WCET Estimation for CNN Inference on FPGA SoC with Multi-DPU Engines

Wei Zhang, Yunlong Yu, Xiao Jiang, Nan Guan, Naijun Zhan, and Lei Ju

***Abstract*—The Deep Learning Processor Unit (DPU) released in the official Xilinx Vitis AI toolchain stands as a commercial off-the-shelf solution tailored for accelerating convolutional neural network (CNN) inference on Xilinx FPGA devices. While most FPGA accelerator focus on high performance and energy-efficiency, analyzing the worst-case execution time (WCET) bound is essential for using CNN accelerations in real-time embedded systems design. In this work, we show that in a multi-DPU environment, the observed worst-case inference time for a CNN inference task could become 3X larger w.r.t. the best case inference time, which prompts the prominent importance of a static timing analysis for FPGA-based CNN inference. We propose, to the best of the authors' knowledge, the first static timing analysis framework for CNN inference in a multi-DPU environment. The proposed framework introduces a generalized timing behavior model for shared bus arbitration and memory access contention between parallel running DPU engines. Additionally, it incorporates a fine-grained memory access contention analysis that takes into account the characteristics of deep learning applications. For a single-DPU environment, the analysis result is 27% tighter in average compared with the state-of-the-art results. Furthermore, our proposed method produces relatively tight estimated results in the multi-DPU environment.**

***Index Terms*—FPGA, static timing analysis, memory contention, WCET estimation**

I. Introduction

With the rapid development of machine learning technology, Convolutional Neural Networks (CNNs) have been widely adopted in timing critical embedded systems. For example, multiple CNN models are used in the open autonomous driving system Apollo for perception tasks including semantic segmentation, obstacle detection, and so on [1]. In order to improve the performance of computation and memory intensive CNN inference on the resource constrained embedded devices, heterogeneous embedded computing platforms become the industrial practice for smart embedded system design. Typical heterogeneous processors used for CNN inference including general purpose graphics processing units (GPGPUs), field-programmable gate arrays (FPGAs), and neural processing units (NPUs).

Wei Zhang, Yunlong Yu, and Xiao Jiang are with the School of Cyber Science and Technology, Shandong University, Qingdao, China, and also with Quan Cheng Laboratory, Jinan, China.

Nan Guan is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China.

Naijun Zhan is with the School of Computer Science and Key Laboratory of High Confidence Software Technology, Peking University, Beijing, China, and also with ZGC Lab. Beijing, China.

Lei Ju is with Quan Cheng Laboratory, Jinan, China (Corresponding author: julei@sdu.edu.cn).

Among different types of heterogeneous processors, FPGA demonstrates programmability and promising energy efficiency. Many FPGA-based accelerators have been proposed for CNN inference [2]. With application-specific hardware design and ingenious software and hardware co-optimization techniques, the performance and energy efficiency of FPGA-based CNN inference accelerators have improved dramatically over the last decade. In December 2019, Xilinx has officially released the Deep Learning Processing Unit (DPU) with the Vitis AI framework [3]. Xilinx DPU is a programmable engine dedicated for convolutional neural network inference, which is built on a specialized instruction set to facilitate the fast deployment of efficient CNN inference on various Xilinx FPGA devices. Xilinx provides IPs for different DPU architectures, including B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096. In general, larger architecture provides better inference performance, at the cost of higher FPGA hardware resource usage. Furthermore, several DPUs can be deployed simultaneously on an FPGA chip with adequate logic and on-chip memory resources. Each DPU executes a CNN inference task independently (with possibly different CNN models), which allows parallel execution of multiple CNN inference tasks. For example, at most 3 B4096 DPU architecture IPs can be fit into the Xilinx ZCU102 evaluation board with the XCZU9EG Zynq Ultrascale+ MPSoC device.

Soundly bounding the worst-case execution time (WCET) is crucial for design of time critical embedded systems. Recently, some machine learning-based methods [4], [5] are proposed to predict the execution time for CNN tasks on DPU. However, these methods focus on producing an accurate average execution time estimates rather than generating a safe upper bound on the execution time, which hinders their direct application in designing time-critical systems. While static timing analysis techniques that yields a sound WCET bound have been well-studied for CPU-based systems [6], [7], little work has investigated the worst case timing behavior of CNN inference on FPGA. The first static timing analysis framework for DPU-based CNN inference has been proposed in [8]. However, it targets on a single DPU environment.

In this paper, we first show that for a given CNN model, the DPU-based inference time is very stable in a single-DPU environment (i.e., across different input images). It meets the expectation since the computation workload remains the same for input images with fixed resolution, and the intra-task memory contention follows a similar pattern across inputs. However, we then demonstrate that in a multi-DPU environment, where independent CNN tasks are executed on different DPU engines, the variance between the best-case and

worst-case inference time of a single CNN inference task is up-to 300%, which prompts the prominent importance of a static timing analysis for CNN inference on FPGA with multi-DPU engines. In general, since each DPU core has dedicated hardware resource to perform computation, the variance on the per-image inference time is mainly caused by uncertain shared memory access behaviors in a multi-DPU environment.

Contribution. This paper proposes a static timing analysis framework for CNN inference in a multi-DPU environment. The technical contributions are as follows.

- We perform a set of micro-benchmark studies to investigate the memory access behavior of DPU-base CNN inference. Based on the study, we construct a generalized timing behavior model for shared bus arbitration and memory access contention, which allows deployment of arbitrary number of DPU engines and port connection configurations.
- Based on the above-mentioned memory timing behavior model, we propose a static timing analysis framework for DPU-based CNN inference. For single-DPU environment, our estimated bound is 27% tighter compared with the state-of-the-art results. Moreover, to the best of the authors' knowledge, this is the first framework that supports WCET analysis of CNN inference in the multi-DPU environment.
- We show that the memory configuration on the DPU ports has distinct impact on both the measured and estimated CNN inference time in the multi-DPU environment. The default port configuration fails to provide the optimal estimated WCET bound. On the other hand, our analysis framework allows the system designer to find the WCET of CNN inference under different DPU memory configuration, which leads to better system resource provisioning.

II. Background

A. FPGA SoCs and Xilinx DPU

FPGA is a hardware-programmable device with a high integration of resources, which is widely considered as an accelerator for computationally-intensive applications because of its incomparable parallel acceleration potential. Unlike GPUs, FPGAs have a flexible hardware configuration, and provide a better performance per watt [9]. Owing to such advantages, FPGAs have been extensively used for accelerating DNNs [10]–[12].

FPGA is often combined with one or more CPU processors to be integrated as an FPGA SoC. A schematic diagram of a FPGA SoC is shown in Figure 1. It integrates a processing system (PS) and programmable logic (PL). PS and PL communicate with each other via AXI protocols for data transfer between the PL and the PS [13].

The Xilinx Deep Learning Processing Unit (DPU) is a configurable computation engine for CNNs on an FPGA SoC, which makes good use of the features of FPGA SoCs. The DPU is deployed in the PL, and connect to the PS via three different AXI ports, `M_AXI_INS`, `M_AXI_DATA0`, and `M_AXI_DATA1`, for transferring instructions and data words.

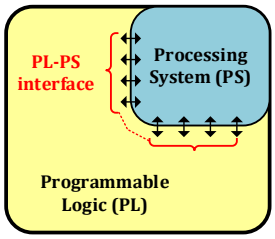


Fig. 1: A schematic diagram of FPGA SoCs.

The three interfaces of a DPU need to be connected to PL-PS interfaces to access the DDR memory. `M_AXI_DATA0` and `M_AXI_DATA1` are 128-bit interfaces and are responsible for data transmission, while `M_AXI_INS` is 32-bit interface and is responsible for instruction transmission.

There are several DPU architectures, including B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096, for users to choose from, each architecture of DPU requires different on-chip resources and provides different computing power. The Xilinx DPUCZDX8G IP is named with its peak operations per cycle. For instance, B4096 indicates that it can theoretically perform 4096 (multiplication and accumulation) operations per cycle. In general, a larger DPU core provides better performance, at the cost of higher FPGA resource usage. We show the resource usage of different DPU architectures in Table I. Users can change DPU configurations based on the available resource of the platform and computational requirement before deployment.

TABLE I: Resource Usage of Different DPU Architectures.

DPU Arch (PP*ICP*OCP)	BRAM	URAM to BRAM	DSP
B512(4*8*8)	73.5	4	78
B800(4*10*10)	91.5	2.25	117
B1024(8*8*8)	105.5	4	154
B1152(4*12*12)	123	2.75	164
B1600(8*10*10)	127.5	2.25	232
B2304(8*12*12)	167	2.75	326
B3136(8*14*14)	210	3.25	436
B4096(8*16*16)	257	3.75	562

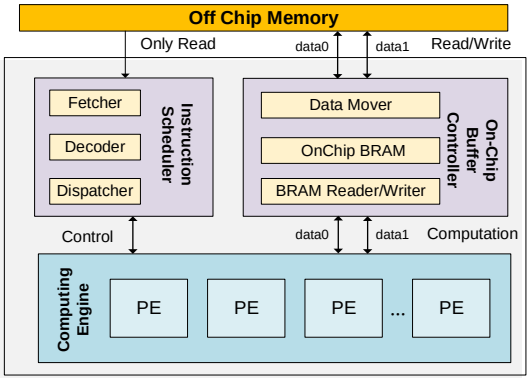


Fig. 2: DPU Hardware Architecture.

The hardware architecture of DPU is shown in Figure 2. The user is able to write high-level language programs on the PS side to control the DPU core to run the user-specified model through a number of drivers and APIs. A

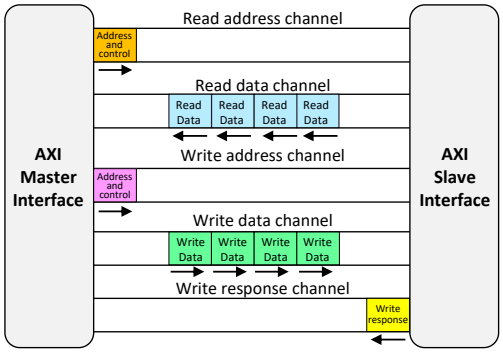


Fig. 3: Different channels between AXI master interface and slave interface.

CNN model must be compiled into a `.xmodel` file by Vitis AI [3] before running on a DPU. After start-up, the DPU fetches instructions from the off-chip memory to control the operation of the computing engine. On-chip memory is used to buffer input, intermediate, and output data to achieve high throughput and efficiency and reduce the external memory bandwidth consumption. A deep pipelined design is used for the computing engine. The processing elements (PEs) take full advantage of the fine-grained building blocks such as multipliers, adders, and accumulators in Xilinx devices.

The performance of a DPU highly depends on the memory access latency and its own computing capability. The computing capability is determined by the allocated resources and the system frequency, which are inherent to the DPU's architecture. When a DPU is deployed on an FPGA, its computing power remains fixed since it has dedicated computing resources. Consequently, the variation in the execution time primarily stem from the unpredictable memory access latency.

B. Advanced eXtensible Interface(AXI)

AXI is the most important part of the Advanced Microcontroller Bus Architecture (AMBA) protocol proposed by ARM [14]. The AXI bus has five channels: read address channel, write address channel, read data channel, write data channel, and write response channel. Each channel is unidirectional and follows the same handshake mechanism, relying on the VALID and READY signals to complete a single exchange of information which we call a transfer. There can be multiple data transfers per address, this is called a burst. A transaction is an entire burst of transfers, containing an address transfer, one or more data transfers, and a response transfer for write transactions.

An AXI read transaction requires multiple transfers on 2 read channels. First, an address request is sent from the master to the slave on the read address channel to set the address and some control signals. Then the data for this address is transmitted from the slave to the master on the read data channel. An AXI write transaction requires multiple transfers on the 3 write channels. First, an address request is sent from the master to the slave on the write address channel. The data for this address is then transmitted from the master to the slave on the write data channel. Finally, the write response is sent from the slave to the master on the write response channel to indicate whether the transmission is successful. The data flow on different channels between the master port and the slaver port are presented in Figure 3.

III. Motivation

Bounding the worst-case execution time (WCET) is crucial in real-time system design, especially when the execution time of the real-time tasks is unstable. In this section, we perform a set of experiments to measure the execution time of CNN inference on DPU to reveal its timing behavior.

The experiment is conducted on Xilinx ZCU102 evaluation board with Zynq UltraScale+ XCZU9EG MPSoC. Firstly, we configure a B4096 DPU, and execute `Mobilenetv2` [15] on it. We run `Mobilenetv2` 20000 times with random images with the same dimension from imageNet dataset [16] and measure the execution time for each inference. Experimental results in Figure 4 show that, the minimum inference time per image is 5.56ms, while the maximum inference time is 5.81ms. The results indicate a relatively small variation in inference time, with a difference of less than 7% during single DPU execution. Similarly, when performing `Yolov3` [17] inference with Cityscapes [18], we observe a 2.49% variation in inference time between the best and worst measurements (i.e., 8.867 and 9.093, respectively) during the 20000 iterations. Therefore, it can be observed that CNN inference in a single DPU environment exhibits a highly stable measured execution time.

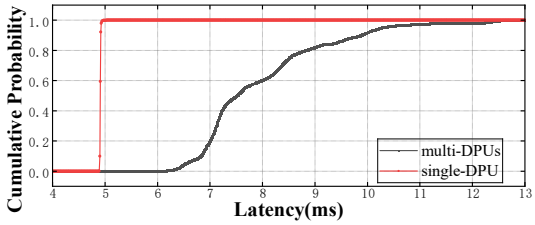


Fig. 4: Inference time distribution of Mobilenetv2 in the single-DPU and multi-DPU environments.

Most modern autonomous systems generally execute multiple CNN applications in parallel for different tasks. For instance, in the open autonomous driving system Apollo, several DNNs need to be executed in parallel [19] to perform various tasks including planing, localization, perception, etc. Xilinx provides the capability to deploy multiple DPUs to execute different CNN instances concurrently. In general, static timing analysis for the inference time of a particular CNN inference task in a multi-DPU environment can be a very challenging problem, due to the shared resource contentions between different DPUs.

To demonstrate the timing analysis challenge in a multi-DPU environment, we deploy three B4096 DPUs concurrently on the ZCU102 platform (i.e., maximum number of B4096 DPU cores can be deployed on ZCU102). For clarity, we refer the three DPUs as `DPU1`, `DPU2` and `DPU3`, respectively. We follow the official guidelines [20] to configure each DPU, ensuring proper connectivity and interface setup. Specifically, the

instruction interfaces of the three DPUs are interconnected and connected to S_AXI_LPD. As for the DPU data interfaces, the connections are as follows:

- DPU1: data0 to S_AXI_HP0, data1 to S_AXI_HP1
- DPU2: data0 to S_AXI_HP2, data1 to S_AXI_HP3
- DPU3: data0 to S_AXI_HPC0, data1 to S_AXI_HPC1

We conducted experiments running two instances of YOLOv4 [21] and one instance of Mobilenetv2 on DPU1, DPU2, and DPU3, respectively. We continuously process $20,000$ input images using Mobilenetv2 on DPU1 while simultaneously running YOLOv4 on DPU2 and DPU3 with different sets of images. The distribution of the inference time of Mobilenetv2 in the multi-DPU environment is depicted in Figure 4, where the maximum execution time is approximately 200% greater than the minimum observed inference time.

The examples presented above highlight the significance of timing analysis for CNN inference in a multi-DPU environment, particularly in the context of real-time system design using DPUs. However, the state-of-the-art worst-case analysis [8] for CNN inference only targets on the single-DPU environment and does not support to produce a WCET bound in a multi-DPU environment. Unlike the single-DPU scenario, measurement based WCET estimation might fail to find a safe bound for individual CNN inference tasks running on each DPU due to the huge variance between the best and worst inference time per input image. During the CNN inference, DPU accesses data from the memory and perform the computation on its dedicated PEs. Note, the computation phase may overlap with the memory access phase. For a given CNN model and input image size, since the Multiply-Accumulate (MAC) operations are exact and the FPGA-based DPU engine has very regular clock-level timing behavior, the computation cost (including access latency of the FPGA on-chip block RAMs) is deterministic. Our results shown in Figure 4 also demonstrate the stable inference time over different input images with same dimension in a single-DPU environment, as the off-chip main memory access contention behavior between different ports of a DPU are very similar across different input images.

In a multi-DPU environment, the main memory access contention cost becomes uncertain due to non-deterministic interleaving among different DPU engines. As a result, effectively bounding the shared memory contention cost is critical for obtaining a static worst-case inference time.

IV. System Model

This section first presents the hardware model of Xilinx DPU on FPGA SoC in section IV-A. To simplify the presentation, we use Xilinx ZCU102 to illustrate the hardware model. Note that the proposed method generalizes the hardware architecture and can also be extended to deal with DPUs on other FPGA SoCs. Then the software model of DPUs is also presented in section IV-B.

A. Hardware Model

DPUs are deployed in the PL of the FPGA, and each DPU contains three ports, two data ports, and one instruction port, to access data/instructions from the memory, respectively. These ports are connected to the PL-PS interfaces through the AXI interconnects.

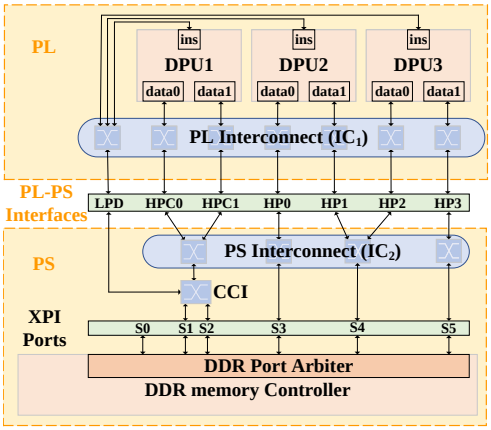


Fig. 5: The system architecture.

In the case of Xilinx ZCU102, there are 7 PL-PS interfaces available, LPD, HPC0, HPC1, HP0, HP1, HP2, and HP3 as shown in Figure 5. Different PL-PS interfaces will access data from different XPI ports through a PS interconnect and a Cache Coherent Interconnect(CCI). Then, these XPI ports are arbitrated in a round robin mode by default to access the DDR memory. Thus, when a transaction is issued by a DPU, it may contend with other transactions on at most 3 interconnects (i.e., PL interconnect, PS interconnect, and CCI), as well as the DDR port arbiter. As discussed in Section III, contentions on these interconnects and DDR port arbiter lead to the unpredictable timing behavior of DPUs. A detailed analysis of such contentions is presented in the section V.

1) Interconnect: When the number of data ports and instruction ports exceeds the available PL-PS interfaces, an interconnect is used to arbitrate memory access transactions. An interconnect contains several master ports and several slave ports, where the master ports are responsible for sending requests of data and slave ports receive and process the requests from the master ports. When the counts of master ports exceeds the counts of slave ports, the interconnect typically employs an arbitration scheme to determine which master port is granted access to the shared resources at any given time. For the PL interconnect, the master ports are connected to DPUs' data ports or instruction ports while the slave ports are connected to the PL-PS interfaces.

In the case of Xilinx ZCU102, which has 7 PL-PS interfaces, it is capable of deploying 3 B4096 DPUs. These 3 DPUs together have a total of 6 data ports and 3 instruction ports to share only 7 PL-PS interfaces. By the DPU configuration provided by Xilinx [3], 3 instruction ports are connected to the LPD interface via an AXI interconnect, while other ports have their own PL-PS interfaces as shown in Figure 5. Consequently, in the worst-case, when a DPU fetches an instruction from the DDR memory, it needs to wait for all the transactions that have been issued on the corresponding PL-PS interface. Similarly, on the PS interconnect, a transaction must also wait transactions on other master ports. The CCI

is a special case, as it contains two master ports and two slave ports, transactions issued on it do not need to wait for other transactions from other ports, but may lead to additional memory access latency.

The timing behavior of DPUs when executing in isolation shown in section III implies that, the memory access latency is relatively stable if there are no contentions on these interconnects. Based on this observation, we firstly measure the memory access latency without taking into account any contentions that may arise on the interconnects or the DDR port arbiter. Subsequently, we statically bound the additional execution time that is caused specifically by these contentions.

2) PL-PS interface: A Xilinx FPGA SoC typically contains three types of PL-PS interfaces, each with different memory access times. DPUs with the same architecture connected with different types of PL-PS interfaces have different inference time. Generally, HP offers the shortest memory access time, followed by HPC, and finally LPD.

We conduct experiments to explore the influence of different memory port configurations on the execution time of DPU applications. Specifically, we deploy a B4096 DPU on the FPGA SoC and test it with three different memory port configurations:

1) conf 1: instruction port is connected with `S_AXI_HP3`, while two data ports are connected with `S_AXI_HP0`.
2) conf 2: instruction port is connected with `S_AXI_HP3`, while two data ports are connected with `S_AXI_HPC0`.
3) conf 3: instruction port is connected with `S_AXI_HP3`, while two data ports are connected with `S_AXI_LPD`.

For each memory configuration, we run three different CNNs, YOLOv3, YOLOv4, Mobilenetv2 for 5000 times and measure the average execution time.

TABLE II: The execution time of different DNNs using different types of PL-PS interfaces

DNN	Configuration	execution time (ms)
Object Detect (YOLOv3)	conf 1	8.256
	conf 2	9.304
	conf 3	24.821
YOLOv4	conf 1	62.146
	conf 2	73.175
	conf 3	187.052
Mobilenetv2	conf 1	2.894
	conf 2	3.304
	conf 3	8.919

Experimental results shown in Table II reveal that various types of PL-PS interface exhibit distinct memory access latencies. Therefore, when computing the WCET of DPUs, the access speed of different types of PL-PS interfaces should also be considered.

B. Software Model

CNNs tasks executed on the DPU can be divide into four different phases:

- Read instructions phase: The DPU reads instructions from the DDR memory through the instruction port.
- Read data phase: The DPU reads data words from the DDR memory through the two data ports.
- Write data phase: The DPU writes data words to the DDR memory through the two data ports.
- Elaboration phase: The duration that the DPU only performs calculations and no activity on the bus is performed.

It has been revealed that the DPU employs a parallel-series execution model, where the read data phase occurs in parallel with the read instructions phase and the write data phase [8], as illustrated in Figure 6. During the memory access phases, the DPU also performs computations concurrently. We follow the existing method [8] to measure the time duration when there is no bus activity to bound execution time of computation that do not overlap with memory access, referred to as the Elaboration phase. Given that various DPU architectures adhere to the same execution model but differ in resource utilization, the proposed method is not constrained by the specific DPU architecture in use.

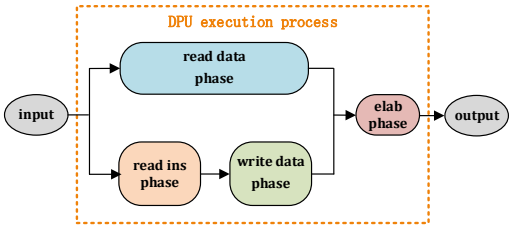


Fig. 6: The DPU execution model.

The Xilinx Vitis AI compiler fuses one or more consecutive layers into one DPU node, the basic DPU processing unit, to reduce the memory communication overhead and improve the parallelism. We use the control flow graph (CFG) composed by DPU nodes to model the structure of CNN tasks as shown in Figure 7. Note, the DPU node and the control flow between them can be obtained at compile time by the Xilinx Vitis AI compiler, and each DPU node follows the parallel-series execution model illustrated in 6. We show a fragment of the CFG of `MobilenetV2` compiled by Vitis AI 2.0 in Figure 7. Xilinx Vitis AI compiler fuses the convolution layer, the BiasAdd layer, and the Relu layer into a single DPU node.

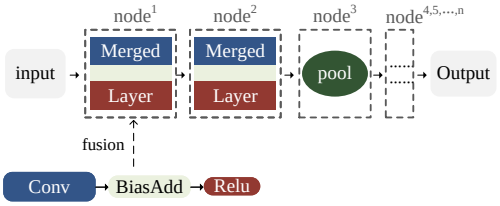


Fig. 7: The fragment of CFG of MobilenetV2 compiled by Vitis AI.

V. Timing Analysis for CNN tasks

This section introduces the proposed timing analysis framework. We begin by introducing the model of the execution time in section V-A. Subsequently, we show the proposed timing analysis method in section V-B and section V-C. To enhance clarity and comprehension, we provide a summary of

the notations and terminologies employed in this work, which can be found in Table III.

TABLE III: Notations and terminologies

Symbol	Meaning
$t^{addr/word}_{read/write}$	the measured maximum time duration for transferring data (i.e., an address request, a write response, or a data word) between DPU ports and PL-PS interface.
$t^{HP/HPC/LPD}_{read/write}$	the measured maximum time duration from when a transaction is issued by HP/HPC/LPD in PS until the latest response is received.
n_{IC}	counts of active master ports on the interconnect IC
N_{INS}	counts of transactions issued by DPU instruction port
$N^{D\_0/D\_1}_{R/W}$	counts of transactions issued by DPU *data0*/*data1* port in read/write data phase
Δ_{INS}	counts of data words issued by DPU instruction port in read instructions phase
$\Delta^{D\_0/D\_1}_{R/W}$	counts of data words issued by DPU *data0*/*data1* port in read/write data phase
$T_{R/W/INS/ELAB}$	the execution time of each DPU phase
$T^{trsf/wait}_{R/W/INS}$	the transfer/waiting time of each DPU phase
$T^{extra}_{R/W/INS}$	the additional execution time incurred due to the contention and interference among multiple DPUs
$T^{base}_{R/W/INS}$	the worst-case execution time when the DPU is executed independently

A. Modeling inference time with multi-DPU engines

Based on the parallel-series DPU execution model shown in Figure 6, the DPU inference time can be bounded by:

$$T = \max\{T_R, T_{INS} + T_W\} + T_{ELAB} \tag{1}$$

T_{ELAB} denotes the time duration that has no off-chip memory accesses and is thus relatively stable in a single DPU environment. We follow the method [8] to measure the T_{ELAB} in a single DPU environment to bound the time required for elaboration in the multi-DPU environment. Given that CNN tasks face increased memory access delays due to inter-DPU contentions while computation time remains relatively stable, the T_{ELAB} determined in the single DPU context serves as a conservative estimate for time allocation in the multi-DPU setting. Therefore, the major problem of computing the worst-case inference time is to bound the memory access time of other DPU phases, i.e., T_R, T_{INS}, and T_W.

We further decompose the execution time of different memory access phases into two components: the base execution time (i.e., $T^{base}_{R/W/INS}$) and the extra execution time (i.e., $T^{extra}_{R/W/INS}$). The base execution time of a transaction denotes the memory access time without contentions with other DPUs, while the waiting time of a transaction denotes the time that a transaction needs to wait on the interconnects and the DDR port arbiter.

$$T_\alpha = T^{base}_\alpha + T^{extra}_\alpha, \alpha \in \{R, W, INS\}$$

By equation 1 the worst-case execution time can be further represented by:

$$T = \max\{T^{base}_R + T^{extra}_R, T^{base}_{INS} + T^{extra}_{INS} + T^{base}_W + T^{extra}_W\} + T_{ELAB}$$

Since,

$$\max\{T^{base}_R + T^{extra}_R, T^{base}_{INS} + T^{extra}_{INS} + T^{base}_W + T^{extra}_W\} \leq \max\{T^{base}_R, T^{base}_{INS} + T^{base}_W\} + \max\{T^{extra}_R, T^{extra}_{INS} + T^{extra}_W\}$$

We define the base execution time (i.e., T^{base}) and the extra execution time (i.e., T^{extra}) as:

$$T^{base} = \max\{T^{base}_R, T^{base}_{INS} + T^{base}_W\} \tag{2}$$

$$T^{extra} = \max\{T^{extra}_R, T^{extra}_{INS} + T^{extra}_W\} \tag{3}$$

Then, the worst case execution time can also be bounded by:

$$T^{base} + T^{extra} + T_{ELAB}$$

where the base execution time analysis (T^{base}) and the extra execution time analysis (T^{extra}) are shown in section V-B and V-C1, respectively.

B. Computing the base execution time of each DPU phase

The base execution time analysis produces the inference time when the DPU is executed independently. The proposed base execution time analysis is a generalized WCET analysis framework that considers various memory port configurations, where transactions issued by different DPU phases can contend with each other, resulting in waiting time. We further decompose the base execution time into two components: the transfer time (i.e., $T^{trsf}_{R/W/INS}$) and the waiting time (i.e., $T^{wait}_{R/W/INS}$). The transfer time denotes the duration required for transactions to complete without encountering any contention. The waiting time refers to the additional waiting time resulting from contentions among read transactions between the instruction port and the data ports of the same DPU. It is important to distinguish the waiting time from the extra execution time, as the extra execution time specifically accounts for the time spent on contentions among different DPUs.

$$T^{base}_\alpha = T^{trsf}_\alpha + T^{wait}_\alpha, \alpha \in \{R, W, INS\}$$

The completion of a transaction involves four sequential steps. Initially, a read transaction initiates an address request to the PL-PS interface (represented by t^{addr}_{read}). Subsequently, this request is transmitted to the memory (i.e., DDR). Following this, multiple data words are read back to the PL-PS interface. Finally, diverse data words are dispatched to the DPU (indicated by t^{word}_{read}). In our study, due to the inability to monitor DDR port activities, we merge the second and third steps, and the time duration is bounded by measuring the period from when the transaction is triggered by the PL-PS interface until all responses are received by the PL-PS interface, denoted as $t^{HP/HPC/LPD}_{read/write}$. Moreover, a transaction may encompass varying counts of memory transfers. Given the unavailability of detailed DPU hardware design information and the relatively constant transfer time, we measure the maximum transfer time (i.e., t^{addr}_{read}, $t^{HP/HPC/LPD}_{read}$, and t^{word}_{read}) to set an upper limit on the transfer time of the read instruction phase. In scenarios where the specifics of the DDR bus architecture and DPU design are publicly accessible, a more granular transfer time analysis could be integrated to enhance the precision of the analysis. Therefore, the transfer time of the read instructions phase is bounded by:

$$T^{trsf}_{INS} = N_{INS} \cdot t^{addr}_{read} + N_{INS} \cdot t^{HP/HPC/LPD}_{read} + \Delta_{INS} \cdot t^{word}_{read}$$

Different from the instruction phase, the DPU fetch data via two data ports. Since the interleaving of transactions between different data ports are not detailed, we pessimistically consider that the transactions issued from different data ports are not overlapped to safely compute the upper bound on the memory access time of the read data phase.

$$\begin{aligned} \mathrm{T_R^{trsf}} = \ & (\mathrm{N_R^{D\_0}} + \mathrm{N_R^{D\_1}}) \cdot \mathrm{t_{read}^{HP/HPC/LPD}} + \\ & (\mathrm{N_R^{D\_0}} + \mathrm{N_R^{D\_1}}) \cdot \mathrm{t_{read}^{addr}} + (\Delta_\mathrm{R}^{\mathrm{D\_0}} + \Delta_\mathrm{R}^{\mathrm{D\_1}}) \cdot \mathrm{t_{read}^{word}} \end{aligned} \tag{4}$$

Compared to the read data phase, the write data phase involves not only transferring data to the memory but also waiting for a corresponding write response. We also measure the maximum duration for transferring the write response to bound the write response duration, denoted by write $\mathrm{t_{write}^{resp}}$. So, the transfer time of the write data phase is bounded by:

$$\begin{aligned} \mathrm{T_W^{trsf}} = \ & (\mathrm{N_W^{D\_0}} + \mathrm{N_W^{D\_1}}) \cdot \mathrm{t_{write}^{HP/HPC/LPD}} + \\ & (\mathrm{N_W^{D\_0}} + \mathrm{N_W^{D\_1}}) \cdot (\mathrm{t_{write}^{addr}} + \mathrm{t_{write}^{resp}}) + (\Delta_\mathrm{W}^{\mathrm{D\_0}} + \Delta_\mathrm{W}^{\mathrm{D\_1}}) \cdot \mathrm{t_{write}^{word}} \end{aligned} \tag{5}$$

Since the AXI has independent channels for read data, read address, write data, write address, and write response, when the DPU operates independently, contention may arise between the read data phase and the read instruction phase on the read data and read address channels, leading to wait times. When two transactions contend, the maximum wait time equals the transfer time of the currently processed transaction. The transfer time of a transaction is dictated by the specific PL-PS interface it connects to and can be denoted as $\mathrm{t_{read}^{trsf}}(\mathrm{D\_0/D\_1/INS})$. Therefore, following the round-robin arbitration policy, we have the following equation:

$$\begin{aligned} \mathrm{T_{INS}^{wait}} =& \min\{2 \cdot \mathrm{N_{INS}}, \mathrm{N_R^{D\_0}} + \mathrm{N_R^{D\_1}}\} \\ & \cdot \max\{\mathrm{t_{read}^{trsf}}(\mathrm{D\_0}), \mathrm{t_{read}^{trsf}}(\mathrm{D\_1})\} \\ \mathrm{T_R^{wait}} =& \min\{\mathrm{N_{INS}}, \mathrm{N_R^{D\_0}} + \mathrm{N_R^{D\_1}}\} \cdot \mathrm{t_{read}^{trsf}}(\mathrm{INS}) \end{aligned}$$

By the round-robin policy, a transaction from the read instruction phase can wait for a maximum of two read data transactions, considering that a DPU features two data ports (i.e., $2 \cdot \mathrm{N_{INS}}$). Additionally, the total number of transactions that the read instruction phase needs to wait for never surpasses the total count of transactions from the read data phase (i.e., $\mathrm{N_R^{D\_0}} + \mathrm{N_R^{D\_1}}$). Therefore, $\mathrm{T_{INS}^{wait}}$ takes the minimum value between them. Therefore, the waiting time does not exceed the total transfer time of transactions that need to wait. The computation for $\mathrm{T_R^{wait}}$ is similar.

C. Extra execution time analysis

Parallel executing CNN tasks on different DPUs may issue transactions simultaneously, and thus may contend with each other, leading to non-deterministic memory access latency. We in this section first show a basic extra execution time analysis in section V-C1, and then a fine-grained extra execution time analysis is shown in V-C2.

1) Extra execution time analysis between CNN tasks: The extra execution time is generally spent on the interconnect arbitration and the DDR port arbitration. During the memory access, three interconnects may be visited, PL interconnect, PS interconnect, and CCI as shown in Figure 5. All the interconnects adopt the round-robin arbitration mechanism [14]. Since CCI contains two master ports and two slave ports, under the round-robin arbitration mechanism, a transaction does not need to wait other transactions when visiting CCI. In this section, we compute the extra execution time on the PL interconnect (denoted by $\mathrm{IC_1}$), the PS interconnect (denoted by $\mathrm{IC_2}$), and the DDR port arbiter (denoted by PA).

$\mathrm{IC_1}$ and $\mathrm{IC_2}$ may contain several master ports, and only one slave port. Master ports and slave ports have a read and a write channel which can be access in parallel. In this paper, we consider a general interconnect model, n:1 interconnect model, i.e., the interconnect contains n master ports, denoted by $\mathrm{P_1}, \mathrm{P_2}, \ldots, \mathrm{P_n}$, and one slave port. We denote the counts of transactions issued on P_i as N_i. For N_i transactions issued by DPU $\mathrm{DPU_a}$, we can use the following lemma to calculate the maximum number of transactions of other DPUs that they need to wait for.

Lemma 1. *For all the read/write transactions issued on P_j by DPU_a, the maximum number of transactions from other DPUs they should wait for can be bounded by:*

$$\begin{aligned} N\_R^i_{wait}(DPU_a) &= \sum_{i=1}^{n_{IC}} min\left\{N_j^{read}(DPU_{others}), N_i^{read}(DPU_a)\right\} \\ N\_W^i_{wait}(DPU_a) &= \sum_{i=1}^{n_{IC}} min\left\{N_j^{write}(DPU_{others}), N_i^{write}(DPU_a)\right\} \end{aligned} \tag{6}$$

, where $N_j^{write}(DPU_a)$/$N_j^{read}(DPU_a)$ returns the count of transactions that issued on P_i's write/read channel from DPU_a, which can be computed through a backward data-flow analysis during the computation of each DPU phase. $N_j^{write}(DPU_{others})$/$N_j^{read}(DPU_{others})$ returns the count of transactions that issued by other DPUs, which can be computed by $N_j^{read} - N_j^{read}(DPU_a)$.

Proof. On the interconnect, transactions on a master port (i.e., P_i) should wait for transactions on its sibling master ports. Since the interconnect adopts a round robin arbitration method, for each of its sibling master ports (i.e., P_j), the maximum number of transactions P_i should wait does not exceed the counts of transactions issued on P_j from other DPUs (i.e., $\mathrm{N_j^{read}(DPU_{others})}$) and the counts of transactions issued on its own port P_i (i.e., $\mathrm{N_i^{read}(DPU_a)}$). So the total number of transactions from other DPUs that need to wait do not exceed the sum of the wait transactions of all the sibling ports, as shown in equation 6.

□

We follow lemma 1 to compute the extra execution time of each DPU phase on $\mathrm{IC_1}$, $\mathrm{IC_2}$, and PA, which are denoted by $\mathrm{T^{IC_1}}$, $\mathrm{T^{IC_2}}$, $\mathrm{T^{PA}}$ and detailed in the following. So the extra execution time of each DPU phase can be computed by:

$$\begin{aligned} \mathrm{T_{INS}^{extra}} =& \mathrm{T_{INS}^{IC_1}} + \mathrm{T_{INS}^{IC_2}} + \mathrm{T_{INS}^{PA}} \\ \mathrm{T_R^{extra}} =& \mathrm{T_R^{IC_1}} + \mathrm{T_R^{IC_2}} + \mathrm{T_R^{PA}} \\ \mathrm{T_W^{extra}} =& \mathrm{T_W^{IC_1}} + \mathrm{T_W^{IC_2}} + \mathrm{T_W^{PA}} \end{aligned} \tag{7}$$

Read instructions phase: According to the hardware structure shown in section IV-A, each DPU port can only be

connected with one master port of an interconnect in IC_1 and IC_2. We denote the port numbers connected with the instruction port of the DPU in IC_1 and IC_2 as IC_1^{ins} and IC_2^{ins}, respectively. By lemma 1, the transactions from other DPUs that may be contend with can be computed as N_read_{wait}, and therefore the extra execution time of the read instruction phase on IC_1 is bounded by:

$$T_{INS}^{IC_1} = N\_read_{wait}^{IC_1^{ins}} \cdot t_{read}^{PS}(ins) \tag{8}$$

Similarly, on the IC_2, the extra execution time of the read instruction phase is bounded by:

$$T_{INS}^{IC_2} = N\_read_{wait}^{IC_2^{ins}} \cdot t_{read}^{PS}(ins) \tag{9}$$

The computation method for extra execution time on PA is similar to the which on the interconnect. On the DDR port arbiter, among all the 6 DDR ports, at most 5 can be used by DPUs [20], and at most one can be serviced at any given time. Due to the lack of publicly disclosed detailed information about the DDR, it is challenging to accurately measure the time required to complete a transaction service. Therefore, in a conservative manner, we adopt the duration from the time when a transaction is issued from the HP interface to the time when the the corresponding first word from the DDR memory is read back to the HP interface after removing all the interference, denoted by t_{read}^{HP}/ t_{write}^{HP}, to bound the DDR port service time. It is worth noting that if the internal details of the DDR become publicly available, our results can be further refined and improved.

We denote the XPI port number connected with the instruction port of the DPU in PA layer as XPI^{ins}, the extra execution time of the read instruction phase on PA is bounded by:

$$T_{INS}^{PA} = N\_read_{wait}^{XPI^{ins}} \cdot t_{read}^{HP} \tag{10}$$

Read/write data phase: The method for computing the waiting time of the read data phase and the write data phase is different from the read instructions phase, as each DPU accesses data via two separated data ports. We denote the port number of interconnects connected with the two data ports of the DPU as IC^{D_0} and IC^{D_1}. In this paper, we take the worst-case by analyzing the waiting time of each DPU data port individually and summing them together as the total waiting time. Similar to equation 8, the waiting time of the read/write data phase on IC_1 and IC_2 are bounded by:

$$\begin{aligned} T_R^{IC} &= N\_read_{wait}^{IC^{D\_0}} \cdot t_{read}^{PS(D\_0)} + N\_read_{wait}^{IC^{D\_1}} \cdot t_{read}^{PS(D\_1)} \\ T_W^{IC} &= N\_write_{wait}^{IC^{D\_0}} \cdot t_{write}^{PS(D\_0)} + N\_write_{wait}^{IC^{D\_1}} \cdot t_{write}^{PS(D\_1)} \end{aligned} \tag{11}$$

We denote the sequence number of the XPI port connected with the two data ports of the DPU in PA layer as XPI^{D_0} and XPI^{D_1}. To avoid double counting the waiting time, we divide the analysis into two cases:

- If the two DPU data ports are connected to the same XPI port, i.e., $XPI^{D\_0} = XPI^{D\_1}$, the computation is similarly to that of the waiting time computation for instruction phase.
- If the two DPU data ports are connected to different XPI ports, i.e., $XPI^{D\_0} \neq XPI^{D\_1}$, the counts of read transactions they should wait for from other ports (i.e., XPI_i) can not exceed the counts of read transaction issued on these ports (i.e.,$min\{N_R^{XPI_i}, N_R^{XPI^{D\_0}} + N_R^{XPI^{D\_1}}\}$). So, the total number of read transaction that these two ports need to wait from other ports is bounded by $\sum_{i=1}^{5} min\{N_R^{XPI_i}, N_R^{XPI^{D\_0}} + N_R^{XPI^{D\_1}}\} - N_R^{XPI^{D\_0}} - N_R^{XPI^{D\_1}}$. In addition, the waiting time between these two ports should also be counted (i.e., $2 \cdot min(N_R^{XPI^{D\_0}}, N_R^{XPI^{D\_1}})$).

Thus, the waiting time of read/write data phase in PA is: if $XPI^{D\_0} = XPI^{D\_1}$:

$$\begin{aligned} T_R^{PA} &= N\_read_{wait}^{XPI^{D\_0}} \cdot t_{read}^{HP} \\ T_W^{PA} &= N\_write_{wait}^{XPI^{D\_0}} \cdot t_{write}^{HP} \end{aligned} \tag{12}$$

if $XPI^{D\_0} \neq XPI^{D\_1}$:

$$\begin{aligned} T_R^{PA} =& \Big\{ \sum_{i=1}^{5} min\left\{N_R^{XPI_i}, N_R^{XPI^{D\_0}} + N_R^{XPI^{D\_1}}\right\} - N_R^{XPI^{D\_0}} \\ & - N_R^{XPI^{D\_1}} + 2 \cdot min(N_R^{XPI^{D\_0}}, N_R^{XPI^{D\_1}}) \Big\} \cdot t_{read}^{HP} \\ T_W^{PA} =& \Big\{ \sum_{i=1}^{5} min\left\{N_W^{XPI_i}, N_W^{XPI^{D\_0}} + N_W^{XPI^{D\_1}}\right\} - N_W^{XPI^{D\_0}} \\ & - N_W^{XPI^{D\_1}} + 2 \cdot min(N_W^{XPI^{D\_0}}, N_W^{XPI^{D\_1}}) \Big\} \cdot t_{write}^{HP} \end{aligned} \tag{13}$$

2) A fine-grained extra execution time analysis between CNNs: The above section shows a basic extra execution time analysis, and we denote the analyzed extra execution time as T_{basic}^{extra}. The basic execution time analysis performs the contention analysis at the granularity of CNN tasks, i.e., considers that a transaction may wait all the transactions from the parallel executing CNN tasks. However, DPU nodes of CNN tasks compiled by Vitis AI are generally executed sequentially [22], and parallel executing CNN tasks exhibits limited node-level interleaving behavior. Therefore, the contention between different DPU nodes also follows the node-level interleaving constraint.

CON. 1. *Let us assume DPU nodes n_A^i, n_A^j, n_B^i, and n_B^j from two parallel execution CNN tasks A and B, We suppose that n_A^i is executed before n_A^j, and n_B^i is executed before n_B^j. If transactions from n_A^i contend with transactions from n_B^j, transactions from n_A^j never contend with transactions from n_B^i. This is because, when n_B^j is visited, n_B^i is passed and never be visited until B is finished.*

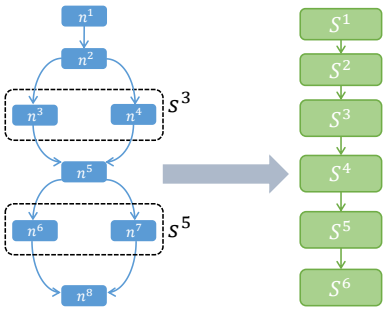


Fig. 8: Sequentialized CFG of CNN tasks.

Based on the above analysis, we make a further step to perform a fine-grained extra execution time analysis by considering the node-level interleaving constraint. Firstly, we combine parallel DPU nodes into a segment to get a sequentialized CFG that composed by segments as shown in Figure 8. In the example, parallel DPU nodes n_3 and n_4 are combined into a segment S_3, n_6 and n_7 are combined into a segment S_5, while other DPU nodes are remained as a segment. For each pair of segments between parallel executing CNN tasks, we follow the method presented in section V-C1 to compute the extra execution time resulted from contentions between them. Then, we adopt the contention matrix [23] to compute the worst-case extra execution time between two CNN tasks. In specific, for two parallel executing CNNs, a contention matrix (dentoed by Mat$[m][n]$) is built to model the extra execution time between each pair of segment as shown in Figure 9. The worst-case extra execution time can be computed by finding the longest path that can only go to the right or the downside from the Mat$[0][0]$ to Mat$[m-1][n-1]$. The proof is presented in [23] and is thus is omitted.

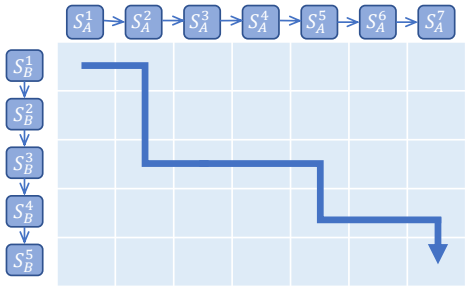


Fig. 9: The transformed sequentialized DPU execution graph.

In case of a CNN task has more than one parallel executing task, we compute the extra execution time with each of them individually and sum them up as the total extra execution time of the task under analysis. We denote the extra execution time computed by the fine-grained analysis method as $T_{\text{fine}}^{\text{extra}}$. So the extra execution time can be bound by:

$$T^{\text{extra}} = \min\{T_{\text{fine}}^{\text{extra}}, T_{\text{basic}}^{\text{extra}}\}$$

Finally, the WCET of the CNN task is bounded by

$$T^{\text{base}} + T^{\text{extra}} + T_{\text{ELAB}}$$

VI. Evaluation

In this section, we present the estimated WCET bounds for various DPU applications with diverse DPU configurations to showcase the applicability of the proposed method. Additionally, comparisons with the state-of-the-art method [8] and the measured results are provided to demonstrate the accuracy and soundness of the proposed methodology, respectively.

A. Experimental setup

The experiment is conducted with Xilinx ZCU102 evaluation board. We deploy DPUs on the FPGA SoC via Vitis 2021.2 and Vitis AI 2.0. The input images used in the experiments were randomly selected from the Cityscapes dataset [18] and ImageNet dataset [16]. In the experiment, the clock domain of the DPU is set as 300MHz, the default value in Vitis.

Follow the profiling method presented in section V, we firstly deploy only one DPU to profile the transfer time and counts of transactions of different CNN tasks in different phases. In order to precisely profile the transfer time, we make the following memory port configuration to avoid contentions on interconnects:

$$\texttt{ins} \Longrightarrow \texttt{S\_AXI\_HP0}$$
$$\texttt{data0} \Longrightarrow \texttt{S\_AXI\_HP1}$$
$$\texttt{data1} \Longrightarrow \texttt{S\_AXI\_HP3}$$

The CNN models are selected from the Xilinx Vitis AI Model Zoo as shown in Table IV. We run different CNNs 10000 times with different input images, and adopt the AXI Performance Monitors (APM) [24] to measure the counts of transactions of different DPU phases. To demonstrate the applicability of the proposed method, we adopt two DPU architectures, i.e., B4096 and B3136. Experimental results show that the counts of transactions of different DPU phases via a specific DPU port are constant, and the counts are shown in Table IV. For a given CNN task, the accessed data and instruction words exhibit similarity across different DPU architectures, but diverge in the duration of the elaboration phase. Typically, CNN applications experience longer elaboration phases on smaller DPUs due to the diminished computational resources, leading to extended computation times.

The proposed method decomposes a CNN task into sequentialized segments to derive a tight estimated WCET bound. We in this paper firstly get the control flow of different CNN tasks and follow the method presented in section V-C2 to partition a CNN network into different segments and measure the bus activity. For example, OD_SSD is partitioned into four segments, and we run each segment independently to measure the bus activity as shown in Table V.

We utilized the Xilinx System ILA [28] to measure the transfer times of data and address requests via the read and write channels correspondingly. Additionally, we profiled the transfer time of write responses. The experimental results, as illustrated in Table VI, demonstrate the consistency in transfer times across various requests.

We run each CNN task independently 1 million times and use the AXI Performance Monitors (APM) to profile the transfer time of transactions via different PL-PS interfaces. For all the measured time, we adopt the maximum one as the transfer time of each PL-PS interface. For transactions on HP interface, the transfer time for read is bounded by 35 clock cycles, while the transfer time for write is bounded by 25 clock cycles. For transactions on HPC interface, the transfer time for read is bounded by 38 clock cycles, while the transfer time for write is bounded by 28 clock cycles. For LPD interface, we separately analyze the transfer time of read data transactions and instruction transactions as there is a huge gap between them. Experimental results show that, the transfer time of read data transactions is bounded by 146 clock cycles, the transfer time of read instruction transactions is bounded by 40 clock cycles. The transfer time for write transactions is bounded by

TABLE IV: Measured bus activity of CNN tasks with different DPU architectures

CNN	B4096						B3136					
	port	N_R	Δ_R	N_W	Δ_W	ELAB(ms)	port	N_R	Δ_R	N_W	Δ_W	ELAB(ms)
Yolov4 [21]	**ins**	67869	271474			0.55	**ins**	64110	256441			1.11
	data0	538948	4783468	194863	2071760		**data0**	289449	2924614	132577	1289706	
	data1	240981	2277140	195578	2067473		**data1**	280804	2859145	129431	1266401	
MobileNetv2 [15]	**ins**	16867	66465			0.20	**ins**	15073	60295			1.02
	data0	33608	378167	10698	167589		**data0**	17676	201364	2667	41125	
	data1	17955	204656	8775	136974		**data1**	17420	198497	2665	41115	
SqueezeNet [25]	**ins**	9992	39969			0.10	**ins**	5931	23727			0.42
	data0	37903	403834	4745	31923		**data0**	8037	89512	5668	37139	
	data1	16677	184348	4310	27664		**data1**	7876	87763	5141	32986	
Lane Detect (VpgNet) [26]	**ins**	14058	56234			0.23	**ins**	12552	50211			0.87
	data0	32987	341890	36624	196182		**data0**	23071	234938	32518	174841	
	data1	18736	185130	33349	19600		**data1**	22939	233901	29715	174175	
Object Detect (YOLOv3) [17]	**ins**	14295	57178			0.59	**ins**	109206	436827			1.24
	data0	55141	689476	26984	280745		**data0**	389287	3924561	77050	1083905	
	data1	28456	365534	27094	282157		**data1**	390396	3929578	76240	1071641	
Pedestrian Detect (SSD) [27]	**ins**	12060	48239			0.70	**ins**	12156	48626			1.61
	data0	40090	428144	17696	268108		**data0**	29188	305487	16860	256068	
	data1	21058	216557	16693	253262		**data1**	31625	330589	17181	261103	
Object Detect (SSD) [27]	**ins**	8905	35620			0.34	**ins**	11646	46587			2.31
	data0	48007	503056	19852	301354		**data0**	33275	360383	20482	307502	
	data1	23170	244071	17915	272489		**data1**	33416	362339	19254	291694	

TABLE V: Measured bus activity of partitioned segments of OD_SSD

OD_SSD	port	N_R	Δ_R	N_W	Δ_W
segment1	**ins**	824	3295		
	data0	2016	21269	5042	78434
	data1	1340	15365	5042	78434
segment2	**ins**	2022	8089		
	data0	16141	176591	6396	11573
	data1	7494	79496	6376	101260
segment3	**ins**	1635	6540		
	data0	14075	141210	3570	53820
	data1	6825	71075	3577	53858
segment4	**ins**	4424	17696		
	data0	15775	163986	4843	67527
	data1	7511	78136	2920	38936

TABLE VI: Transfer time of data requests and address requests

request	t^{addr}_{read}	t^{word}_{read}	t^{addr}_{write}	t^{word}_{write}	t^{resp}_{write}
cycles	1	1	1	2	1

74 clock cycles. To sum up, the transfer time of transactions in PS is shown in Table VII.

TABLE VII: Maximum transfer time of a transaction

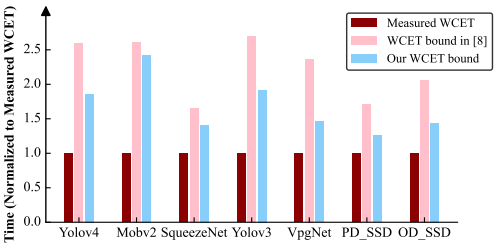
request	t^{HP}_{read}	t^{HP}_{write}	t^{HPC}_{read}	t^{HPC}_{write}	$t^{LPD\_D}_{read}$	$t^{LPD\_INS}_{read}$	t^{LPD}_{write}
cycles	35	25	38	28	146	40	74

B. Evaluation under the single DPU environment

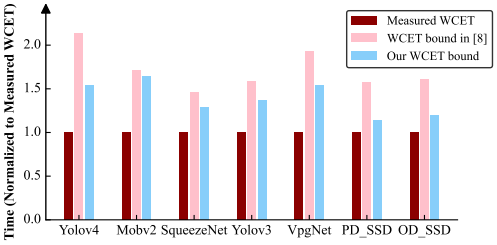
We first evaluate the proposed method by comparing it with the state-of-the-art method [8]. Since the WCET analysis method presented in [8] is independent with the memory port configuration and did not reveal their memory port configuration, we follow the default memory configuration recommended by Vitis AI:

$$\texttt{ins} \Longrightarrow \texttt{S\_AXI\_HP0}$$
$$\texttt{data0} \Longrightarrow \texttt{S\_AXI\_HP1}$$
$$\texttt{data1} \Longrightarrow \texttt{S\_AXI\_HP3}$$

Using the profiled transfer time (i.e., shown in Table VII) and bus activity data (i.e., shown in Table IV) from various CNN tasks, we follow the WCET computation method stated in [8] to calculate the estimated WCET bound of the state-of-the-art method. The resulting WCET values for both the proposed method and the state-of-the-art method are depicted in Figure 10[1]. Similarly, we execute each CNN task one million times and treat the maximum execution time among these trials as the measured WCET as shown in Figure 10.



(a) The comparison of WCET bounds with B4096.



(b) The comparison of WCET bounds with B3136.

Fig. 10: Comparison between the estimated WCET of our method, the state-of-the-art method, and measured WCET. All the results are normalized to the measured WCET.

Experimental results show that, in the single DPU environment, the WCET estimated by the proposed method is smaller than that of the state-of-the-art method [8]. To sum up, the WCET bound obtained by our approach is tightened by more than 27% on average. For the best case, the estimated WCET bound of our method is only 128% of the measured WCET bound, which is very close to the measured WCET bound. Note that, even we have executed each CNN task for one million times, we may still fail to obtain the actual worst-case

[1]The detailed procedure for calculating the WCET of each CNN task is openly available at https://github.com/212SoleYu/WCET-Multi-DPUs.

TABLE VIII: Comparison between our estimated WCET and measured WCET in the 2-DPU environment.

CNN Conf	B4096						B3136					
	CNN 1			CNN 2			CNN 1			CNN 2		
	Measured	Our	Ours/Measured	Measured	Our	Our/Measured	Measured	Our	Our/Measured	Measured	Our	Our/Measured
Mobv2 & PD_SSD	4.496	14.138	3.144	10.803	15.744	1.457	7.041	15.982	2.269	11.290	17.958	1.590
OD_SSD & Yolov3	9.970	30.416	3.050	71.391	133.080	1.864	11.573	32.049	2.769	98.749	141.428	1.432
Squeezenet & PD_SSD	3.000	8.037	2.678	10.616	11.735	1.105	3.708	8.542	2.303	11.111	14.266	1.284
Mobv2 & Yolov3	5.077	16.650	3.279	71.038	126.264	1.777	5.567	17.615	3.164	98.528	181.024	1.837

TABLE IX: Comparison between our estimated WCET and measured WCET in the 3-DPU environment.

CNN Configurations		CNN 1			CNN 2			CNN 3		
		Measured	Our	Our/Measured	Measured	Our	Our/Measured	Measured	Our	Our/Measured
B4096	OD_SSD & PD_SSD & Yolov3	11.62	32.17	2.77	13.35	29.73	2.23	83.09	166.56	2.00
	VpgNet & Mobv2 & Squeezenet	8.56	23.62	2.76	5.76	18.05	3.13	4.58	9.91	2.16
	Mobv2 & Squeeznet & OD_SSD	5.30	13.68	2.58	3.58	7.87	2.20	10.65	27.49	2.58
	Yolov4 & Yolov4 & Mobv2	57.01	180.67	3.17	56.51	181.34	3.21	11.50	22.95	1.99
B3136	OD_SSD & PD_SSD & Yolov3	13.31	34.92	2.62	12.91	32.20	2.31	108.26	177.41	1.64
	VpgNet & Mobv2 & Squeezenet	8.69	24.85	2.86	6.02	19.51	3.24	4.35	11.64	2.67
	Mobv2 & Squeeznet & OD_SSD	5.71	15.28	2.67	4.16	9.61	2.31	12.40	30.07	2.42
	Yolov4 & Yolov4 & Mobv2	66.18	187.01	2.82	66.80	189.03	2.83	11.10	29.08	2.62

execution time. By utilizing a tighter estimated WCET bound, computing resources can be saved at the design stage and a more accurate schedulability test can be produced.

C. Evaluation under the multiple DPU environment

As this is the first study to conduct WCET analysis in the multi-DPU environment, we evaluate our method in the multi-DPU environment by comparing the estimated WCET bound of our method with the measured WCET. The evaluation is firstly conducted in a dual-DPU environment. Two DPUs are deployed on the FPGA SoC with the default memory configuration of Vitis AI:

```
DPU1 : ins ⟹ S_AXI_HPC0
    data0 ⟹ S_AXI_HP0  data1 ⟹ S_AXI_HP1
DPU2 : ins ⟹ S_AXI_HPC1
    data0 ⟹ S_AXI_HP2  data1 ⟹ S_AXI_HP3
```

Also, we execute these two CNNs in parallel for 1 million times and measure the worst-case execution time. Experimental results are shown in Table VIII. When executing two DPUs in parallel, the presence of contentions between them has a substantial impact on the execution time of CNN tasks. As a result, the execution time of CNNs exhibits low predictability. So, in the dual-DPU environment, the estimated WCET bound of our method tends to be more pessimistic compared to the single-DPU environment. Nevertheless, in the best-case, the estimated WCET bound is 1.73x of the measured WCET bound which is relatively tight.

Furthermore, we deploy 3 DPUs on the FPGA SoC, we follow the official recommended memory port configuration:

```
DPU1 : ins ⟹ S_AXI_LPD
    data0 ⟹ S_AXI_HP1  data1 ⟹ S_AXI_HP2
DPU2 : ins ⟹ S_AXI_LPD
    data0 ⟹ S_AXI_HP3  data1 ⟹ S_AXI_HP3
DPU3 : ins ⟹ S_AXI_LPD
    data0 ⟹ S_AXI_HPC0  data1 ⟹ S_AXI_HPC0
```

Experimental results are shown in Table IX. In the best-case scenario, the estimated WCET bound is 164% of the measured WCET bound, indicating a relatively tight estimation. Conversely, in the worst-case scenario, the estimated WCET bound rises to 317% of the measured WCET bound, with an average estimation of 252% of the measured WCET bound.

D. Discussion of Pessimism in the Estimated Results

It is critical to emphasize that in hard real-time systems, violations of timing constraints may lead to catastrophic consequences. Given the lack of detailed DPU hardware design specifications, our methodology makes conservative assumptions to ensure a safe upper-bound estimation of inference time. First, to bound transaction transfer times, we adopt the longest observed duration across variable-length data transfers, as transactions may involve different counts of data transfers. Second, despite the DPU's dual data ports, we pessimistically model transactions from these ports as non-overlapping to account for worst-case scheduling behavior. Third, in contention scenarios between DPUs, we assume all transactions conflict with concurrent CNN task executions on the shared interconnect. While these choices result in a pessimistic worst-case execution time (WCET) bound compared to empirical measurements especially in the multi-DPU environment, the derived bound remains provably safe. Furthermore, our method can be extended to tighten the estimate once comprehensive hardware design information regarding the DPU becomes publicly available.

To evaluate the impact of our conservative modeling choices, we systematically relax two key assumptions in our methodology. While our method assumes non-overlapping transactions from a DPU's dual data ports (due to undisclosed hardware architecture), we introduce the overlapping ratio R_{ol}, representing the proportion of temporally overlapping inter-port transactions. This refines the transfer time estimation as:

$$\mathrm{T}_{\mathrm{R}}^{\mathrm{trsf}} = (\Delta_{\mathrm{R}}^{\mathrm{D\_0}} + \Delta_{\mathrm{R}}^{\mathrm{D\_1}} - \mathrm{R_{ol}} * \min(\Delta_{\mathrm{R}}^{\mathrm{D\_1}}, \Delta_{\mathrm{R}}^{\mathrm{D\_0}}) \cdot \mathrm{t}_{\mathrm{read}}^{\mathrm{word}} + (\mathrm{N}_{\mathrm{R}}^{\mathrm{D\_0}} + \mathrm{N}_{\mathrm{R}}^{\mathrm{D\_1}} - \mathrm{R_{ol}} * \min(\mathrm{N}_{\mathrm{R}}^{\mathrm{D\_0}}, \mathrm{N}_{\mathrm{R}}^{\mathrm{D\_1}})) \cdot (\mathrm{t}_{\mathrm{read}}^{\mathrm{HP/HPC/LPD}} + \mathrm{t}_{\mathrm{read}}^{\mathrm{addr}})$$

$$\mathrm{T}_{\mathrm{W}}^{\mathrm{trsf}} = (\Delta_{\mathrm{W}}^{\mathrm{D\_0}} + \Delta_{\mathrm{W}}^{\mathrm{D\_1}} - \mathrm{R_{ol}} * \min(\Delta_{\mathrm{W}}^{\mathrm{D\_1}}, \Delta_{\mathrm{W}}^{\mathrm{D\_0}}) \cdot \mathrm{t}_{\mathrm{read}}^{\mathrm{word}} + (\mathrm{N}_{\mathrm{W}}^{\mathrm{D\_0}} + \mathrm{N}_{\mathrm{W}}^{\mathrm{D\_1}} - \mathrm{R_{ol}} * \min(\mathrm{N}_{\mathrm{W}}^{\mathrm{D\_0}}, \mathrm{N}_{\mathrm{W}}^{\mathrm{D\_1}})) \cdot (\mathrm{t}_{\mathrm{write}}^{\mathrm{HP/HPC/LPD}} + \mathrm{t}_{\mathrm{write}}^{\mathrm{addr}})$$

Experiments conducted in a single-DPU environment (to isolate intra-DPU effects) compare estimated and measured WCETs across varying R_{ol}. Results shown in Table X demonstrate that while analyzed bounds remain safe for some CNN workloads, excessive relaxation (e.g., when R_{ol} is larger than 0.75 for Yolov3) risks unsafe WCET underestimation.

To address pessimism in multi-DPU contention modeling, we introduce the contention ratio R_{con}, representing the proportion of transactions competing for shared interconnect resources. This parameter refines the extra execution time computations in Equations 8, 9, 10, 11, 12, and 13 by scaling the count of conflict transactions with R_{con}. Table XI compares analyzed and measured extra execution time in a 2-DPU setup. Similarly, relaxing the R_{con} may underestimate the extra execution time (e.g., when R_{con} equals 0.3 for MobileNetv2).

Above results demonstrate that while the analyzed bounds remain safe for certain CNN tasks, relaxing the conservative constraints risks underestimating the WCET. Therefore, in hard real-time systems where meeting deadlines is non-negotiable, it is crucial to adhere to conservative assumptions to prevent potential deadline violations. Conversely, in soft real-time systems where there is more flexibility, a slight relaxation of constraints can lead to a refined WCET estimate.

TABLE X: Comparison between the estimated WCET and the measured WCET with different R_{ol}, where $R_{ol} = 1$ indicates full overlap of transactions from two data ports.

CNN	R_{ol} (Our/Measured)				
	0	0.25	0.5	0.75	1
Yolov4	1.54	1.37	1.20	1.02	***0.90***
MobileNetv2	1.64	1.60	1.59	1.58	1.56
Squeezenet	1.29	1.25	1.22	1.18	1.15
VpgNet	1.54	1.46	1.39	1.32	1.24
Yolov3	1.37	1.21	1.06	***0.91***	***0.76***
PD_SSD	1.17	1.14	1.10	1.07	1.04
OD_SSD	1.33	1.29	1.25	1.21	1.17

TABLE XI: Comparison between the analyzed and measured extra WCET with different R_{con}, where $R_{con} = 1$ indicates that all transactions are contend with other DPU cores.

CNNs	R_{con} (Our/Measured)				
	1	0.8	0.6	0.5	0.3
MobileNetv2 (CNN1)	2.59	2.08	1.56	1.30	***0.78***
PD_SSD (CNN2)	6.26	5.01	3.76	3.13	1.88

E. Some hints for reducing the WCET bound

Based on the proposed method, different memory configurations may lead to different contentions on interconnects and DDR port arbiter, and further leads to different estimated WCET bound. In this section, we give some hints to configure the DPUs, aiming to a minimum estimated WCET bound. In the experiment, we deploy two DPUs, and use four different memory configurations as shown in Table XII. Among all the configurations, configuration 1 is the default from the Vitis AI. However, the estimated WCET bound of CNN1 achieves the minimum WCET bound under configuration 2. This is because that, under configuration 2 the two data ports of `DPU1` are connected to `HP0` and `HP3`, which means that the data transactions issued by `DPU1` will not contend with any other transactions in the IC_1 and IC_2 layers, as illustrated in Figure 4. Moreover, the two data ports of `DPU2` are connected to the `HP1` and `HP2` interfaces, and therefore all the read data transactions and the write data transactions are issued to the same XPI port. Therefore, the data transactions of these two CNN tasks exhibit relatively small contention in the PA layer, and the estimated WCET bound of CNN1 under configuration 2 is tighter. Similarly, CNN2 achieves a tighter WCET bound under configuration 3.

TABLE XII: Different memory port configurations for predictiable WCET bound under the dual-DPU environment.

conf		ins	data0	data1
1	**DPU1**	S_AXI_LPD	S_AXI_HP0	S_AXI_HP1
	DPU2	S_AXI_LPD	S_AXI_HP2	S_AXI_HP3
2	**DPU1**	S_AXI_LPD	S_AXI_HP0	S_AXI_HP3
	DPU2	S_AXI_LPD	S_AXI_HP1	S_AXI_HP2
3	**DPU1**	S_AXI_LPD	S_AXI_HPC1	S_AXI_HPC1
	DPU2	S_AXI_LPD	S_AXI_HPC0	S_AXI_HPC0
4	**DPU1**	S_AXI_HP2	S_AXI_HP1	S_AXI_HP2
	DPU2	S_AXI_HP3	S_AXI_HP1	S_AXI_HP3

VII. Related work

FPGA-based accelerators for DNNs have been studied by researchers, some of them focus on achieving a balance between hardware resources and processing speed [10]–[12], while others focus on maximizing performance [29]–[31]. Xilinx DPU, as one of the most mature FPGA-based DNN accelerator in industry today, has naturally become a popular research subject for many researchers. [32] built an energy prediction model which predicts the energy for any DNN running on a DPU. Zhu [33] proposed a high performance task assignment framework for DPU-based DNN acceleration platform. Wu [34] proposed an end-to-end solution to autonomous driving based on Xilinx DPU.

Timing predictability is an essential issue that must be considered in real-time systems. Because the WCET analysis for multi-core processors appears to be more necessary, more and more studies focus on analyzing the execution predictability of multi-core systems [23], [35]–[37]. The real-time performance of FPGA-based systems has also been studied extensively. Previous study [38] presented timing anomalies in FPGA due to the interference on the shared resources. Restuccia [39] provided a fine-grained model of the AXI bus and AXI interconnects in FPGA SoCs. Mattheeuws proposed a

TABLE XIII: Changing memory configurations to reduce the estimated WCET bound.

CNN Configurations		**CNN 1**			**CNN 2**		
		Measured	Our	Our/Measured	Measured	Our	Our/Measured
Mobv2 & Yolov3	**conf 1**	5.16	16.75	3.25	70.69	126.77	1.79
	conf 2	4.69	12.90	2.75	70.80	137.66	1.94
	conf 3	5.07	15.58	3.08	134.07	146.06	1.08
	conf 4	5.30	14.65	2.77	95.93	125.31	1.31
Squeeze & Yolov4	**conf 1**	3.88	8.07	2.08	50.72	85.27	1.68
	conf 2	3.97	5.46	1.38	51.02	96.71	1.90
	conf 3	5.33	6.96	1.31	98.73	102.49	1.04
	conf 4	3.86	6.54	1.82	74.29	83.89	1.13
PD_SSD & VpgNet	**conf 1**	11.20	16.90	1.51	8.759	20.69	1.68
	conf 2	11.88	16.06	1.35	9.22	26.07	2.83
	conf 3	11.64	20.52	1.76	14.45	22.61	1.57
	conf 4	11.94	21.68	2.10	6.93	10.63	1.53

methodology to characterize the interference to multi-core host processors caused by accelerators implemented in the FPGA SoCs [40]. Furthermore, Shikha Goel and Rajesh Kedia introduced machine learning-based frameworks [4], [5] designed to predict the average execution time for CNN tasks th multiple DNN accelerators on FPGA. Such capability for average performance estimation facilitates the design space exploration of DPU architectures on a resource constraints target FPGA [41], eliminating the need to exhaustively explore all possible DPU architectures.

Restuccia and Biondi completed the first work that addresses time-predictability for FPGA-accelerated DNNs [8]. They presented an execution model for the Xilinx DPU and a response-time analysis to bound the inference time of DNN models when executed on the DPU. However, the propose timing analysis framwork is only applicable to the case where a single DPU performs a single task. [42] presented a timing analysis framework for FPGA accelerators, but does not consider the characteristics of deep learning applications and dedicated accelerators.

VIII. Conclusion

The paper for the first time proposes a generalized WCET analysis framework for CNN tasks under themulti-DPU environment regardless memory port configurations, and produces a relatively tight estimated bound. Our approach can also be applied to the simpler special case of the single-DPU environment. Compared with the existing work for single-DPU environment, the WCET bound obtained by our approach is tightened by more than 27% on average. Moreover, some hint for how to configure memory port to reduce the estimated WCET bound is proposed. Benefited from the tight estimated WCET bound, a large resource over-provisioning in practical real-time system design can be avoided.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (Grant No.62432005, 62302270), Shandong Provincial Natural Science Foundation (Grant No.ZR20220F003, ZR2024MF099), Department of Science & Technology of Shandong Province (Grant No. SYS202201), Quan Cheng Laboratory (Grant No. QCLZD202302), Taishan Scholars Program (No. tsqn202211281).

References

[1] Z. Peng, J. Yang, T.-H. Chen, and L. Ma, "A first look at the integration of machine learning models in complex autonomous driving systems: A case study on apollo," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1240–1250.

[2] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network inference accelerator (2018)," *arXiv preprint arXiv:1712.08934*.

[3] *Vitis AI User Guide (UG1414 v2.0)*. AMD Xilinx, 2022.

[4] S. Goel, R. Kedia, M. Balakrishnan, and R. Sen, "Infer: Interference-aware estimation of runtime for concurrent cnn execution on dpus," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 66–71.

[5] S. Goel, R. Kedia, R. Sen, and M. Balakrishnan, "Express: Cnn execution time prediction for dpu design space exploration," in *2022 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2022, pp. 1–2.

[6] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[7] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, "A survey of timing verification techniques for multi-core real-time systems," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–38, 2019.

[8] F. Restuccia and A. Biondi, "Time-predictable acceleration of deep neural networks on fpga soc platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 441–454.

[9] G. Lacey, G. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," 02 2016.

[10] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2015.

[11] L. Bai, Y. Zhao, and X. Huang, "A cnn accelerator on fpga using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, pp. 1415–1419, 2018.

[12] T. Wang, T. Geng, A. Li, X. Jin, and M. C. Herbordt, "Fpdeep: Scalable acceleration of cnn training on deeply-pipelined fpga clusters," *IEEE Transactions on Computers*, vol. 69, pp. 1143–1158, 2020.

[13] M. E. Louise Crockett, Ross Elliot, *The ZYNQ Book*. University of Strathclyde Glasgow, 2015.

[14] *AMBA AXI and ACE Protocol Specification (Version H.c)*. ARM.

[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[17] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 04 2018.

[18] 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 *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3213–3223.

[19] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella Ferrer, and F. J. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the Nvidia Xavier," in *31st Euromicro Conference on Real-Time Systems*, vol. 23, 2019.

[20] *DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide (PG338 v3.4)*. AMD Xilinx, 2022.

[21] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[22] Y. Xing, S. Liang, L. Sui, X. Jia, J. Qiu, X. Liu, Y. Wang, Y. Shan, and Y. Wang, "Dnnvm: End-to-end compiler leveraging heterogeneous optimizations on fpga-based cnn accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2668–2681, 2019.

[23] W. Zhang, M. Lv, W. Chang, and L. Ju, "Precise and scalable shared cache contention analysis for wcet estimation," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1267–1272.

[24] *AXI Performance Monitor v5.0 LogiCORE IP Product Guide (PG037 v5.0)*. AMD Xilinx, 2017.

[25] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," 2016.

[26] S. Lee, J. Kim, J. Shin Yoon, S. Shin, O. Bailo, N. Kim, T.-H. Lee, H. Seok Hong, S.-H. Han, and I. So Kweon, "Vpgnet: Vanishing point guided network for lane and road marking detection and recognition," in *Proceedings of the IEEE ICCV*, 2017, pp. 1947–1955.

[27] L. Wei, A. Dragomir, E. Dumitru, S. Christian, R. Scott, F. Cheng-Yang, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*, 2016.

[28] *System Integrated Logic Analyzer v1.1 LogiCORE IP Product Guide(PG261 v1.1)*. AMD Xilinx, 2022.

[29] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 136–143, 2019.

[30] S. Kala, B. R. Jose, J. Mathew, and S. Nalesh, "High-performance cnn accelerator on fpga using unified winograd-gemm architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, pp. 2816–2828, 2019.

[31] D.-T. Nguyen, T. N. Nguyen, H. Kim, and H.-J. Lee, "A high-throughput and power-efficient fpga implementation of yolo cnn for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, pp. 1861–1873, 2019.

[32] S. Goel, M. Balakrishnan, and R. Sen, "Energynn: Energy estimation for neural network inference tasks on dpu," *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 64–68, 2021.

[33] J. Zhu, L. Wang, H. Liu, S. Tian, Q. Deng, and J. Li, "An efficient task assignment framework to accelerate dpu-based convolutional neural network inference on fpgas," *IEEE Access*, vol. 8, pp. 83 224–83 237, 2020.

[34] T. Wu, W. Liu, and Y. Jin, "An end-to-end solution to autonomous driving based on xilinx fpga," *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 427–430, 2019.

[35] W. Zhang and J. Yan, "Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches," *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 455–463, 2009.

[36] S. Wegener, "Towards multicore wcet analysis," in *WCET*, 2017.

[37] P. Benedicte, C. Hernández, J. Abella, and F. J. Cazorla, "Hwp: Hardware support to reconcile cache energy, complexity, performance and wcet estimates in multicore real-time systems," in *ECRTS*, 2018.

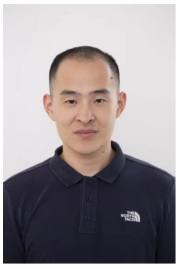
[38] H. Shah, K. Huang, and A. Knoll, "Timing anomalies in multi-core architectures due to the interference on the shared resources," *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 708–713, 2014.

[39] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni and G. C. Buttazzo, "Modeling and analysis of bus contention for hardware accelerators in fpga socs," in *ECRTS*, 2020.

[40] M. Mattheeuws, B. Forsberg, A. Kurth, and L. Benini, "Analyzing memory interference of fpga accelerators on multicore hosts in heterogeneous reconfigurable socs," *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1152–1155, 2021.

[41] R. Kedia, S. Goel, M. Balakrishnan, K. Paul, and R. Sen, "Design space exploration of fpga-based system with multiple dnn accelerators," *IEEE Embedded Systems Letters*, vol. 13, no. 3, pp. 114–117, 2020.

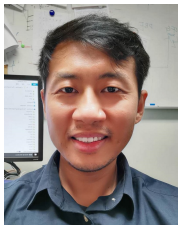
[42] F. Restuccia, M. Pagani, A. Biondi, M. Marinoni, and G. Buttazzo, "Bounding memory access times in multi-accelerator architectures on fpga socs," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 154–167, 2022.



Wei Zhang is currently an associate professor at the School of Cyber Science and Technology, Shandong University, Qingdao, China. He received his Ph.D. degree from The Hong Kong Polytechnic University, in 2021. His research interests include design, analysis and optimization of real-time systems and intermittent computing systems.

Yunlong Yu received his B.Sc. degree from Shandong University, Jinan, China in 2023. He is currently pursuing Ph.D. degree at Shandong University. His research interests include real-time system, embedded system, and High Level Synthesis.

Xiao Jiang received his M.Sc. degree from Shandong University, Jinan, China in 2023. He is currently working at Ant Group. His research interests include real-time system, embedded system, and High Level Synthesis.



Nan Guan is currently an associate professor at the Department of Computer Science, City University of Hong Kong. He received his BE and MS from Northeastern University, China in 2003 and 2006, respectively, and PhD from Uppsala University, Sweden in 2013. His research interests include real-time embedded systems and cyber-physical systems. Recently, he is working on cutting-edge research on embedded systems and IoT.

Naijun Zhan received the B.Sc. degree in mathematics and the M.Sc. degree in computer science from Nanjing University, Nanjing, China, in 1993 and 1996, respectively, and the Ph.D. degree in computer science from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2000. He was with the Faculty of Mathematics and Information, University of Mannheim, Mannheim, Germany, as a research fellow, from 2001 to 2004. Since then, he joined the Institute of Software, Chinese Academy of Sciences, Beijing, China, as an Associate Research Professor, and later was promoted to be a Full Research Professor in 2008, and a Distinguished Professor in 2015. Currently, he is a full professor at the Department of Computer Science, Peking University. His research interests include real-time, embedded and hybrid systems, program verification, concurrent computation models, and modal and temporal logics.

Lei Ju Dr Ju received his Ph.D. in 2010 from School of Computing, National University of Singapore. In 2011, he started working as an associate professor in School of Computer Science and Technology, Shandong University. His research interests focus on design, analysis and optimization of real-time systems and embedded networks. He has authored a number of referred publications (including DAC, RTAS, DATE), and served as the technical program committee of several international conferences.