI compiler tool that parses the quantized DNN model and creates an intermediate representation of the DNN. Such an intermediate representation is then mapped to a sequence of instructions for the DPU. The Vitis AI tool places

the generated instructions into a `.xmodel` file. At this point, the Vitis AI software application that is going to run on the target FPGA SoC platform can be built. Its development relies on the Vitis AI software libraries, which contains a set of APIs for the whole operation of the DPU core. The application is then cross-compiled for the target platforms using the Xilinx cross-compilation tools.

2) Running on the target FPGA SoC platform: Vitis AI applications are distributed as Vitis AI images, which are based on a Petalinux[1] image target for the FPGA SoC platform under analysis. Once launched, a Vitis AI application starts the configuration phase. This phase includes the preparation of the memory buffers in the central DRAM memory for the execution of the DPU. The application loads in DRAM memory the instructions and the weights provided by the `.xmodel` file. The DPU core is then configured by the Vitis AI software application. Once the configuration is done, the Vitis AI software application triggers the DPU to start the execution. At this point, the DPU is autonomous in the execution of the DNN model. The Vitis AI software task is suspended until the execution of the DPU completes (the DPU will notify the processors once the execution is done by means of an interrupt signal). The DPU fetches the instructions, the weights, and the input image to be processed by the DNN from the DRAM buffer prepared by the Vitis AI software application. The fetching of instructions and data is performed in parallel by the DPU core leveraging multiple memory ports.

D. DPU core hardware accelerator

The DPU disposes of direct access to the memory and the PS through multiple AXI interfaces. Figure 1 illustrates a sample FPGA SoC architecture including a DPU core. With reference to the figure, S is the AXI-lite subordinate interface leveraged by the SW-tasks running in the PS for the configuration of the DPU core; M^{INS} is the AXI manager interface used by the DPU core for fetching the DNN instructions to be executed from the DRAM memory; and M^{DATA} is the AXI manager interface leveraged by the DPU core for reading and writing data from/to the DRAM memory. This latter interface is used to fetch the parameters (mainly weights) of the various DNN layers, to fetch the input image to be processed, to read/write intermediate results generated during DNN inference, and eventually writing the final DNN outputs.

Unfortunately, the DPU core is distributed as a closed-source IP block only. To the best of records, no detailed information about its internal behavior is publicly available.

III. Profiling the DPU

Although the DPU is a proprietary accelerator distributed as a closed-source module, it was possible to understand several aspects of its execution behavior by developing a custom hardware module deployed on the FPGA fabric to perform advanced profiling of the DPU execution and memory access patterns. It is worth stressing the fact that such an advanced analysis was possible only thanks to the hardware programmability of FPGA SoC — the same analysis would be very difficult, if not impossible, to be conducted on commercial GPU-based SoC platforms.

To obtain precise measurements, we developed a multi-channel hardware profiler that we integrated in the stock hardware design of the Vitis AI framework. We connected our hardware profiler to probe all of the interfaces and ports of the DPU, such as the AXI interfaces and the interrupt line, and accurately keep track of the interaction of the DPU with the DRAM memory with the fine granularity of each clock beat. It is connected in parallel with the stock connections, hence ensuring that the execution of the DPU is not perturbed by the profiler.

Our hardware profiler is capable of recording the behavior of the DPU at different levels: **(i)** the DPU bus activity for a given DNN model, i.e., the number of read/write transactions issued over time, their burst length, and the corresponding amount of exchanged data, **(ii)** the structure of execution phases of the DPU core, **(iii)** the execution time of the phases and the variability of the total inference time and **(iv)** the identification of pipelining among the execution phases (i.e., the time in which multiple execution phases are overlapped).

To consider a representative profiling campaign, we focused on popular state-of-the-art DNN models that are part of modern ADAS applications: they include **(i)** a lane detection DNN based on a VpgNet model [12], **(ii)** a plate detection DNN model, **(iii)** a plate recognition DNN model, **(iv)** an object detection DNN based on a Yolov3 model [13], **(v)** an object detection DNN based on an SSD model [14], and **(vi)** a pedestrian detection DNN based on an SSD model. For each of such DNN models, we recorded 1000 DPU executions (to perform inference of the network) by means of our hardware profiler. The latest version (v1.3.2) at the time of writing of Xilinx Vitis AI was used together with the stock Vitis AI configuration based on Petalinux [15] as provided by Xilinx. The profiling was conducted on a Xilinx ZCU102 development board equipped with the Xilinx Zynq Ultrascale+ FPGA SoC. The considered DPU architecture is the default one (version 3.3) provided with Vitis AI for the Zynq Ultrascale+. The DPU clock is left to the default value of 330Mhz. All the DNN models under analysis operate on images. Following the stock Vitis AI working flow, each execution involves the analysis of a single image and produces an output result. In the following, we refer to a single execution as a *DPU job*. During our profiling campaign, the input image for each DPU job was randomly picked from the CityScapes dataset for ADAS applications [16].

A. Profiling the DPU bus activity

Table I reports the bus activity recorded with our hardware profiler. The columns of Table I report the per-job bus activity of the DPU, in order **(1)** number of read transactions for instruction fetching (i.e., issued via the M^{INS} port), **(2)** data words fetched corresponding to instructions, **(3)** number of read transactions for data read (i.e., issued via the M^{DATA} port)

[1]Petalinux is a Linux distribution based on Yocto and targeted to run on Zynq Ultrascale+ platforms.

TABLE I
PROFILED PER-JOB BUS ACTIVITY OF THE DPU CORE FOR THE DNN MODELS FOR ADAS APPLICATIONS UNDER ANALYSIS.

DNN bus activity	Num Instr trans	Data instr words (size)	Num Read trans	Data read words (size)	Num write trans	Data write words (size)
Lane Detect	17186	68744 (275KB)	91939	1179184 (17.99MB)	48314	424350 (6.48MB)
Plate Detect	2347	9388 (37KB)	7607	83027 (1.27MB)	246	16960 (0.3MB)
Plate Num	9872	39488 (154KB)	53327	579962 (8.85MB)	6075	48908 (0.7MB)
Object Detect (Yolo)	16060	64240 (251KB)	84410	1011665 (15.44MB)	25457	540416 (8.24MB)
Object Detect (SSD)	9920	39680 (155KB)	68943	725208 (11.06MB)	4705	554510 (8.46MB)
Pedestrian Detect (SSD)	11655	46620 (182KB)	56597	639932 (9.76MB)	5505	506096 (7.72MB)

TABLE II
MEASURED PER-JOB EXECUTION TIMES OF THE DPU PHASES AND TOTAL MEASURED PER-JOB INFERENCE TIMES.

DNN model inference times (ms)		Instrc fetch	Read data	Write data	Pure Elaboration	Total Inference
Lane Detect (VpgNet)	min	2.23	6.15	4.17	0.01	7.09
	avg	2.26	6.81	4.19	0.04	7.1
	max	2.29	7.11	4.20	0.58	7.12
Plate Detect	min	0.29	0.51	0.12	0.01	0.73
	avg	0.3	0.71	0.13	0.02	0.74
	max	0.31	0.74	0.14	0.2	0.75
Plate Num	min	1.27	2.46	0.56	0.01	3.05
	avg	1.28	2.99	0.57	0.02	3.06
	max	1.32	3.01	0.58	0.2	3.07
Object Detection (Yolov3 ADAS)	min	2.28	7.49	3.46	0.01	7.99
	avg	2.31	7.93	3.47	0.1	8.01
	max	2.35	8.01	3.48	0.23	8.02
Object Detection (SSD ADAS)	min	1.52	6.92	3.74	0.01	8.39
	avg	1.54	8.3	3.76	0.1	8.4
	max	1.56	8.4	3.77	0.7	8.41
Pedestrian Detection (SSD ADAS)	min	1.60	7.95	3.47	0.01	9.10
	avg	1.62	8.87	3.48	0.1	9.11
	max	1.64	9.11	3.49	0.6	9.12

(4) data words fetched corresponding to data read, **(5)** number of write transactions for data write **(6)** data words written to the memory. Such results are reported for each of the DNN models under analysis.

Note that, even though we performed 1000 executions, the amount of interactions with the memory changes very little from one execution to another: we recorded changes in the order of less than 0.1% only. The bus activity of the DPU also resulted to be independent of the input image. This first observation provides a hint on the *high predictability* of the execution of the DPU core. Indeed, memory access and memory contention are the major sources of unpredictability for FPGA-based accelerators — a predictable bus activity has hence a strong positive impact on the predictability of the whole system [17], [18].

Besides the amount of data exchanged, other important features to characterize the performance and predictability of the DPU core are how the data are exchanged, in other words, the parallelism of the manager ports M^{INS} and M^{DATA} (i.e., the number of transactions each port can have pending, also called number of outstanding transactions) and the burst length of the bus transactions. The parallelism of ports M^{INS} and M^{DATA} influences the execution time of the DPU. Typically, the higher the parallelism of the port, the higher the data throughput. Unfortunately, these numbers are not publicly disclosed by Xilinx.

Thus, we developed a specific functionality in our hardware profiler to retrieve these characteristics. We experimentally found that the parallelism of the port M^{DATA} is of 14 read outstanding transactions and 7 write outstanding transactions. Differently, the parallelism of the M^{INS} port is of 2 outstanding read transactions (it is worth remembering that M^{INS} is used by the DPU only for reading instructions). Concerning the burst length of transactions, we found that the burst length of the transactions issued on M^{DATA} varies during the DPU execution, from the minimum of one word to the maximum of 256 word for both read and write transactions – these correspond to the limits defined by the AXI standard. Differently, the burst length of the transactions issued on the M^{INS} port is fixed and equal to 4 words.

B. Measuring the DPU execution phases

From the profiling we noted how the DPU has three concurrent bus activities: read instructions, read data, and write data. Each of such activities is associated with an execution time contributing to the total execution time of a DPU job (also called *inference time*). Four execution phases for the DPU have been identified: **(1)** *Read instructions phase*: The instructions are fetched from the DRAM memory through the M^{INS} port. (In Section VI we demonstrate how using the OCM memory for instructions fetching helps reduce the pessimism of the analysis and slightly increases the performance). **(2)** *Read data phase:* The DNN model data (weights, biases, activations) and the input image are fetched from the DRAM memory through the M^{DATA} port. **(3)** *Write data phase:* The results of the computation are written to the DRAM memory through the M^{DATA} port. **(4)** *Elaboration phase:* It is defined as the time when the DPU has not yet completed the job and no activity on the bus is performed (i.e., the hardware accelerator is making progress by only computing). The response time of each DPU phase has been measured leveraging our hardware profiler. For instance, to measure the execution time of the read instructions phase, our hardware profiler tracks the time the DPU is active on the M^{INS} port, considering served and pending transactions. The hardware profiler stops the profiling when the DPU interrupt is raised, locally storing the profiled values that can be read by software as standard memory-mapped registers. Table II reports the minimum, average, and maximum recorded times for each of the DPU phases, accompanied by the measured total inference time, for the 1000 executions under analysis.

C. Observations

This extensive profiling campaign allows us to make some observations, which are later leveraged in Section IV-A to design an appropriate model of the DPU with the purpose of bounding response times during DNN acceleration (Section V). The observations follow:

1) The software-DPU interactions are limited to the DPU configuration: We developed a specific functionality in our hardware profiler to record the interactions between the Vitis AI control SW-task and the DPU during execution. We detected that the software-hardware interactions of the tested DNNs are limited to the configuration phase of the DPU. This means that no software-hardware interactions are present during the execution phase (inference) of the DNN model on the DPU. This consideration allows to reduce the complexity of the worst-case analysis proposed in Section V.

2) The bus activity of the DPU is constant and independent from the specific job: From the measurements we observed that, given a DNN model, the bus activity of the DPU does not vary job by job.

From the measurements we observed that, given a DNN model, the bus activity of the DPU does not vary job by job.

This suggests that the DPU implements the very same and predictable behavior given the structure of the network, the resolution of the input image, and the size of the outputs. This also means that the amount of instructions fetched, read data, and write data is fixed and known a priori. Table I reports the profiled bus activity for the DNN models under analysis.

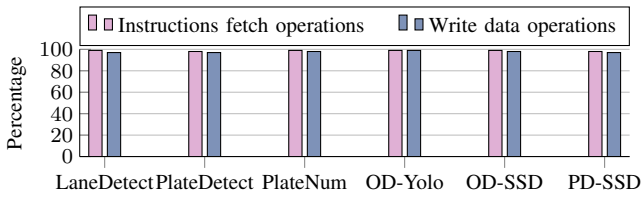

Fig. 2. Minimum measured percentage of the total inference time for which DPU instructions operations and write operations are overlapped with the DPU read data operations.

3) The DPU exhibits high parallelism in the execution of phases: By checking the results of Table II it is possible to notice how, for all the tested DNNs, the results for the read data phase and the total inference time are very similar. Also, the sum of the read instruction phase, read data phase, and write data phase is way higher than the total measured inference time. Guided by this observation, we developed a specific functionality in our hardware profiler to measure the parallel execution of **(i)** the instruction and write data phases of the DPU, with **(ii)** the DPU read data phase. The results are reported in Figure 2. The results showed how at least 97-98% of the execution of the former phases is overlapped with the one of the latter. This has been confirmed by also visually inspecting the bus activity using the Xilinx Integrated Logic Analyzer (ILA) — an excerpt of the waveform track of the bus signals is reported in Figure 3, showing the instruction and data read transactions performed by the DPU. Figure 3 shows how the data read operations (DATA PORT) and the instruction read operations (INSTRUCTIONS PORT) are operated in parallel by the DPU.

From these considerations, it is possible to conclude the following: **(i)** the DPU executes multiple phases in parallel, hence suggesting an internal pipelined implementation (as it is common for many FPGA-based accelerators); and **(ii)** the read data phase provides the dominant contribution to the total inference time, i.e., the instruction fetching phase and the write data phase are almost completely hidden due to the pipelined behavior. These conclusions motivate the definition of a series-parallel model for the DPU in the following (Section IV).

4) Limited fluctuations of response times: The results reported in Table II finally show how the fluctuations of the response times are quite limited. They are mainly attributed to the delays experienced at the memory controller to access the external DRAM.

IV. Modeling

This section presents a model for the DPU and the platform considered in this paper. The assumptions made in deriving the model are supported by the technical information available in the official documentation [19] for the platform under analysis (Xilinx Zynq Ultrascale+) and the profiling campaign discussed in Section III.

A. The DPU core model

This section presents an execution model of the DPU. Note that, although it targets a specific DNN accelerator, the model is general enough to be extended to cope with other high-performance, instruction-based hardware accelerators for DNN acceleration featuring autonomous memory access.

1) DPU per-job bus traffic: From Observation III-C2 (Section III) we know that the amount of data exchanged with the memory by the DPU is constant and predictable, i.e., it can be retrieved and does not change from DPU job to DPU job. We then model the DPU bus traffic as follows.

Read instructions phase. Each job the DPU issues at most N_R^{INS} read transactions through port M^{INS}, fetching Δ_R^{INS} data words corresponding to the instructions for the execution of the DNN model under evaluation. Port M^{INS} can issue at most $N_{\text{R,outs}}^{\text{INS}}$ outstanding transactions (i.e., pending at the same time).

Read data phase. Each job of the DPU issues at most N_R^{DATA} read transactions through port M^{DATA}, fetching Δ_R^{DATA} data words corresponding to the input data and DNN model parameters (i.e., bias, weights, and activations). Port M^{DATA} can issue at most $N_{\text{R,outs}}^{\text{DATA}}$ outstanding read transactions.

Write data phase. Each job of the DPU issues at most N_W^{DATA} write transactions through port M^{DATA}, writing Δ_W^{DATA} data words corresponding to the results. Port M^{DATA} can issue at most $N_{\text{W,outs}}^{\text{DATA}}$ write outstanding transactions.

To be as general as possible and obtain a model that can also be applied to other accelerators, no specific pattern of transactions is considered, i.e., transactions can be arbitrarily distributed within the time span in which a DPU job is pending.

2) DPU execution: From Observation III-C3 we know that the DPU overlaps the execution of multiple phases and that the contribution of the read data phase is dominant for the total inference time. For this reason, we adopt a parallel-series model for the DPU execution as illustrated in Figure 4.

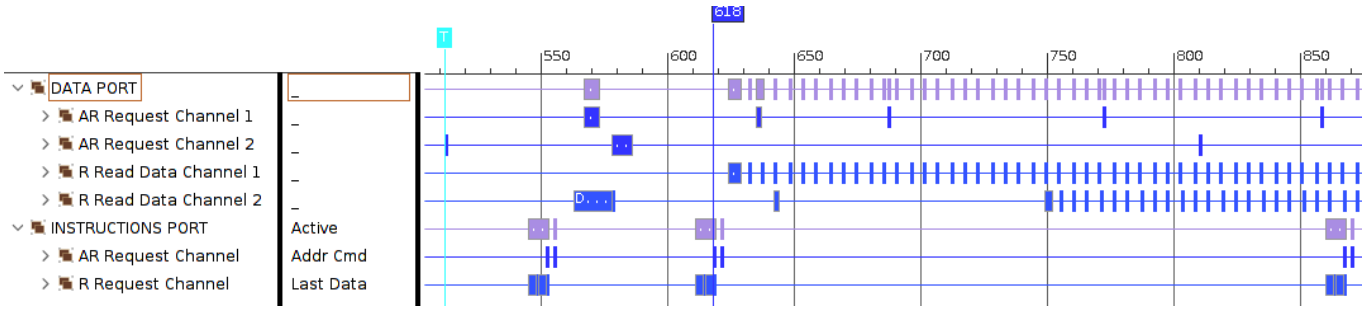

Fig. 3. A particular of the hardware waveform track captured using a Xilinx System ILA recording the bus behaviour of the DPU core when accelerating the Yolov3 object detection DNN for ADAS applications considered in this work.

Our execution model assumes that the DPU executes the read data phase in parallel with the instructions phase and the write data phase. The profiling campaign did not reveal any evident parallel execution for the instruction phase and the write data phase. Therefore, to be conservative and obtain a safer model, these two phases are considered to be serially executed. Finally, our model considers that the read data phase and the instructions and write data phase can be followed by some serial operations operated by the DPU. Note that our model admits that all such phases are not necessarily contiguously executed by the DPU core – the execution of each phase can be interrupted and resumed during the DPU job. The serial operations consist in the elaboration phase profiled in Section III, which is modeled as follows.

Elaboration phase. In this phase the DPU spends at most E_{time} time units and performs computations only, i.e., no bus interfaces is active for read/write activities.

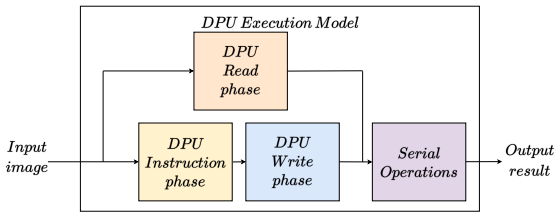

Fig. 4. The proposed execution model of the DPU core.

B. Interconnect model

As introduced in Section II-A, ports M^{DATA} and M^{INS} of the DPU are connected to the FPGA-PS interface. The configuration port S of the DPU is connected to the PS-FPGA interface. Following the results of the profiling campaign of Section III, we assume that the FPGA-PS interface and the PS interconnect do not introduce limits to the structure of the AXI transactions issued by the DPU. This means that the burst lengths of the transactions and the number of outstanding transactions issued by the DPU are not limited by the PS interconnect. Being the PS interconnect AXI-based, there are separate channels for the propagation and service of read transactions and write transactions. Thus, read and write transactions do not suffer mutual interference during propagation. Read and write transactions can instead suffer mutual interference when they are served by the DRAM memory controller — this effect is considered in the DRAM memory controller model proposed in Section IV-C. From the definition of the AXI standard, the propagation of address and data into the AXI interconnect is pipelined.

C. PS, DRAM memory controller, and OCM

The DRAM memory controller resides in the PS of the FPGA SoC platform and is shared between the processors and the DPU. The DRAM memory controller is conceptually split in two blocks: **(i)** the AXI interface block, which arbitrates the incoming AXI transactions received at its multiple AXI subordinate ports; and **(ii)** the DDR physical core block, which issues the corresponding read and write requests to the controller's physical layer, and eventually drives the DRAM memory by generating control and data signals. In commercial platforms, the internal architecture of the DDR physical core includes multi-level queue structures, managed with dedicated scheduling policies. These can include reordering for throughput optimization and efficiency [20], [21]. Unfortunately, the internals of the DDR physical core block of the Zynq Ultrascale+, including the scheduling policies and the queues structure, are not available to us (to the best of our records, they are not publicly disclosed). Given that our focus is on bounding the response times of accelerated DNN models running on the DPU, rather than reversing the behavior of commercial DRAM memory controllers, a fine-grained modeling of the DDR physical core block goes beyond the scope of this paper and it is not addressed here. Thus, a coarse-grained model of the DRAM-related delays is adopted; nevertheless, note that if the internals of the DDR controller were known, then our results can be refined (e.g., by adopting the results from [20], [21]). From the perspective of modules deployed into the FPGA fabric, and hence from the one of the DPU too, the address requests directed to the DDR memory controller are served in order (see [19], p. 440). Therefore, the order of the data read responses on the AXI data read channel follows the order of the address read requests granted at the AXI address read channel. This property is guaranteed by the DRAM Memory Controller AXI Interface block and is independent of the internal scheduling

policies of the DDR Physical core block, which may include internal reordering that affects the worst-case service time of a request. This property enables a fine-grained analysis of competing read transactions issued by the DPU. Conversely, no service order is guaranteed among transactions of different types. As such, the mutual interference between read and write transactions issued by the DPU cannot be accurately bounded, as it still significantly depends on the internals of the memory controller. Based on these considerations, we model the worst-case delays introduced by the PS and the memory controller as components that introduce the following (cumulative) delays on each transaction:

- $d^{\text{read}}_{\text{DRAM}}$ bounds the maximum time elapsed from (i) the sampling of a read transaction directed to the DRAM memory at the FPGA-PS interface to (ii) the availability of the first word of the corresponding data at the FPGA-PS interface *without accounting for the interference of other read transactions issued by the DPU*;
- $d^{\text{write}}_{\text{DRAM}}$ bounds the maximum time elapsed from (i) the sampling of the last word of data of a PS write transaction directed to the DRAM memory at the FPGA-PS interface to (ii) the availability of the corresponding write response at the FPGA-PS interface.

Note that, from the above definitions, the terms $d^{\text{read}}_{\text{DRAM}}$ and $d^{\text{write}}_{\text{DRAM}}$ also account for the worst-case memory interference generated by the processor cores in the PS. Refinements of the model to enable a more accurate analysis of the DRAM-related delays are discussed in Section IX.

The Zynq Ultrascale+ platform also includes an On-Chip Memory (OCM), a memory-mapped DRAM of size 256KB placed in the PS. From our profiling campaign it emerged that, in Vitis AI applications, the OCM is not used by the processors or the DPU. In the following, we show how the OCM can be used to improve the execution predictability of the DPU by hosting DPU instructions. Therefore, we model the OCM as a memory that is exclusively access for reading instructions by the M^{INS} manager port of the DPU. Similarly to the case of the DRAM memory, we model the worst-case delays introduced by the OCM as follows:

- $d^{\text{read}}_{\text{OCM}}$ bounds the maximum time elapsed from (i) the sample of a read transaction directed to the OCM memory at the FPGA-PS interface to (ii) the availability of the first word of the corresponding data at the FPGA-PS interface.

Note that, by definition, $d^{\text{read}}_{\text{DRAM}}$, $d^{\text{write}}_{\text{DRAM}}$, and $d^{\text{read}}_{\text{OCM}}$ include the propagation times introduced by the PS internal logic and the overall service time at the DRAM memory controller or the OCM. These parameters depend on the internals of the PS and can be quantified using the documentation provided by the SoC producer (when available) – again, as the documentation of the platform under analysis does not disclose such details [19], we experimentally profiled these parameters.

D. The Vitis AI control SW-task model

The DPU is setup and activated by a control software application SW^{DPU}. At the beginning of execution, SW^{DPU} configures the DRAM memory buffer for the execution of the DPU. Such a memory buffer contains the input for the DPU, such as the DPU instructions and the parameters of the DNN model under analysis and the input image. SW^{DPU} also prepares another memory buffer to host the results provided by the DPU. SW^{DPU} then configures the execution by accessing the internal registers of the DPU. Once the setup is completed, the DPU is activated and starts the inference of the DNN model under analysis, following the phases described in Section IV-A. At this point, SW^{DPU} suspends its execution, waiting for the interrupt of the DPU signaling completion.

V. Response-time analysis

This section presents a response-time analysis to bound the total inference time of DNN models when executed by the DPU. To this purpose, we adopt a bottom-up approach. The analysis leverages the parameters obtained by profiling reported in Section III. Although the values of these parameters proved to be very predictable, they can be anyway monitored at run-time so that response-time bounds can be deemed safe as long as the monitored values are below those obtained from profiling. Otherwise, an exception can be raised. The monitoring can be performed by an FPGA module similar to the hardware profiler introduced in Section III.

A. Bounding the response times of the DPU phases

1) Read transactions directed to the DRAM with no contention: According to the AXI standard [7], each AXI read transaction begins with the issuing of an *address request* and terminates with the sampling of the read data. By the AXI standard, the time each address request has to be held on the bus to be correctly issued is fixed and equal to d_{addr}, while the time each data word has to be held on the bus to be correctly sampled is fixed and equal to d_{word}. The following lemma bounds the response time of read transactions for both port M^{INS} and port M^{DATA}. In the former case, the lemma can be applied with $N_R = N_R^{\text{INS}}$ and $\Delta_R = \Delta_R^{\text{INS}}$, while in the latter case with $N_R = N_R^{\text{DATA}}$ and $\Delta_R = \Delta_R^{\text{DATA}}$.

Lemma 1. *In conditions of no contention from read transactions issued by the DPU on other ports, the response time of N_R read transactions issued by the DPU to read Δ_R data words on a port is bounded by:*

$$D^{NoCont}_{R,DRAM}(N_R, \Delta_R) = N_R \cdot (d_{addr} + d^{read}_{DRAM}) + \Delta_R \cdot d_{word}. \tag{1}$$

Proof. During the address phase, each address request R_{addr} of one of the N_R transactions is issued by a port of the DPU to be then sampled by the FPGA-PS interface. By the AXI standard, this phase lasts d_{addr} time units per transaction. R_{addr} is then propagated to the DRAM memory controller by the PS interconnect and eventually served by the DRAM memory controller. Once the latter provides the required data, the read data words are propagated to the FPGA-PS interface by the PS interconnect to be eventually sampled by the DPU. By the model of Section IV-C, the delay associated to this process is bounded by $d^{\text{read}}_{\text{DRAM}}$ per transaction. Hence the first term of

Eq. (1). The lemma follows by accounting for the time the Δ_R read data words have to be held on the bus to be sampled by the DPU (last term of Eq. (1)). □

2) Write transactions directed to the DRAM: Still, according to the AXI standard [7], each AXI write transaction begins with the issuing of an *address request*, proceeds with the transmission of the data to be written, and terminates with the reception of a *write response*. Times d_{addr} and d_{word} also hold for write transactions, while the time write responses must be held on the bus to be correctly sampled is fixed and equal to d_{bresp}. As stated in Section IV-C, contention for write transactions directed to the DRAM is all accounted for in the term $d^{\text{write}}_{\text{DRAM}}$, hence the following lemma bounds their response time.

Lemma 2. *The response time of the N_W^{DATA} write transactions issued by the DPU to write Δ_W^{DATA} data words via port M^{DATA} is bounded by:*

$$D_{DATA}^{W,DRAM} = N_W^{DATA} \cdot (d_{addr} + d_{DRAM}^{write} + d_{bresp}) + \\ + \Delta_W^{DATA} \cdot d_{word}. \tag{2}$$

Proof. According to the AXI standard [7], read and write requests have the same structure. Thus, the total time to issue write requests ($N_W \cdot d_{\text{addr}}$) follows as for Lemma 1. Each write request is followed by the corresponding data words to be written that, in total for all transactions, must be held on the bus for $\Delta_W \cdot d_{\text{word}}$ time units. Write transactions are propagated to the memory controller via the FPGA-PS interface. The memory controller eventually replies with a write response for each transaction. By the model of Section IV-C, the delay associated to this process is bounded by $d^{\text{write}}_{\text{DRAM}}$ per transaction. The lemma follows by noting that each write response, once arrived at the FPGA-PS interface, must be held on the bus for d_{bresp}. □

Note that the above lemmas are independent of the structure of the memory transactions, i.e., they hold independently of the burst length of each transaction. This property is particularly useful in our scenario where the DPU issues bus transactions with variable burst lengths (see Section III).

3) Bounding the interference for read transactions: In the stock configuration provided by Vitis AI, the two data ports of the DPU M^{INS} and M^{DATA} compete to read from DRAM. As mentioned in Section IV-C, read transactions issued by the DPU and directed to the DRAM are served in order. For this reason, the corresponding interference is not accounted for in the term $d^{\text{read}}_{\text{DRAM}}$ of our model as it can be more accurately bounded as follows.

Lemma 3. *The maximum interference suffered by read transactions issued on the M^{INS} port of the DPU due to read transactions issued by the DPU on the other port M^{DATA} is bounded by*

$$min\{N_R^{INS} \cdot N_{R,outs}^{DATA}, N_R^{DATA}\} \cdot d_{DRAM}^{read}. \tag{3}$$

Proof. Recall that, from the perspective of the DPU, the read transactions issued are served in order. Whenever a transaction is issued on port M^{INS} there can be at most $N^{\text{DATA}}_{\text{R,outs}}$ outstanding transactions (i.e., pending but not completed) issued before on port M^{DATA}. Hence, the total number of interfering read transactions is bounded by $N^{\text{INS}}_R \cdot N^{\text{DATA}}_{\text{R,outs}}$. Furthermore, since at most one inference instance of the DPU can be pending at the same time, the total number of interfering read transactions is also bounded by the total number of transactions issued on port M^{DATA}. Hence, the minimum of the two is still a valid bound. The times the address request and the data must be held on the bus (d_{addr} and d_{word}) do not contribute to the interference as ports M^{DATA} and M^{INS} are connected to the FPGA-PS interface with separate AXI links. The lemma follows by noting that each interfering transaction can take at most $d^{\text{read}}_{\text{DRAM}}$ to be served after receiving the FPGA-PS interface. □

In the very same way, the maximum interference suffered by the transactions to read data (i.e. issued on port M^{DATA}) due to the transactions to read instructions is bounded by

$$min\{N^{\text{DATA}}_R \cdot N^{\text{INS}}_{\text{R,outs}}, N^{\text{INS}}_R\} \cdot d^{\text{read}}_{\text{DRAM}}. \tag{4}$$

4) Bounding the response time of the read instruction and read data phases: Leveraging the above results it is now possible to bound the response time of both the read instructions and read data phases. For each phase, this is accomplished by summing the response time without contention from read transactions issued by the other phase (Lemma 1) and the contention bound of Lemma 3 and Equation (4).

It follows that the response time of the read instructions phase of the DPU is bounded by

$$D^{R,\text{DRAM}}_{\text{INS}} = D^{\text{NoCont}}_{R,\text{DRAM}}(N^{\text{INS}}_R, \Delta^{\text{INS}}_R) + \\ + min\{N^{\text{INS}}_R \cdot N^{\text{DATA}}_{\text{R,outs}}, N^{\text{DATA}}_R\} \cdot d^{\text{read}}_{\text{DRAM}}, \tag{5}$$

while the one of the read data phase is bounded by

$$D^{R,\text{DRAM}}_{\text{DATA}} = D^{\text{NoCont}}_{R,\text{DRAM}}(N^{\text{DATA}}_R, \Delta^{\text{DATA}}_R) + \\ + min\{N^{\text{DATA}}_R \cdot N^{\text{INS}}_{\text{R,outs}}, N^{\text{INS}}_R\} \cdot d^{\text{read}}_{\text{DRAM}}. \tag{6}$$

B. Bounding the total DPU response time

Theorem 1. *The total DPU response time is bounded by*

$$T_{DPU}^{DRAM} = max\{D_{DATA}^{R,DRAM}, D_{INS}^{R,DRAM} + D_{DATA}^{W,DRAM}\} + D_{elab}. \tag{7}$$

Proof. Following the parallel-series execution model introduced in Section IV-A2, the read data phase is executed in parallel with the instruction read and data write phases. Hence, these three phases are completed after at most the maximum of the time required to complete the first phase, and the one required to complete the other two phases. The former is bounded by Eq. (6), while the latter is bounded by Eq. (5) plus the bound implied by Lemma 2. Hence the first term of Eq. (7). The lemma follows by also summing the duration of the elaboration phase, which is modeled at the end of the parallel-series flow. □

It is worth observing that the above bound can be refined if more information about the internal behavior of the DNN accelerator is available, hence enabling a refinement of the parallel-series model of Section IV-A2.

VI. Improving the DPU execution predictability

This section presents a technique to improve the execution predictability of the DPU. From the above sections, we know that the DPU suffers auto-interference due to concurring transactions on the M^{INS} and M^{DATA}. Although this interference is explicitly bounded in the analysis of Section V-A4, it is still a relevant source of pessimism for the overall response-time bound.

As it is the case for other commercial SoC platforms, the Xilinx Zynq Ultrascale+ includes an OCM that can be accessed by FPGA modules though a dedicated interconnect in PS, as illustrated in Figure 1. Note that the path between the DPU and the DRAM memory controller and the path between the DPU and the OCM are parallel and independent (this can be confirmed by checking the official technical reference manual of the Zynq Ultrascale+ [19]). Given that our profiling campaign of Section III revealed that the size of DPU instructions is generally compatible with the size of the OCM and that the OCM is not used by Vitis AI applications, we investigate the opportunity of storing the DPU instructions on the OCM. This makes it possible to leverage the parallelism of the two memories (OCM and DRAM) and the corresponding bus paths, as well as tightening the response-time bound by suppressing the mutual interference generated by read transactions on ports M^{INS} and M^{DATA}.

In particular, by analyzing in detail the compiled `.xmodel` file produced by the Vitis AI compiler for all the tested DNN models for ADAS applications, we found that instructions represent just a little portion of the file.

Unfortunately, the structure of the `.xmodel` file is not publicly documented by Xilinx and hence instructions cannot be easily extracted to be placed in the OCM. Furthermore, the low-level driver for the DPU that sets the addresses of the memory buffers that contain the DPU instructions and data is not configurable and its source code is not publicly available. This means that it is not possible to directly configure the DPU with a different address for instruction fetching so that instructions are fetched from the OCM, while data are still fetched from the DRAM.

To solve this problem we developed a minimal custom FPGA module, which is presented next.

A. The DPU Instruction Dump - Address Translator IP

The *DPU Instruction Dump - Address Translator* (DICTAT) is a custom IP developed to allow the DPU fetching instructions from the OCM. DICTAT is placed between the M^{INS} of the DPU and the FPGA-PS interface. It has two functionalities: **(1)** Dumping the DPU instructions of a DNN model under analysis; and **(2)** Rerouting the read transactions issued by the DPU on port M^{INS} to target the OCM memory. These two functionalities are described next.

1) Dumping the DPU instructions: This step is executed only once (i.e., it is a setup step) for a given DNN model under analysis. From our profiling campaign, it emerged that SW^{DPU} places the DPU instructions in multiple small DRAM memory buffers and not in a single buffer as one may expect – this makes it non-trivial to dump the instructions. DICTAT is capable of sniffing the DPU instructions fetched by the DPU during execution by monitoring the bus traffic generated on port M^{INS}. Before starting the DPU execution, DICTAT is setup with a memory address pointing to a pre-allocated DRAM memory buffer where it contiguously writes all sniffed instructions. The sniffed instructions can then be easily exported into a dump file. By experiments, we confirmed that the instructions fetched by DPU jobs remain the same for a given DNN model, even considering different input images. This means that the dumped DPU instructions can be placed into the OCM just once – for instance, the dumped instructions can be loaded from the file to the OCM before the execution of the DNN model or in parallel with the configuration of the DPU. Indeed, still from the profiling campaign of Section III, we noted that the configuration time of the DPU is long enough to cover the memory copy of the small amount of instructions from the dump file to the OCM memory.

2) Rerouting M^{INS}'s read transactions to the OCM: Once the DPU instructions have been dumped and placed in OCM, DICTAT is configured with the base address of the OCM. Once the DPU starts the execution, DICTAT translates on-the-fly the read requests issued on port M^{INS}, directed to the default DRAM buffer, to transactions directed to the OCM. To do so, DICTAT keeps track of the execution phases of the DPU, using the interrupt it generates as a synchronization signal to restart the translation from the beginning of the OCM buffer.

DICTAT supports any burst length allowed by the AXI standard and is completely transparent to the DPU and the underlying FPGA SoC platform. To achieve high performance and minimal area footprint, DICTAT has been developed in HDL. Also, DICTAT proactively acts on read transactions so that it does not introduce any additional latency on the bus: this way, the original performance of the DPU is not affected.

B. Refining the analysis

DICTAT is capable of completely decoupling the service of bus transactions issued on port M^{INS} from those issued on port M^{DATA}. This means that the transactions issued on M^{DATA} do not interfere with the transactions issued on M^{INS}, and vice-versa. Therefore, it is possible to refine the response-time bound of Section V-A4.

The transactions issued to the OCM features strong similarities with respect to those issued to the DRAM. Differences only reside in the different path requests and data have to traverse.

By making similar considerations to the ones done in Section V-A1 (see Lemma 1) and considering the worst-case propagation and service time for the OCM ($d^{\text{read}}_{\text{OCM}}$, introduced in Sec. IV-C), the response time of the transactions to read instructions from the OCM is bounded by:

$$D^{\text{NoCont}}_{R,\text{OCM}}(N^{\text{INS}}_R, \Delta^{\text{INS}}_R) = N^{\text{INS}}_R \cdot (d_{\text{addr}} + d^{\text{read}}_{\text{OCM}}) + \Delta^{\text{INS}}_R \cdot d_{\text{word}}. \tag{8}$$

Given that, in the presence of DICTAT, no interference between the read instructions and read data phases are possible,

their response time is simply bounded by $D^{\text{NoCont}}_{R,\text{OCM}}(N^{\text{INS}}_R, \Delta^{\text{INS}}_R)$ and $D^{\text{NoCont}}_{R,\text{DRAM}}(N^{\text{DATA}}_R, \Delta^{\text{DATA}}_R)$, respectively. Hence, analogously as done for Theorem 1, the total DPU response time with instructions fetched from the OCM is bounded by:

$$T^{\text{DRAM+OCM}}_{\text{DPU}} = max\{ D^{\text{NoCont}}_{R,\text{DRAM}}(N^{\text{DATA}}_R, \Delta^{\text{DATA}}_R), \\ D^{\text{NoCont}}_{R,\text{OCM}}(N^{\text{INS}}_R, \Delta^{\text{INS}}_R) + D^{W,\text{DRAM}}_{\text{DATA}} \} + D_{\text{elab}}. \quad (9)$$

VII. Experimental evaluation

This section presents the results of an experimental evaluation that was conducted to assess the quality of the response-time bounds and the DPU performance when using DICTAT.

A. Experimental setup

The hardware platform and the software system used for the evaluation are the same as the profiling campaign of Section III.

Concerning the modules deployed on the FPGA, we created a custom hardware design that integrates **(i)** our clock-level multi-channel hardware profiler (see Section III), **(ii)** the DICTAT module presented in Section VI-A to enable the use of the OCM memory, **(iii)** a Xilinx System ILA [22] IP for model validation and debugging, and **(iv)** the stock Xilinx setup of a DPU accelerator. The presented results were obtained through the analysis of 1000 DPU executions for each of the considered DNN models. Each execution was tested on a randomly picked image taken from the Cityscapes ADAS dataset [16].

B. Profiling the platform

This first set of experiments aimed at profiling the timings of the FPGA SoC platform under analysis (Zynq Ultrascale+). They include the worst-case propagation and service times at the DRAM memory controller, i.e., $d^{\text{read}}_{\text{DRAM}}$ and $d^{\text{write}}_{\text{DRAM}}$, and the ones at the OCM memory, i.e., $d^{\text{read}}_{\text{OCM}}$ introduced in Section IV-C. The profiling also copes with the bus times d_{addr}, d_{word}, and d_{bresp}. The delays are profiled by both considering the tracks provided by the ILA we integrated in the system and by implementing specific functionalities in our custom hardware profiler introduced in Section III. The observed bus times (constant by the AXI standard) expressed in clk cycles are $d_{\text{addr}} = 1$, $d_{\text{bresp}} = 1$, and $d_{\text{word}} = 1$ for read transactions and $d_{\text{word}} = 2$ for write transactions (the differences in the value of d_{word} for read and write is most probably due to an internal design choice of the DPU). As introduced in Section IV-C, the cumulative worst-case delays for accessing the DRAM memory from the DPU depends on many different aspects, mostly related to the platform design choices and the activities performed by the processors in the PS. As one may easily expect, such parameters considerably influence the worst-case analysis. As a representative setting for our experiments, which do not have the scope of precisely characterizing the memory interference generated by the processors, we considered the stock Petalinux-based Vitis AI image executing a Vitis AI application that only accelerates DNNs with the DPU. Under this setting, we experimentally estimated these delays to be upper bounded by $d^{\text{read}}_{\text{DRAM}} = 40$, $d^{\text{write}}_{\text{DRAM}} = 30$, and $d^{\text{read}}_{\text{OCM}} = 40$ clock cycles, respectively, refereed to the clock domain of the DPU (330 MHz) – these are equivalent to 130, 96, and 130 clock cycles, respectively, at the DRAM memory since it features a highest clock rate (1067 MHz).

C. Stock DPU configuration with instructions in DRAM

This set of experiments aims at comparing the measured results for the stock Vitis AI configuration leveraging the DRAM memory for fetching data and instructions against the response-time bound presented in Section V. The comparison considers both the total inference time and each phase individually. The results are reported in Figure 5. As it can be noted from the figure, the response-time bounds are always safe and not excessively pessimistic. The figure also shows that the largest pessimism is introduced for the read instructions and read data phases. Despite the fact that the method presented in Section VI helps reduce this pessimism (other experiments on this follow), most of the pessimism is still attributed to the lack of detailed documentation for the PS interconnect and the DRAM memory controller. Indeed, note that **(i)** not all memory transactions directed to the DRAM actually incur the maximum delay of parameters $d^{\text{read}}_{\text{DRAM}}$ and $d^{\text{write}}_{\text{DRAM}}$; and **(ii)** the DRAM is capable of serving multiple transactions in parallel (i.e., those directed to different DRAM banks), while our analysis conservatively assumes that all transactions are serially served. Coping with these aspects is out of the scope of this work and we are confident that our analysis can significantly benefit from a fine-grained modeling of the DRAM memory controller and the PS interconnect.

D. Using DICTAT: DPU instructions in OCM

This set of experiments aims at evaluating the performance of the DPU when instructions are fetched from the OCM thanks to DICTAT and evaluating the corresponding response-time bounds of Section VI-B. The same experimental evaluation performed in the profiling campaign of Section III to obtain Table II has been repeated under this setting. The results are reported in Table III. Unfortunately, the size of the instructions fetched by the Lanedetect DNN model is larger than the size of the OCM memory available on the Ultrascale+ (256KB), hence we were not able to test this model with DICTAT. Note however that this limitation does not hold for newer FPGA SoC such as the Versal by Xilinx, which integrates a 4MB OCM [23].

The results reported in Table III show how, besides reducing the variability of the measured results with respect to Table II, leveraging the OCM also slightly improves the total performance in all of the tested DNN models. In the best case, the Plate Detect DNN shows a reduction in the total response time around 6%. The improvements on the read instruction phase are even more evident: in all of the considered scenarios the response times of this phase drops by at least 50%. The best results are obtained for the DNN exchanging the highest amount of data — the Object Detection DNNs based on Yolov3 and SSD show a reduction in the

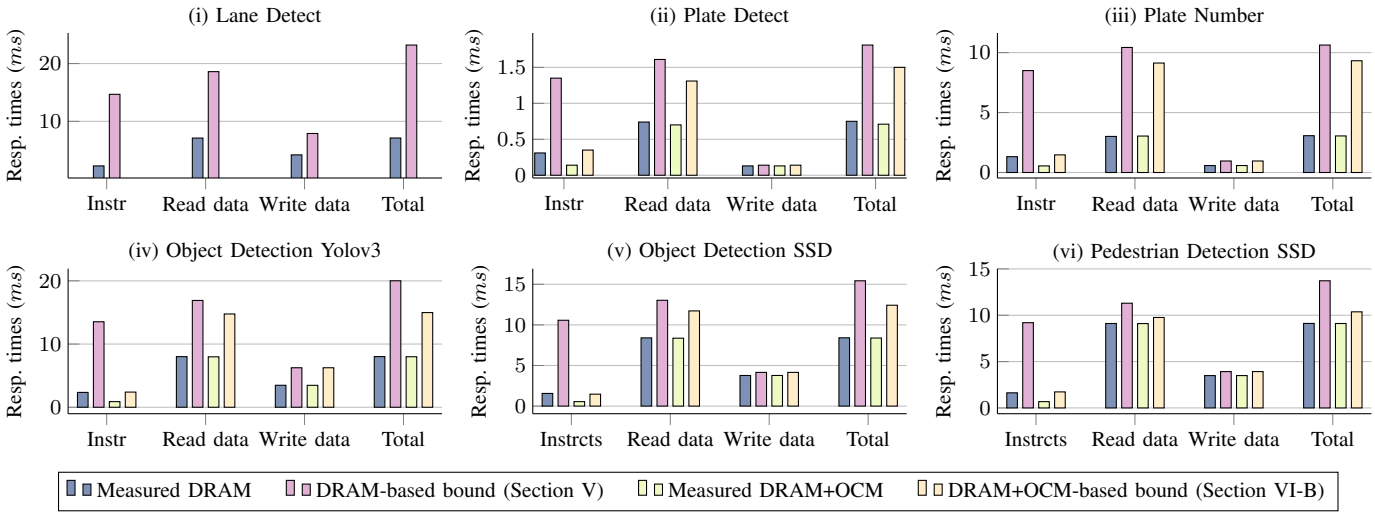

Fig. 5. Measured results for the DNN execution phases and total per-job inference times for the DNN models under analysis, compared with the bounds proposed in Section V and Section VI-B

maximum measured DPU instruction phase response time of 62% and 64%, respectively.

While improving the average performance of the DNN model is not the main purpose of this paper, this is an interesting opportunity for future work. The major result obtained by leveraging the OCM is the overall improvements in the pessimism of the analysis. The comparison of the response-time bounds against the measured times is also reported in Figure 5. The figure shows that all bounds are safe also in this case. As we expected, the bound with instructions fetched from OCM is tighter than the one with instructions fetched from DRAM.

Indeed, for all of the tested DNN models, the improved bound of Section VI-B shows a reduced pessimism of at least 12% on the total inference bound with respect to the other bound. In memory-intensive DNN models, the improved OCM+DRAM-based bound shows its best result – on the Yolov3 Object Detection DNN and on the SSD DNN for pedestrian detection it reduces the pessimism by 25% with respect to DRAM-based bound. These results certify how the interference generated by concurrently accessing the DRAM memory strongly impacts the pessimism of the system and how leveraging dedicated memories for fetching instructions can considerably reduce the pessimism of worst-case analysis.

TABLE III
MEASURED PER-JOB EXECUTION TIMES OF THE DPU PHASES AND TOTAL MEASURED PER-JOB INFERENCE TIMES LEVERAGING THE OCM TO HOST THE DPU INSTRUCTIONS.

DNN model inference times OCM (ms)		Instrc fetch	Read data	Write data	Pure Elab	Total Inference
Plate Detect	min	0.12	0.49	0.13	0.01	0.69
	avg	0.13	0.68	0.13	0.02	0.7
	max	0.14	0.7	0.14	0.2	0.71
Plate Num	min	0.53	2.45	0.56	0.01	3.03
	avg	0.54	2.99	0.57	0.1	3.04
	max	0.55	3.04	0.58	0.23	3.05
Object Detection (Yolov3 ADAS)	min	0.87	7.2	3.45	0.01	7.97
	avg	0.88	7.87	3.46	0.02	7.98
	max	0.89	7.98	3.47	0.59	7.99
Object Detection (SSD ADAS)	min	0.53	7.32	3.74	0.01	8.36
	avg	0.54	8.26	3.76	0.02	8.37
	max	0.55	8.36	3.77	0.34	8.38
Pedestrian Detection (SSD ADAS)	min	0.66	7.95	3.47	0.01	9.09
	avg	0.67	8.89	3.48	0.1	9.1
	max	0.68	9.10	3.49	0.7	9.11

TABLE IV
FPGA RESOURCE CONSUMPTION ON THE ZYNQ ZCU102.

IP	Xilinx DPU	DICTAT	HW profiler	Xilinx System ILA
LUT	108134	374	256	21877
CLB	203525	662	1837	33152
BRAM	518	0	0	128
DSP	1394	0	0	0

E. Resource consumption

Table IV compares the resource consumption of the hardware IPs integrated into the developed hardware profiling environment. The results show how DICTAT has a very limited resource consumption. This ensures an easy integration to fit area-constrained designs in platforms featuring a lower amount of FPGA resources.

VIII. RELATED WORK

To the best of our records, this is the first work that addresses time-predictability for FPGA-accelerated DNNs. Most efforts on analyzing and improving the real-time performance of DNNs were instead focused on GPU-based platforms. Wurst et al. [24] addressed the modeling of heterogeneous platforms for predicting the performance of autonomous driving applications on Nvidia GPU-based platforms. Gujarati et al. [25] demonstrated how the inference phase of DNN models can show deterministic performance. Zhou et al. [26] proposed a set of techniques to optimize the execution of DNN workloads on GPU in a real-time multi-tasking environment. Tan et al. [27] proposed a method to maximize the throughput of DNN applications on Nvidia Xavier platforms. Liu et

al. [28] and Bateni et al. [29] proposed DNN scheduling techniques for GPU SoC platforms.

More in general, execution predictability of FPGA-based platforms has been investigated by several authors. Gracioli et al. [30] proposed a set of techniques to support the execution of mixed-criticality applications upon FPGA SoC. Geier et al. [31] proposed a methodology for monitoring performance metrics of real-time systems executing on FPGA SoC platforms. Restuccia et al. [32], [33] and Pagani et al. [34] proposed FPGA devices to predictably control the bus traffic generated by hardware accelerators. Efforts have also been spent to analytically bound the delay experienced by AXI bus transactions issued by hardware accelerators on FPGA [35].

Finally, while this work focused on a commercial accelerator for DNNs, it is worth mentioning that many other research efforts have been spent to propose frameworks and automatic tools for the generation of DNN hardware accelerators for FPGA platforms, addressing high performance [36]–[38], the constraints of the FPGA platform [39], [40], and power efficiency [41].

IX. Discussion, Limitations, and Future Work

This section discusses the novelty of this work with respect to commercial tools available for profiling the DPU execution, its limitations, and some relevant future directions for further research on the topic. First of all, it is worth remarking that this work aims at bounding the inference time of DNN models leveraging hardware acceleration on commercial FPGA SoC platforms. This completely differs from the goals of commercial tools available for performance analysis of the DPU, such as the Xilinx Vitis AI profiler tool, which focus on the average performance of the DPU core. Also, the Xilinx System ILA would not be able to extract the metrics reported in Section III. In particular, the Xilinx System ILA is intended for capturing continuous waveform tracks after the trigger of a specific event. Capturing the number of per-job transactions issued by the DPU and the data words exchanged with the memory required the development of custom functionalities analyzing the handshake signals of the AXI bus. Such capabilities are beyond those offered by the Xilinx System ILA. Considerable research and engineering efforts were required to perform modeling and analysis of the DPU core and the system architecture. Despite these efforts, this work comes with several limitations and can benefit from several further improvements. Some relevant research directions for future work are discussed next.

1) Refining the DPU model: The distribution of the DPU core as a closed-source IP makes challenging the derivation of an accurate model. The experimental results of Section VII demonstrate how our model obtained via profiling enabled a safe analysis to bound the execution time of the DPU. However, it is worth noting how the proposed worst-case analysis directly depends on the accuracy of the DPU model. As such, it can significantly benefit from a refined DPU model taking into account detailed information on the DPU internals (e.g., the RTL code of the DPU, a more detailed description of the internal architecture of the DPU, etc.). Thus, we plan to focus on the derivation of more accurate models for the DPU in the future, both by performing further profiling and in the light of any future release of more detailed documentation provided by the official vendor channels.

2) DRAM memory parallelism: Modern DRAM memory controllers offer a considerable degree of parallelism in accessing multiple memory banks. This aspect has been considered in previous works to reduce the pessimism in the worst-case timing analysis of memory accesses [20], [21]. The analysis provided in this paper is pessimistic as it does not consider this aspect due to the reasons mentioned in Section IV-C. As future work, we plan to extend the analysis to explicitly consider parallelism in accessing DRAMs. This could allow obtaining tighter delay bounds.

3) Interference generated by the PS: A fine-grained model of the DRAM memory controller and PS is beyond the scope of this paper. Indeed, we focused on the proposal of a model and an analysis of the DPU core using a coarse-grained characterization of DRAM-related delays (Section IV-C). In particular, the approach adopted in this paper does not explicitly analyze the memory traffic generated by the processor cores in the PS, which is instead pessimistically accounted for in the terms $d^{\text{read}}_{\text{DRAM}}$ and $d^{\text{write}}_{\text{DRAM}}$. Our analysis can be extended in future work to cope with an accurate model for the DRAM memory controller [20], [21] and the explicit consideration of the traffic generated by software workload running in the PS.

4) Extension to other DNN hardware accelerators: The model and analysis proposed in this paper are mainly focused on the DPU provided by Xilinx with the Vitis AI framework. Nevertheless, we believe that a similar approach can be adopted to other instruction-based FPGA accelerators for DNNs that feature autonomous memory access. As future work, we also plan to extend our approach to support popular open-source dataflow DNN accelerators [8], [42]. This step is facilitated by the widespread usage of the AXI bus, which is the de-facto standard for communication for modern hardware accelerators.

X. Conclusions

This paper studied time-predictability in the execution of DNNs accelerated with the DPU FPGA accelerator by Xilinx. A profiling campaign revealed a rather regular execution behavior of the DPU. A model and a response-time analysis have been proposed to characterize the worst-case timing performance of DPU-accelerated DNNs. The DICTAT FPGA module has also been proposed to allow the fetching of DPU instructions from the OCM, improving the execution predictability and the response-time bounds. Experimental results confirmed the effectiveness of the response-time bounds and the execution scheme based on DICTAT.

XI. Acknowledgment

This work has been supported by the EU H2020 project AMPERE under the grant agreement no. 871669.

REFERENCES

[1] D. Casini, A. Biondi, and G. Buttazzo, "Timing isolation and improved scheduling of deep neural networks for real-time systems," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1760–1777, 2020.

[2] Q. Huang, D. Wang, Z. Dong, Y. Gao, Y. Cai, T. Li, B. Wu, K. Keutzer, and J. Wawrzynek, "Codenet: Efficient deployment of input-adaptive object detection on embedded FPGAs," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 206–216.

[3] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith, "Avoiding pitfalls when using nvidia GPUs for real-time tasks in autonomous systems," in *Proceedings of the 30th Euromicro Conference on Real-Time Systems*, 2018.

[4] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna, "Dissecting the cuda scheduling hierarchy: a performance and predictability perspective," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 213–225.

[5] K. O'Neal and P. Brisk, "Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey," in *2018 IEEE Computer SoCiety Annual Symposium on VLSI (ISVLSI)*, 2018, pp. 763–768.

[6] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–8.

[7] *AMBA® AXI™ and ACE™ Protocol Specification*, ARM, aRM IHI 0022D.

[8] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," ser. FPGA '17, 2017.

[9] *CHaiDNN official github.*, Xilinx, https://github.com/Xilinx/chaiDNN.

[10] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[11] *Vitis AI User Guide*, Xilinx, uG1414v1.3.

[12] S. Lee, J. Kim, J. S. Yoon, S. Shin, O. Bailo, N. Kim, T.-H. Lee, H. S. Hong, S.-H. Han, and I. S. Kweon, "Vpgnet: Vanishing point guided network for lane and road marking detection and recognition," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1965–1973.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 21–37.

[15] *Petalinux Tools Documentation*, Xilinx, uG1144.

[16] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Is your bus arbiter really fair? restoring fairness in AXI interconnects for FPGA SoCs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, p. 51, 2019.

[18] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "AXI hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in FPGA SoC," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[19] *Zynq UltraScale+ - Technical Reference Manual, UG1085*, Xilinx.

[20] M. Hassan and R. Pellizzoni, "Bounding dram interference in cots heterogeneous mpSoCs for mixed criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2323–2336, 2018.

[21] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "A holistic memory contention analysis for parallel real-time tasks under partitioned scheduling," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 239–252.

[22] *System Integrated Logic Analyzer v1.0*, Xilinx, 2017, pG261.

[23] *Versal ACAP Technical Reference Manual*, Xilinx, uM011.

[24] F. Wurst, D. Dasari, A. Hamann, D. Ziegenbein, I. Sañudo, N. Capodieci, M. Bertogna, and P. Burgio, "System performance modelling of heterogeneous hw platforms: An automated driving case study," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 365–372.

[25] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.

[26] H. Zhou, S. Bateni, and C. Liu, "S$\hat{3}$DNN: Supervised Streaming and Scheduling for GPU-Accelerated Real-Time DNN Workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 190–201.

[27] S. Tan, E. Jeong, J. Kim, J. Lee, and S. Ha, "Acceleration of deep learning applications by pipelining on nvidia jetson agx xavier," 2020.

[28] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 319–332.

[29] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu, "Co-optimizing performance and memory footprint via integrated CPU/GPU memory management, an implementation on autonomous driving platform," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 310–323.

[30] G. Gracioli, R. Tabish, R. Mancuso, R. Mirosanlou, R. Pellizzoni, and M. Caccamo, "Designing mixed criticality applications on modern heterogeneous mpSoC platforms," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[31] M. Geier, M. Brändle, D. Faller, and S. Chakraborty, "Debugging FPGA-accelerated real-time systems," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 350–363.

[32] F. Restuccia, A. Biondi, M. Marinoni, and G. Buttazzo, "Safely preventing unbounded delays during bus transactions in FPGA-based SoC," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 129–137.

[33] F. Restuccia, A. Meza, and R. Kastner, "Aker: A design and verification framework for safe and secure SoC access control," in *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2021.

[34] M. Pagani, E. Rossi, A. Biondi, M. Marinoni, G. Lipari, and G. Buttazzo, "A bandwidth reservation mechanism for AXI-based hardware accelerators on FPGAs," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[35] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[36] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 73–82.

[37] 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.

[38] H. Fang, Z. Mei, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu, "Encoding, model, and architecture: systematic optimization for spiking neural network in FPGAs," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[39] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "DNNbuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[40] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "DNNexplorer: a framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator," in *Proceedings of*

the 39th International Conference on Computer-Aided Design, 2020, pp. 1–9.

[41] Q. Sun, T. Chen, J. Miao, and B. Yu, "Power-driven DNN dataflow optimization on FPGA," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[42] B. B. Seyoum, M. Pagani, A. Biondi, S. Balleri, and G. Buttazzo, "Spatio-temporal optimization of deep neural networks for reconfigurable FPGA SoCs," *IEEE Transactions on Computers*, 2020.