A Fast, Accurate, and Comprehensive PPA Estimation of Convolutional Hardware Accelerators

Leonardo Rezende Juracy, Alexandre de Morais Amory, and Fernando Gehm Moraes, *Senior Member, IEEE*

***Abstract*— Convolutional Neural Networks (CNN) are widely adopted for Machine Learning (ML) tasks, such as classification and computer vision. GPUs became the reference platforms for both training and inference phases of CNNs due to their tailored architecture to the CNN operators. However, GPUs are power-hungry architectures. A path to enable the deployment of CNNs in energy-constrained devices is adopting hardware accelerators for the inference phase. However, the literature presents gaps regarding analyses and comparisons of these accelerators to evaluate Power-Performance-Area (PPA) trade-offs. Typically, the literature estimates PPA from the number of executed operations during the inference phase, such as the number of MACs, which may not be a good proxy for PPA. Thus, it is necessary to deliver accurate hardware estimations, enabling design space exploration (DSE) to deploy CNNs according to the design constraints. This work proposes a fast and accurate DSE approach for CNNs using an analytical model fitted from the physical synthesis of hardware accelerators. The model is integrated with CNN frameworks, like TensorFlow, to generate accurate results. The analytic model estimates area, performance, power, energy, and memory accesses. The observed average error comparing the analytical model to the data obtained from the physical synthesis is smaller than 7%.**

***Index Terms*— CNN, convolutional hardware accelerator, power-performance-area (PPA) estimation, design space exploration (DSE).**

I. Introduction

MACHINE Learning (ML) is a sub-area of artificial intelligence that contains a class of algorithms able to solve problems involving knowledge and "learning" characteristics from determined patterns. The ML decision capability [1] enables its adoption in classification and pattern recognition problems. Many applications can use ML, such as computational vision, virtual reality, voice assistants, chatbots, health care, and self-driving vehicles [2], [3], [4], [5].

Manuscript received 12 July 2022; revised 30 August 2022; accepted 5 September 2022. Date of publication 15 September 2022; date of current version 9 December 2022. This work was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant 309605/2020-2; in part by the Fundação de Amparo à pesquisa do Estado do RS (FAPERGS) under Grant 21/2551-0002047-4; and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001. This article was recommended by Associate Editor J. Di. *(Corresponding author: Fernando Gehm Moraes.)*

Leonardo Rezende Juracy and Fernando Gehm Moraes are with the School of Technology, PUCRS University, Porto Alegre 90619-900, Brazil (e-mail: leonardo.juracy@acad.pucrs.br; fernando.moraes@pucrs.br).

Alexandre de Morais Amory is with Scuola Superiore Sant'Anna, 56127 Pisa, Italy (e-mail: alexandre.amory@santannapisa.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3204932.

Digital Object Identifier 10.1109/TCSI.2022.3204932

One of the most common ways to deliver ML is by using Artificial Neural Networks (ANN), particularly Convolutional Neural Networks (CNN). CNNs have the advantage of having sparse connections, in contrast to fully connected ANNs, where all neurons of one layer are connected to all neurons of the next layer.

A CNN contains four main layers: (i) convolutional layer, which is the CNN core and performs the synapses by multiplying and accumulating weights and input feature maps; (ii) activation function, a nonlinear transformation sent to the next layer of neurons; (iii) pooling layer, used to reduce the amount of data processed by the CNN; (iv) fully connected layer, used in the classification result.

The deployment of CNNs applications comprises two phases [6]. The first is the training, which defines the weights of the synapse. The second is inference, which uses the weights previously computed during the training phase to classify or predict output values based on the inputs. A CNN can correctly classify inputs not used in the training phase.

The success of CNNs led to the development of frameworks that help developers to build their models by offering mechanisms required for training and inference. Examples of frameworks include Caffe [7], Pytorch [8] and TensorFlow [9]. These frameworks use a high-level approach to abstract the implementation of functions, such as convolution, and aid in implementing CNN applications. Also, these frameworks abstract the training phase by implementing functions like back-propagation algorithms.

GPUs became the reference platform for training and inference due to their tailored architecture to the CNN operators [10], [11], reducing the time spent in training. The main GPU drawback is its energy consumption. Considering energy-constrained applications, such as the Internet of Things (IoT), autonomous driving, and wearable devices, the adoption of specialized hardware for computing inference became a trend.

CNN hardware accelerators are a suitable replacement for CPUs and GPUs for the inference phase [12]. CNN accelerators can reduce power consumption and/or improve throughput [10], [13], [14]. Also, consumer products are increasingly receiving these blocks [15], [16], [17]. Most of these accelerators are application-specific and can focus only on one characteristic to optimize, such as power, performance, or area [18], [19].

The literature presents gaps regarding analyses and comparisons of CNN hardware accelerators. Even with a representative number of accelerators using different implementations, there is a lack of proposals exploring the trade-offs between

implementations. For example, Eyeriss proposes a comparison between accelerators but lacks performance or area trade-offs evaluation [10]. Also, some works compare accelerators considering different technology nodes, resulting in an unfair analysis [20].

For design space exploration (DSE), literature presents works using analytic models integrated into frameworks and system simulators. Analytical models estimate the power, performance, and area (PPA) of hardware accelerators to a given hardware constraint [21], [22]. System simulators [23], [24] describe accelerators in high-level languages, like Python and C++, reducing the design time and providing PPA evaluation. Both analytical and simulator approaches present as drawbacks the PPA accuracy, typically estimated from the number of required MACs (multiply-accumulate operators) [23], [25]. Despite the efforts to increase the abstraction level to implement CNN accelerators using high-level synthesis (HLS) [26], this approach presents several challenges, as (i) long synthesis time; (ii) huge design space; (iii) effective impact of design parameters in the hardware [27].

The goal of this paper is to propose a method to perform a $comprehensive$ DSE in a $fast$ and $accurate$ way, integrated with an ML framework, to estimate the costs of the hardware accelerator parameters. Besides the CNN parameters, the proposed method also allows the user to change hardware parameters, such as dataflow type (weight stationary (WS), input stationary (IS), and output stationary (OS)), memory type, memory latency and accelerator frequency. We also demonstrated that estimating PPA values using MACs produces an inaccurate PPA estimation.

The original contributions of this work include:

1) Adoption of an ML framework (TensorFlow) as a front-end to perform DSE for CNN hardware accelerators;
2) The use of data from the physical synthesis of the complete hardware accelerators, not only from basic components, to obtain accurate results;
3) A method to fairly compare different CNN hardware accelerators, allowing to compare accelerators with different dataflows, considering the same characteristics, such as the technology node, frequency, and memory type;
4) An analytical method to perform DSE. A set of equations derived from the physical synthesis flow enables the ML framework to estimate power, performance, and area. The analytical model, integrated into the ML framework, is the key to obtaining a fast and accurate DSE.

This paper is organized as follows. Section II reviews CNN hardware accelerators and simulators, positioning our work with regard to the state-of-the-art. Section III introduces the convolutional accelerator architecture proposed in this work. Section IV presents two DSE flows, the first based on physical synthesis and the latter on MAC counting. Section V details the main original contribution of this work, the proposed DSE method based using an analytical approach to produce PPA results. Section VI compares the proposed DSE method with the flow described in Section IV and with related works. Finally, Section VII concludes this paper, pointing out the direction for future work.

II. State-Of-the-Art

Section II-A describes frameworks enabling the design and DSE of hardware accelerators. Section II-B presents CNN simulators. Section II-C compares qualitatively the presented works with regard to our proposal.

Besides CNNs, literature also presents Spiking Neural Network (SNN) proposals [28], [29]. SNNs are based on firing patterns computation, similar to the human brain, and are commonly implemented using an analog approach. The advantage of using SNNs is reduced area and energy consumption once analog components can be smaller than a digital adder or multiplier, allowing the connection of thousands of neurons with low area cost. However, our focus is on digital implementations, being SNNs out of the scope of this work.

A. Frameworks for CNN Hardware Accelerators

MLPAT [30] is a framework that allows modeling power, area, and timing for machine learning accelerators. MLPAT models components such as systolic arrays, on-chip memory, and activation pipeline. Also, MLPAT supports different precision types, which allows validating the trade-off between accuracy and precision, and different dataflows, such as WS and OS. As input, the MLPAT allows specifying the accelerator architecture, the circuit, and the technology. The framework generates an optimized chip representation to report the results, such as area, power, and performance.

MAESTRO [31], [32] is a framework to describe and analyze neural network hardware, which allows obtaining the hardware cost to implement a target architecture. It has a domain-specific language to describe the dataflow that allows specifying the number of PEs, memory size, and NoC bandwidth parameters. The results generated by the framework are focused on performance analyses. In recent work, MAESTRO was used to estimate tradeoffs between execution time and energy efficiency for CNN models, such as VGG and AlexNet.

Timeloop [23] is a DSE framework for CNNs. It can emulate a set of accelerators, such as NVDLA [33]. Timeloop focuses on the convolution layer analyses. Timeloop uses as input a workload description, such as input dimension and weight values, a hardware architecture description, such as arithmetic modules, and hardware constraint. Instead of using a cycle-accurate simulator, Timeloop uses data transfers deterministic behavior to perform analytic analyses. As energy models, Timeloop has memory, arithmetic units, and wire/network models based on TSMC 16nm FinFET.

Accelergy [25] allows estimating the energy of accelerators without a complete hardware description, using a library of basic components. Accelergy uses a high-level architectural description to capture the circuit behavior characteristics, such as memory reads. Accelergy considers the number of memory reads and the memory access pattern, which can be random or acccess at the same address repetitively.

Heidorn *et al.* [21] propose an analytical model that estimates throughput and energy to a given hardware constraint. A DSE is proposed to determine the accelerator architecture limits in terms of throughput, number of parallel operations, and memory. The Authors propose an accelerator to evaluate the model with a tile-local memory, a bus, and a coarse-grained reconfigurable array (CGRA). Each CGRA presents a two-dimensional array of PEs, and the accelerator can have more than one CGRA to parallelize the processing.

Zhao *et al.* [22] propose an analytical performance predictor to estimate energy, throughput, and latency for ASIC and FPGA. The predictor uses DNN models, hardware architecture, dataflows types, and hardware cost regarding a technology node. The results are generated with AlexNet and SkyNet CNN models, with Eyeriss, an FPGA implementation from [34], and synthesized results of a proposed accelerator.

DNNExplorer [35] is a framework for DSE of ML accelerators. DNNExplorer supports machine learning frameworks (Caffe and PyTorch), besides three accelerator architectures. The architecture also supports WS and IS dataflows. This framework adopts analytical models to estimate performance and hardware configuration.

Gemmini [36] is an open-source systolic array generator that allows evaluating deep-learning architectures. Gemmini generates a custom ASIC accelerator for matrix multiplication based on a systolic array architecture. Gemmini is compatible with the RISC-V Rocket ecosystem [37].

Interstellar [38] is a DSE framework that uses Halide language (https://halide-lang.org) to generate hardware and compare different accelerators, such as different dataflows (WS, OS, RS) in 2D arrays and a MAC tree schemes. The authors propose a systematic approach to describe the design space of DNN accelerators using Halide. The framework also optimizes the memory hierarchy.

DeepOpt [39] is a DSE framework to explore ASIC implementation of systolic hardware accelerators for CNNs. The main goal of this DSE is to reduce the number of memory accesses based on hardware characteristics like on-chip SRAMs and the number of parallel PEs. The DeepOpt uses a search tree to schedule the convolution process. Thus, it is possible to minimize the number of accesses from memory by modeling memory access patterns (weight and output stationary) and pruning branches from the search tree.

Karbachevsky *et al.* [40] propose a method to estimate area and power values based on the bit operations performed (BOP) metric [41]. BOP is the number of bit operations required to perform the calculation, defined by the input bit size, output bit size, number of inputs, and number of outputs. According to the authors, BOP metric allows estimating the area and power required by accelerator hardware with high accuracy in the early stages of the design process. Also, the method can show the trade-off between the number PEs and the bottlenecks caused by the parameters quantization, such as memory bandwidth or computational resources.

Ferianc *et al.* [42] propose a method to improve the performance of DSE analyses. The method is based on a Gaussian process regression model parameterized by the features of the accelerator and the target CNN, such as filter, channel, and data parallelism. The method is capable of predicting the hardware latency and energy, and was compared to machine learning-based methods to perform DSE (linear regression, Gradient tree boosting, and neural network).

Aladdin [43] is a pre-RTL power-performance accelerator modeling framework. It estimates performance, power, and area. Aladdin infrastructure uses dynamic data dependence graphs (DDDG) to represent accelerators. The DDDG is generated from a C program and allows Aladdin to report the program dependencies and resource constraints.

B. Hardware Simulators

SCALE-Sim (Systolic CNN Accelerator Simulator) [44], [45] is a systolic array cycle-accurate simulator. This simulator allows configuring micro-architectural features such as array size, array aspect ratio, scratchpad memory size, and dataflow mapping strategy. Also, it is possible to configure system integration parameters, such as memory bandwidth. SCALE-Sim simulates convolutions and matrix multiplications, and models the compute unit as a systolic array. Also, it allows simulation in a system context with CPU and DMA components.

STONNE [24] is a cycle-accurate architecture simulator for CNNs which allows end-to-end evaluation. It is connected with Caffe framework [7] to generate the CNNs, and models the MAERI accelerator [46]. The results are focused on performance and hardware utilization. To estimate area and energy, STONNE uses the Accelergy energy estimation methodology [25], which considers basic modules to calculate the energy values, such as adders.

AccTLMSim [47] is a pre-RTL cycle-accurate CNN accelerator simulator based on SystemC transaction-level modeling (TLM). The simulator allows maximizing the throughput performance for a given on-chip SRAM size. An accelerator is proposed to validate the simulator. AccTLMSim is focused only on performance, not power or area.

C. Summary Related to DSE Frameworks and Simulators

Table I summarizes the reviewed works. The second column indicates whether the work has integration with high-level modeling CNN frameworks, such as TensorFlow and Caffe. The third and fourth columns are related to the evaluated metrics. The third column presents metrics based on basic components, such as MACs and register files. The fourth column shows the evaluated metrics regarding the entire convolution, our original contribution.

Maestro [32] does not allow the accelerator simulation, limiting the performance evaluation (e.g., throughput). SCALE-Sim [45] does not provide power or energy results. MLPAT [30] and Timeloop [23] provide PPA based on basic operations, such as adders and multipliers. Methods relying on operations counting do not consider how these operators are interconnected (e.g., 1D or 2D systolic arrays or adder trees), resulting in imprecise hardware metrics.

Works [21] and [22] show analytical results for power, performance, and area. Also, [22] consider features like the dataflow type, which can contribute to the power consumption. Similar occurs to Aladdin [43].

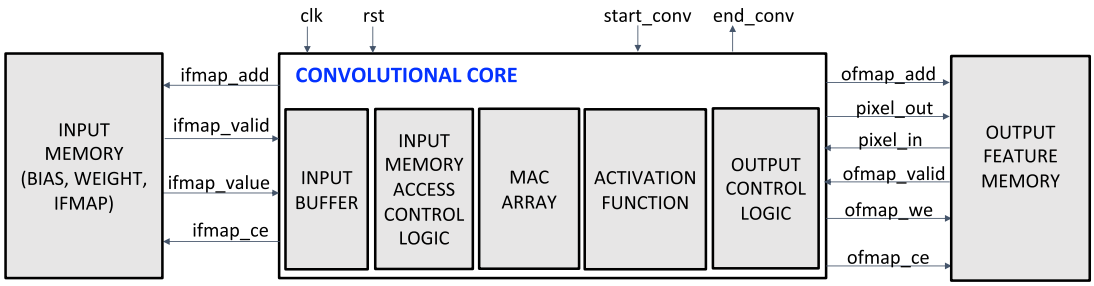


Fig. 1. Generic architecture and the modules required to build the convolutional accelerators.

TABLE I
DSE FRAMEWORKS AND SIMULATORS STATE-OF-THE-ART SUMMARY

Work	Integration with CNN frameworks	Evaluation metrics based on basic components	PPA analyses based on entire convolution
MLPAT [30]	No	PPA	No
Maestro [32]	No	Performance	No
Timeloop [23]	No	PPA	No
Accelergy [25]	No	Power	No
[21]	No	PPA	No
[22]	No	PPA	No
DNNExplorer [35]	Caffe, PyTorch	Performance	No
Gemmini [36]	No	Performance	No
Interstellar [38]	No	Power	No
DeepOpt [39]	No	Performance, Power	No
[40]	No	Area	No
[42]	No	Performance, Power	No
Aladdin [43]	No	PPA	No
SCALE-Sim [45]	No	Performance, Area	No
STONNE [24]	Caffe	Performance	No
SimuNN [48]	TensorFlow	PPA	No
AccTLMSim [47]	No	Performance	No
This work	**TensorFlow**	**No**	**Yes**

Works like Gemmini [36], Interstellar [38], DeepOpt, [40], and [42], have a limited PPA analyses. Also, [40] claims that the main effect of changing the circuit frequency is to reduce power consumption. However, this is not exact since logical synthesis with different frequency constraints produces different area values.

STONNE [24] and SimuNN [48] present flows integrated to frameworks to model CNNs. However, SimuNN has an energy estimation based on basic elements, not considering data movement through the accelerator. STONNE does not address power estimation, but the authors argue that it is possible to integrate STONNE with Accelergy (which only evaluates power). DNNExplorer [35] also allows frameworks to model CNNs, but lacks PPA analyses. SCALE-Sim and AccTLMSim lack integration with frameworks to model CNNs.

Considering the reviewed works, we identify the following gaps:

1) Comprehensive PPA analyses, not considering all performance figures;
2) An analytical method based on the entire convolution accelerator to produce an accurate PPA estimation;
3) A framework or environment to produce a fast DSE.

Our proposal provides a comprehensive PPA estimation method that integrates a CNN modeling framework to perform DSE using data from actual CNNs. The DSE starts with a framework to model CNNs (TensorFlow) and executes an analytical analysis considering the accelerator architecture, allowing a fast and accurate DSE. The adoption of TensorFlow was a design choice, and other frameworks, such as Caffe or Pytorch, can be used as a front-end to execute the training and weight extraction.

III. CONVOLUTIONAL ACCELERATOR ARCHITECTURE OVERVIEW

Figure 1 shows the accelerator model, with a unified interface with input and output external memories. External memories play an important role in the total accelerator energy, being mandatory to evaluate its cost [10]. This model contains three modules:

- **INPUT memory**: stores the IFMAP, filter weights, and bias values. It acts as a read only memory for the accelerator;
- **Convolutional core**, executes the convolution;
- **Output Feature Map (OFMAP) memory**, stores partial and complete convolution values.

The main modules of the convolutional core include:

- **Input buffer**, used to reduce the number of input memory readings. According to the accelerator type, this buffer may store, e.g., an input channel, a set of rows of the input channel, or a set of weights;
- **Input memory access control logic**, used to control the input memory access. This module is an FSM configured according to the dataflow type;
- **MAC array**, is the unit that processes the convolution operation, which may adopt a 2D (matrix) or 1D (vector). It contains MACs and registers;
- **Activation function**, a non-linear function applied to the OFMAP results. Examples are sigmoid, ReLU, leaky ReLU [49]. This works adopts the ReLU function ($max(0, x)$) due to its simpler hardware implementation;
- **Output control logic**, used to control the OFMAP memory access. It can be implemented with buffers to reduce memory access.

Our previous work, [50], details the architecture of the convolutional core (WS and 1D array). The focus of [50] is the convolutional core design, without modeling memory effects in the PPA, neither propose a DSE analytical model.

The dataflow type is related to the reused data and how the MAC array computes the convolution. We implemented in the MAC array the main dataflows used in the literature [51]:

- Weight Stationary – **WS**. Stores weights in an internal buffer, aiming their reuse. Thus, each weight value is read once from the input memory, and the convolution is performed using stationary weights and reads from memory IFMAP values.
- Input Stationary – **IS**. Stores an IFMAP window in an internal buffer to provide its reuse. The window size is equal to the filter size. The accelerator reads IFMAP values once and reads the weight values from memory.
- Output Stationary – **OS**. Stores partial convolution results in registers. The OS does not present buffers to store the inputs; each convolution fetches the IFMAPs and weight values from the input memory.

Figure 1 presents the set of signals between the convolutional core and memories, and signals used by the hardware controlling the accelerator (`start_conv` and `end_conv`). Both memories use a standard set of busses, such as addresses, output data (`ifmap_value` and `pixel_in`), input data (`pixel_out`), memory enable and write notification. Memories have different implementation technologies (e.g., SRAM or DRAM) and latencies, varying according to their specification. Thus, it is necessary to model the memory latency to estimate the power and performance of the convolutional accelerator. A validity signal (`ifmap_valid` and `ofmap_valid`) models the memory latency, indicating the end of memory accesses.

This unified memory interface makes it possible to implement different dataflows with distinct protocols, allowing a fair comparison between the accelerators.

The accelerators adopted for this work have a 3×3 MAC array, and the convolution stride equals 2. Despite being a design limitation, state-of-the-art CNNs adopt these values, such as VGG16 [52], ensuring that the proposed accelerators reflect real CNNs.

IV. Physical Synthesis and MAC-Based Flows

This Section introduces two flows to execute DSE. The first one uses a standard-cell synthesis flow. Results for one instance of each dataflow are the basis for the main original contribution of this work, the analytical DSE flow, presented in the next Section. The second flow uses number of MACs, a method used in related work.

A. Physical Synthesis Flow

The DSE physical synthesis flow performs both logical and physical synthesis steps and uses IFMAP and weights data from real CNNs to obtain PPA results. This flow includes the steps described below.

1) TensorFlow Modeling: Model the CNN application in the TensorFlow framework to generate the VHDL packages used in the RTL and gate-level simulations, and the gold model (expected outputs values). The VHDL packages contain the IFMAP, weights, bias, and OFMAP values. This step also generates parameters for configuring the RTL description, such as the size of the filters, OFMAP, and IFMAP. This step is generic, supporting different CNNs, such as MNIST or CIFAR10.

2) RTL Simulation: Verification of the accelerator description behavior. This step uses the VHDL packages, comparing the results against the gold model. It is necessary to check if the simulation output matches the expected values during the development of a new accelerator. Once validated the accelerator description, it is possible to bypass this step.

3) Physical Synthesis: This step comprises logical and physical synthesis. The logic synthesis inputs include the RTL accelerator description, technology files (LEF and LIB files), and constraints (such as clock frequency or power). The physical synthesis corresponds to the placement and routing. This step generates a gate-level netlist with annotated wire capacitances (SDF file).

4) Annotated Gate-Level Simulation and Power Estimation: Simulation of the annotated gate-level netlist, also using the VHDL packages and the gold model. This step may fail due to the applied constraints, such as clock frequency and input/output delays. In this case, the designer must modify the constraints used in the physical synthesis. The output of this simulation is a VCD file, with the switching activity induced by the CNN IFMAP and weight values. The power estimation tool uses the VCD file to estimate the accelerator power dissipation.

The execution of this flow produces an accurate PPA estimation for a given accelerator architecture with actual CNN data. However, it is necessary to execute this flow for each new set of weights and IFMAPs. The reason is that different data sets present different switching activities, changing the power dissipation. Also, the hardware may show differences due to the number of channels in a given layer or the IFMAP and OFMAP sizes, changing the number of bits in counters or the buffers' depth. Thus, we have an accurate PPA, but it requires a significant processing time, which takes hours.

B. MAC-Based Flow

This Section describes a DSE flow based on estimating the required number of MACs. Several works in the literature use the MAC-based method to evaluate the area, and power [21], [22], [23], [30], [48]. The MAC-based flow does not model memory accesses, having as primary goal a fast area and power estimation.

The MAC-based flow used in this work considers the power and area values related to MACs and registers extracted from the physical synthesis flow. The number of required MACs, registers, and the data width is a function of the accelerator design. The performance, i.e., the number of clock cycles to execute a complete convolution, is obtained through the RTL simulation.

The area and power accuracy of this flow are expected to be worse than the physical synthesis flow, as it does not consider the control circuitry (such as FSMs), buffers, and accesses to the memory.

V. Analytic DSE Flow

The proposed DSE flow analytically estimates the PPA of CNN layers using results obtained from the physical synthesis of one CNN layer. This flow is faster than the physical

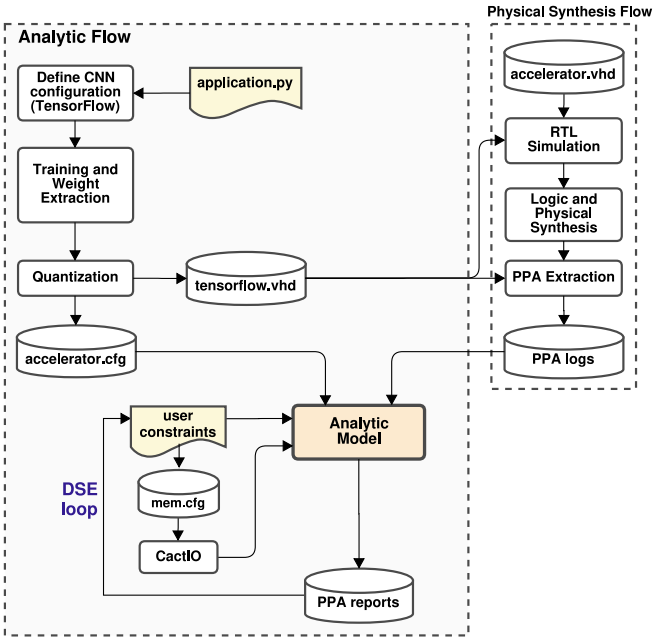


Fig. 2. DSE analytic flow for PPA extraction.

synthesis flow because the synthesis is executed once for each accelerator and is more accurate than the MAC-based flow once it considers the entire convolution hardware. Figure 2 presents the analytic DSE flow.

The flow starts with the training phase of the CNN using TensorFlow, according to the CNN configuration defined in the *application.py* file. The training phase ends when reaching the target accuracy. After the training phase, the flow executes a quantization step, converting 32-bit floating-point weights and IFMAP values to 16-bit integers using a fixed-point representation. Adopting integer values avoids floating-point arithmetic in the accelerator, reducing its area and power consumption. This step generates two output files: (i) *tensorflow.vhd*, with the IFMAP and weights values and the gold model (expected output values) of the first CNN layer; (ii) *accelerator.cfg*, with the CNN convolutional layers parameters.

The right side of Figure 2 corresponds to the physical synthesis flow, presented in Section IV-A. The flow executes the synthesis of the first convolutional layer for each dataflow. The remaining layers are estimated from the first layer results. From one side, this procedure avoids the synthesis of all layers, speeding up the DSE. On the other side, there is a penalty in the PPA accuracy, evaluated in the Results Section. The output of the physical synthesis flow is a set of PPA reports used during DSE.

The *Analytic Model* uses four inputs to execute the DSE loop: (i) configuration of the convolutional layers (*accelerator.cfg*); (ii) user's constraints; (iii) PPA reports related to the synthesis of the first convolutional layer; (iv) memory energy estimated by the Cacti-IO tool [53].

The *accelerator.cfg* file contains hardware parameters (italic parameters corresponds to a variable in the analytic model):

1) Word size (bits);
2) IFMAP size: single integer value (we assume square IFMAPs) - *IFMAP_D*;
3) Number of input channels: integer value - *InChannels*;
4) Number of output channels: integer value - *OutChannels*;
5) Filter size: single integer value (we assume square filters) - *Filter_D*;
6) Stride: integer value.

The *user's constraints* file contains constraints applied to the accelerator (italic parameters corresponds to a variable in the analytic model):

1) Clock period (ns);
2) 2D dataflow type: WS, IS, and OS. It is possible to use buffered versions (output buffer) for WS and IS;
3) Memory features: type (SRAM or DRAM) and its latency - *MemLat*;

For example, a designer wants to define the accelerator for a layer with the following characteristics: (1) 16-bit word size; (2) IFMAP size=32 (cifar10); (3) 3 input channels (RGB); (4) 16 output channels; (5) 3×3 filters; (6) stride=2. The designer executes the *DSE loop* by selecting the clock period, dataflow type, and memory parameters. The DSE loop may be executed up to reaching the user constraint, such as performance, area, or power.

The *analytic model* produces the following results:

- Power: power values for the accelerator, output buffer, and the sum of both (total power), mW;
- Performance: number of clock cycles required to execute the layer convolution;
- Area: area values for the accelerator, output buffer, and the sum of both (total area), μm^2;
- Accelerator energy: total power $\times$ number of cycles $\times$ clock period, pJ;
- Memory accesses:
  - Number of input memory reads (IFMAPs, weights and bias);
  - Number of input memory writes (always zero, once the input memory acts as a ROM);
  - Number of output memory reads (partial sums values);
  - Number of output memory writes (partial sums values, and OFMAPs);
- Memory read energy: total memory reads $\times$ energy per reading (estimated by Cacti-IO), nJ;
- Memory write energy: total memory writes $\times$ energy per writing (estimated by Cacti-IO), nJ;
- Total energy: accelerator energy $+$ memory read energy $+$ memory write energy, nJ.

The physical synthesis produces PPA reports for the accelerator core. The performance to compute the convolution of a given layer is calculated by the analytic model, process detailed on Section V-A. The analytical model calculates the effect of the memory accesses on performance and power (Section V-B). Another component that affects the PPA is the output buffers. The influence of these components is described in Section V-C.

A. Performance Estimation

Note that the OFMAP size is not a parameter in the *accelerator.cfg* file. The OFMAP size is a function of the IFMAP size ($IFMAP\_D$), filter size ($Filter\_D$), and the stride value. Equation 1 computes the OFMAP size.

$$OFMAP\_D = \left\lfloor \frac{IFMAP\_D - FILTER\_D}{Stride} + 1 \right\rfloor \quad (1)$$

For example: a 32 × 32 IFMAP, with 3 × 3 filters and stride=2 generates a 15 × 15 OFMAP.

The convolution performance is a function of the dataflow defined by the designer.

1) WS Dataflow: Equation 2 computes the number of clock cycles, $CyclesWS$, to execute the convolution of a given layer using the WS dataflow. Weights and bias are stationary, i.e., pre-loaded in a buffer.

$$CyclesWS = 6 \times OFMAP\_D^2 \times InChannels \times OutChannels \times (1 + MemLat) \quad (2)$$

where:

- 6 constant: number of clock cycles to read 9 (3 × 3) IFMAP values. Due to the stride value (equal to 2), each reading reuse one column, reducing memory accesses;
- $OFMAP\_D^2 \times InChannels$: number of convolutions to produce one output channel. Due to the accelerator pipeline implementation the IFMAP reading and the convolution occurs in parallel;
- The process is repeated for all output channels ($OutChannels$);
- The constant value added to the memory latency (1 clock cycle) corresponds to the address phase.

Equation 2 computes most of the clock cycles required to calculate the convolution of a given layer (>80%). The analytical model also computes the time to read the weights and the number of bubbles in the pipeline when it is necessary to return to the first X coordinate after $OFMAP\_D$ convolutions. For the sake of simplicity, the other equations are not presented.

2) IS Dataflow: Equation 3 computes the number of clock cycles, $CyclesIS$, to execute the convolution of a given layer using the IS dataflow. In the IS approach values read from the IFMAP are stationary, i.e., they are used to compute a partial output value at each output channel.

$$\begin{aligned} Term1 &= OutChannels \times (1 + MemLat) \\ &\quad + (Filter\_D^2 \times OutChannels \times InChannels) \\ &\quad \times (1 + MemLat) \\ Term2 &= Filter\_D^2 \times OFMAP\_D^2 \\ &\quad \times InChannels \times (1 + MemLat) \\ Term3 &= (9 \times OFMAP\_D^2 \times InChannels \\ &\quad \times OutChannels) \\ CyclesIS &= Term1 + Term2 + Term3 \end{aligned} \quad (3)$$

where:

- $Term1$: cycles to load bias and weights, and store in internal buffers. The number of bias values is equal to the number of $OutChannels$;
- $Term2$: cycles to read $Filter\_D \times Filter\_D$ IFMAP values read from the memory;
- $Term3$: cycles to execute all convolutions of the layer.

Equation (3) computes most of the clock cycles required to calculate the convolution of a given layer ($CyclesIS >$ 83% for IS and $CyclesIS >$ 90% for buffered IS). As in the previous dataflow, the time spent with bubbles is also accounted by the proposed analytic model.

The WS dataflow reads the IFMAP for each partial result (Equation 2). For the IS dataflow, a partial reading of the IFMAP is performed (Term 2 of Equation 3), reusing these values for all partial convolutions (Term 3 of Equation 3).

3) OS Dataflow: The OS dataflow does not have buffers for IFMAP and weight values. Thus, the OS dataflow reads 18 values from the input memory to execute each convolution (9 weights and 9 IFMAP values). Due to the pipeline implementation, the convolution occurs in parallel to the memory reading.

Equation 4 computes most of the clock cycles required to calculate the convolution of a given layer (>98%). The analytic model considers the number of clock cycles to write in the OFMAP memory and the bubbles in the pipeline. The number of memory readings is the main difference concerning the WS dataflow (Equation 2), which is larger in the OS dataflow.

$$CyclesOS = 18 \times OFMAP\_D^2 \times InChannels \times OutChannels \times (1 + MemLat) \quad (4)$$

B. Memory Accesses Estimation

The number of memory accesses is a function of the dataflow defined by the designer.

1) Memory Readings for the WS Dataflow: Equation 5 presents the number of memory readings for WS and buffered WS dataflows.

$$\begin{aligned} Term1 &= 6 \times (OFMAP\_D + 5) \\ &\quad \times InChannels \times OutChannels \\ Term2 &= (Filter\_D^2 + 1) \times InChannels \\ &\quad \times OutChannels \\ Term3 &= 6 \times OFMAP\_D^2 \times InChannels \\ &\quad \times OutChannels \\ MemReadWS &= Term1 + Term2 + Term3 \end{aligned} \quad (5)$$

where:

- $Term1$: refers to "invalid" readings. At the end of each row, the WS accelerator accesses memory locations not used in the convolution. It would be possible to avoid these readings at the cost of more control logic in the hardware. Our design choice was to keep the hardware simple.
- $Term2$: number of reads to load weight and bias values;
- $Term3$: number of reads to load IFMAP values (core of Equation 2).

2) Memory Readings for the IS Dataflow: Equation 6 presents the number of memory readings for IS and buffered IS dataflows.

$$\begin{aligned} Term1 &= OutChannels + ((Filter\_D^2) \\ &\quad \times InChannels \times OutChannels) \\ Term2 &= Filter\_D^2 \times OFMAP\_D^2 \times InChannels \\ MemReadIS &= Term1 + Term2 \end{aligned} \tag{6}$$

where:

- $Term1$: number of reads to load bias and weight values;
- $Term2$: number of reads to load $Filter\_D \times Filter\_D$ IFMAP values from the memory.

3) Memory Readings for the OS Dataflow: Equation 7 presents the number of memory readings for the OS dataflow.

$$\begin{aligned} MemReadOS &= 18 \times OFMAP\_D^2 \\ &\quad \times InChannels \times OutChannels \end{aligned} \tag{7}$$

It is possible to observe the smaller number of memory accesses for the IS dataflow (Equation 6 $Term2$) compared to the WS and OS dataflows (Equations 5 $Term3$ and 7).

4) Memory Writings: Equation 8 computes the number of memory writings for a buffered accelerator (WS and IS). The output buffer reduces the memory writes once partial results are stored on it.

$$\begin{aligned} OfmapWrites\ (buffered\ acc.) &= OFMAP\_D^2 \\ &\quad \times OutChannels \end{aligned} \tag{8}$$

Equation 9 computes the number of memory writings for a non-buffered accelerator (WS, IS, and OS). Non-buffered accelerators read and write partial sums in the output memory.

$$\begin{aligned} OfmapWrites\ (non\ buffered\ acc.) &= OFMAP\_D^2 \\ &\quad \times OutChannels \times InChannels \end{aligned} \tag{9}$$

C. Output Buffer Area and Power Estimation

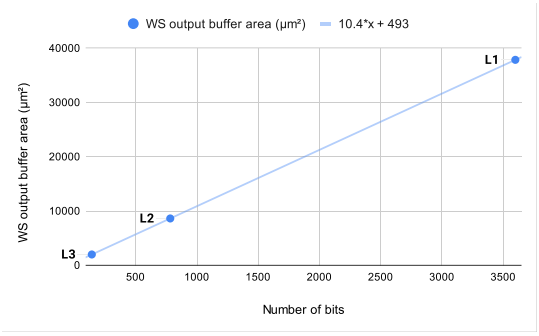
Two dataflows may present an output buffer: WS and IS. The output buffer area and power are obtained from interpolation. The data source for the interpolation are the results obtained from the physical synthesis flow for a three-layer Cifar10 CNN (presented in Section VI). The variable *NumBits* is the number of bits of each output buffer. Equation 10 computes the number of bits for the WS output buffer. The WS output buffer has the size of one OFMAP channel ($OFMAP\_D^2$) multiplied by the word size (16 bits).

$$NumBits(WS) = OFMAP\_D^2 \times 16 \tag{10}$$

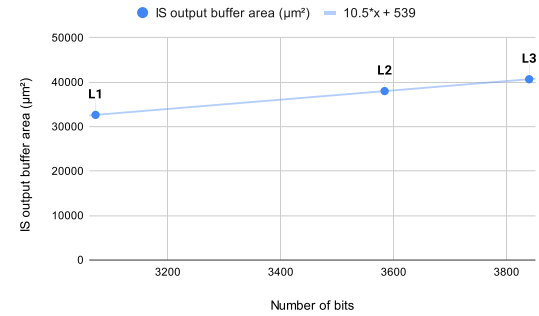
Equation 11 computes the number of bits for the IS output buffer. The IS dataflow computes one line of results ($OFMAP\_D$), for all output channels ($OutChannel$).

$$NumBits(IS) = OFMAP\_D \times OutChannel \times 16 \tag{11}$$

Figure 3 presents in the x-axis the number of bits for each dataflow, and in the y-axis the area. The Cifar10 CNN has three convolutional layers. The OFMAP sizes for each layer are (Figure 4): L1: 15×15, 16 output channels; L2: 7×7, 32 output channels; L3: 3×3, 64 output channels. According to Equation 10 the size of the WS output buffers decreases from layer 1 to layer 3 since the OFMAP size reduces. On the other side, according to Equation 11 the size of the IS output buffers increases from layer 1 to 3 due to the increase in the number of output channels.



(a) Buffered WS



(b) Buffered IS

Fig. 3. Output buffer area results obtained from the physical synthesis flow, for the three layers of Cifar10 CNN.

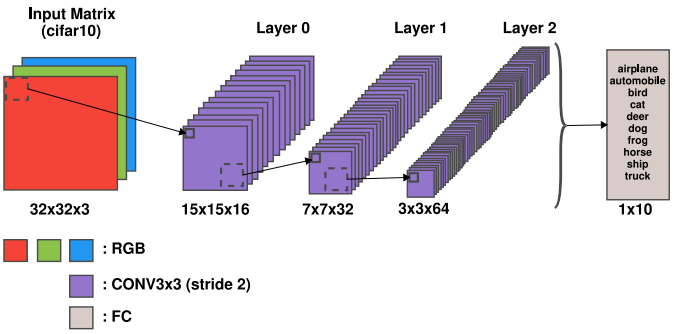


Fig. 4. Cifar10 CNN.

The interpolation of the area results are used to compute the output buffer area. Equations 12 and 13 compute the output buffer area for WS and IS, respectively.

$$WSOutputBuffArea = (10.4 \times NumBits) + 493 \tag{12}$$

$$ISOutputBuffArea = (10.5 \times NumBits) + 539 \tag{13}$$

The same interpolation method is applied to obtain the power consumption due to the output buffers. However, each memory type has a interpolation equation, due to the latency access. Equations 14 and 15 compute the output buffer power dissipation for WS and IS using a SRAM, respectively.

Equations 16 and 17 compute the output buffer power dissipation for WS and IS using a DRAM, respectively.

$$SRAM\_WSOutputBuffPower = 0.0792 + (0.000305 \times NumBits) + (0.0000000117 \times NumBits^2) \quad (14)$$

$$SRAM\_ISOutputBuffPower = -5.4 + (0.00346 \times NumBits) + (-0.000000402 \times NumBits^2) \quad (15)$$

$$DRAM\_WSOutputBuffPower = 0.0794 + (0.000245 \times NumBits) + (0.0000000109 \times NumBits^2) \quad (16)$$

$$DRAM\_ISOutputBuffPower = -7.98 + (0.00484 \times NumBits) + (-0.000000595 \times NumBits^2) \quad (17)$$

VI. Results

Section VI-A details the experimental setup adopted to obtain the results. Next, we evaluate the MAC-based and analytic flows using the physical synthesis flow as reference, in Section VI-B and Section VI-C, respectively. Section VI-D compares the analytical mode to related works.

A. Experimental Setup

We adopt the CNN illustrated in Figure 4 as a case study, with three convolutional layers and one fully-connected layer. TensorFlow executes the fully-connected layer, not accelerated in hardware. The number of filters per layer is 16, 32, and 64. The CNN implemented in the TensorFlow uses the Cifar10 dataset with a $32 \times 32 \times 3$ (RGB) IFMAP. After training, the obtained accuracy was 67%. The obtained accuracy with the quantization using 16-bit words at the inputs was 66.98%, a value 0.02% smaller than the one obtained in TensorFlow with float point values.

Cadence Genus and Innovus tools were used for logic and physical synthesis, with 28nm technology and a frequency of 500MHz. The logic synthesis uses clock-gating to reduce the accelerator energy consumption. The power dissipation uses the VCD file generated after a post-physical synthesis simulation and the Cadence Voltus tool.

The netlist simulation inputs are the Cifar10 CNN layers (Figure 4) extracted from TensorFlow, according to the flow presented in Figure 2). Layer 0 uses a $32 \times 32 \times 3$ IFMAP (RGB image from CIFAR10 dataset), 16 3×3 filters, stride 2, generating a $15 \times 15 \times 16$ output. Layer 1 uses a $15 \times 15 \times 16$ IFMAP, 32 3×3 filters, stride 2, generating a $7 \times 7 \times 32$ output. Layer 2 uses a $7 \times 7 \times 32$ IFMAP, 64 3×3 filters, stride 2, generating a $3 \times 3 \times 64$ output. Thus, power values come from a real dataset and not synthetic values. The total energy is computed by multiplying the average power by the number of clock cycles required to execute a complete convolution.

The Cacti-IO tool [53] models the external memories. For a 28nm 64KB SRAM, Cacti-IO reports 0.01356nJ for reading and 0.01351nJ for writing. For a 64kB DRAM, 0.1633nJ for reading and 0.1662nJ for writing.

TABLE II

MAC-Based and Physical Synthesis Flows Area Results (μm^2) for Layer 1. Maximum Error: 39.90% in Layer 2 and Minimum Error: 29.34% in Layer 0

dataflow	MAC-based flow	Physical Synthesis flow	|error (%)|
WS	9,468.41	15,037.57	37.03
buffered WS	9,468.41	13,777.02	31.27
IS	9,468.41	15,702.45	39.70
buffered IS	9,468.41	14,156.94	33.12
OS	9,468.41	15,336.23	38.26

TABLE III

MAC-Based and Physical Synthesis Flows Power Results (mW) for Layer 1. Maximum Error: 44.58% in Layer 0 and Minimum Error: 12.62% in Layer 1

dataflow	MAC-based flow	Physical Synthesis flow	|error (%)|
WS	1.16	0.87	33.33
buffered WS	1.16	0.92	26.09
IS	1.16	1.92	39.58
buffered IS	1.16	1.90	38.95
OS	1.16	1.03	12.62

B. MAC-Based DSE Flow Results

This Section evaluates the MAC-based DSE flow, using the physical synthesis flow as a reference. The goal is to demonstrate that executing power and area estimation using the required amount of MACs (Section IV-B) does not produce accurate results.

Table II presents area results for the three dataflows and their versions with output buffers for the CIFAR10 CNN layer 1. The MAC-based flow area is the same, regardless of the dataflow type, because this flow only considers the arithmetic core plus input and output registers. The area of the synthesized accelerators does not include the output buffers in such a way to fairly compare both approaches. The difference observed in the non-buffered and buffered versions is due to the presence of control logic to access the OFMAP memory, not present in the buffered versions, translating in a larger area.

As expected, the MAC-based flow underestimates the accelerator area because it does not consider the control logic (FSMs), internal registers, internal buffers, and logic to interconnect components. The area estimation error varies from 29.3% to 39.9%, with an average error of 35.9%.

Table III presents power results using the same setup of the previous table. The MAC-based flow power is the same because this flow only considers the arithmetic core.

The MAC-based flow also presents significant errors in the power estimation. The main reasons for the differences observed in the power estimation are the switching activity and the presence of input buffers. The MAC-based flow uses an average of the switching activity of MACs and registers, fixing this value for the estimation (in this example, 1.16 mW). The physical synthesis uses the switching activity of the whole circuit with actual data. The actual switching activity may be smaller than the average due to the presence of IFMAP and weight values near zero. This fact justifies optimization

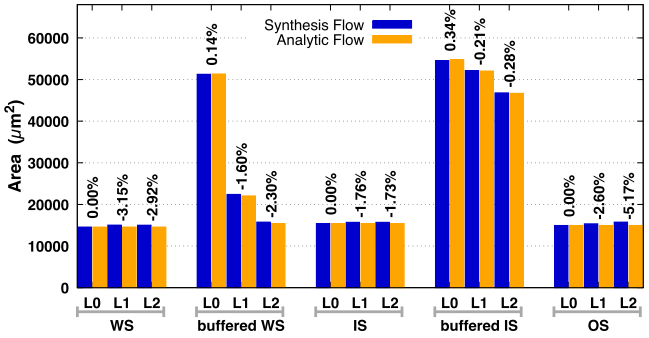


Fig. 5. Area estimation using the analytic DSE flow, for the 3 Cifar10 layers (L0, L1, L2) (percentages represent the difference between flows).

techniques such as pruning [54]. Power is overestimated for WS and OS, and underestimated for IS due to the presence of input buffers. The absolute power estimation error varies from 12.62% to 44.58%, with an average absolute error of 30.1% for layer 1.

We demonstrate from the above results that estimating area and power using the number of arithmetic operators produces results that are far from the actual hardware. Thus, we claim that methods such as the analytical flow evaluated below produce reliable PPA estimates for CNN hardware accelerators.

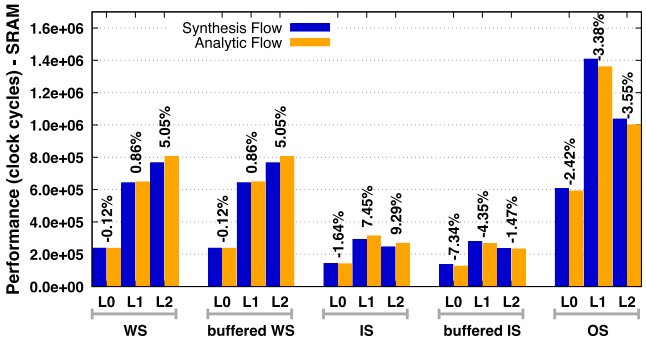
C. Analytic DSE Flow Results

This Section evaluates the proposed analytic DSE flow. The reference for the analytic DSE flow is the physical synthesis flow for the Cifar10 CNN layer 0.

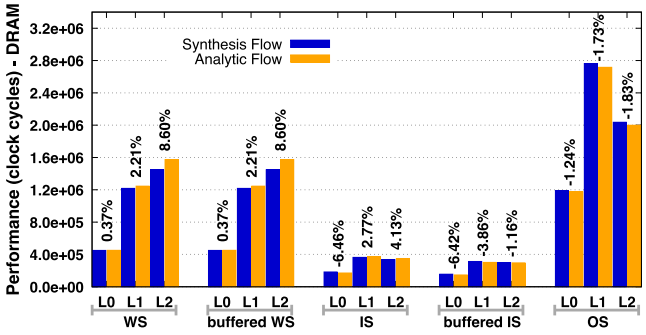
*1) **Area Estimation**:* Figure 5 presents results for area estimation. The WS, IS, and OS areas are the same in layer L0 (error=0) since the synthesis of this layer is the reference for the other layers. Due to the interpolation approach, dataflows with output buffers presented a small error in layer L0 (below 1%). The area error estimation for layers L1 and L2 stays below 4% for layers L1 and L2, excepting OS L2, with an error of 5.17%. The reason for explaining the error is the increased number of output channels compared to L0, which affects the control logic and counters. The OS dataflow does not have buffers, requiring a more complex circuitry to manage memory accesses.

It is worth highlighting that the IS implementations require an input buffer to store weights and bias values, which increase the accelerator area. The area for these buffers is not included in the area results since this buffer acts as a cache memory, requiring an external memory. We adopted this approach because memory compilers are needed to generate these buffers. The use of memory compilers is considered a future work. The IS for this CNN needs 7,168, 74,240, and 295,936 bits for layers L0, L1, and L2, respectively. Thus, in terms of total area, the IS dataflow is larger than the others, requiring further development to reduce this area, as splitting the weights into small samples to minimize the output buffer size and not read and store all weight values in internal buffers.

The overall area estimation error stays below 6%, with an average error of **1.85**% with a standard deviation of 1.51% (minimal error: 0.14%, maximum error: 5.17%).



(a) SRAM memory, with access latency of 2 clock cycles.



(b) DRAM memory, with access latency of 5 clock cycles.

Fig. 6. Performance estimation, in clock cycles, using the analytic DSE flow, for the 3 Cifar10 layers (L0, L1, L2), for SRAM and DRAM memories (percentages represent the difference between flows).

*2) **Performance Estimation**:* Figure 6 presents results for performance estimation, using Equations from Section V-A. The performance results consider different memory latencies, two clock cycles for SRAM and five clock cycles for DRAM. The main source of error is due to synchronization states not included in the Equations. These synchronization states occur mainly at the end of a row, where buffers must be flushed to start a new one. The largest errors occur in layer two due to the larger number of output channels.

The overall performance estimation error stays below 10%, with an average error of **3.50**% with a standard deviation of 2.78% (minimal error: 0.12%, maximum error: 9.29%).

*3) **Memory Accesses Estimation**:* Figure 7 presents results for memory accesses. Estimating the number of memory accesses is mandatory to estimate the total energy consumed by the accelerator.

The analytical model correctly captures the number of memory accesses, except for IS and buffered IS readings, with the worst-case occuring in layer 0 – 9.38%. The reason is similar to the error observed in the performance estimation, where there are synchronization states, mainly in the exchange of rows. At the end of a row, there are invalid reads to avoid increasing the complexity of the FSM. Thus, the memory is active for some clock cycles, inducing these invalid reads. If the memory latency is small (2 cycles for SRAM), more invalid reads may occur, while this effect is masked for higher latencies (5 cycles for DRAM).

TABLE IV

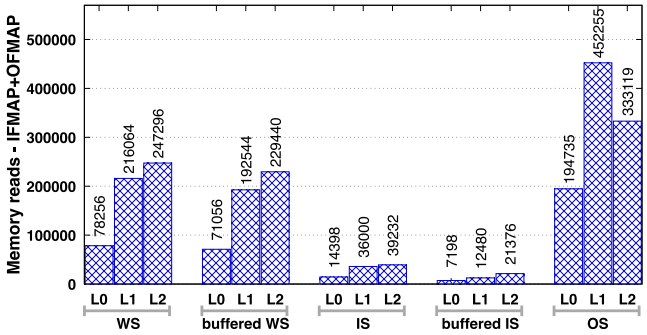
CIFAR10 CNN POWER ANALYTIC RESULTS (ACCELERATOR CORE) - *mW*

dataflow	ws			ws buf			is			is buf			os		
results	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$
SRAM - L0	0.95	0.95	**0.00**	2.32	2.32	0.05	2.04	2.04	**0.00**	4.04	4.06	0.34	1.04	1.04	**0.00**
SRAM - L1	0.87	0.95	9.39	1.25	1.31	5.34	1.92	2.04	6.25	3.73	3.93	5.47	1.02	1.04	1.47
SRAM - L2	0.88	0.95	7.78	1.03	1.11	7.70	1.73	2.04	17.99	3.22	3.53	9.56	1.03	1.04	0.68
DRAM - L0	0.74	0.74	**0.00**	1.91	2.13	0.07	1.79	1.79	**0.00**	3.84	3.96	0.08	0.88	0.88	**0.00**
DRAM - L1	0.71	0.74	4.32	1.06	1.13	6.26	1.70	1.79	5.41	3.54	3.84	5.48	0.87	0.88	0.87
DRAM - L2	0.71	0.74	4.23	0.87	0.93	6.79	1.45	1.79	23.55	2.87	3.44	14.46	0.88	0.88	0.47

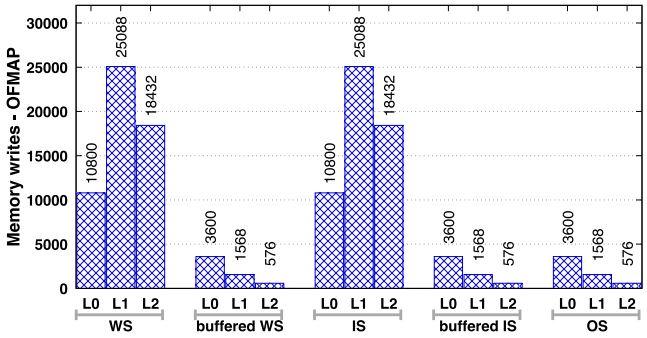
TABLE V

CIFAR10 CNN ENERGY ANALYTIC RESULTS (ACCELERATOR CORE AND EXTERNAL MEMORIES) - *nJ*

dataflow	ws			ws buf			is			is buf			os		
results	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$	synt. flow	analytic	$\|error\%\|$
SRAM - L0	1,207	1,207	0.00	1,013	1,013	0.00	342	332	2.69	147	138	6.26	2,690	2,690	0.02
SRAM - L1	3,270	3,270	0.00	2,633	2,633	0.00	828	817	1.28	191	181	5.55	6,155	6,155	0.01
SRAM - L2	3,603	3,603	0.00	3,120	3,120	0.00	782	778	0.49	298	295	1.29	4,526	4,524	0.04
DRAM - L0	14,569	14,577	0.05	12,196	12,204	0.07	4,037	4,037	0.00	1,665	1,664	0.00	32,401	32,398	0.01
DRAM - L1	39,376	39,460	0.21	31,624	31,708	0.26	9,923	9,924	0.00	2,172	2,172	0.00	74,120	74,115	0.01
DRAM - L2	43,119	43,454	0.78	37,234	37,569	0.90	9,426	9,426	0.00	3,541	3,541	0.00	54,492	54,481	0.02



(a) Number or memory readings.



(b) Number or memory writings.

Fig. 7. Estimated number of memory accesses.

*4) **Accelerator Power Estimation**:* Table IV shows the results for power estimation. Note that this table only considers the accelerator core. Similar to the area estimation, the L0 is the reference. WS, IS and OS present an absolute error equal to 0% (bold values on Table IV), while the buffered dataflows presented an error due to the interpolation approach.

TABLE VI

SUMMARY OF THE ANALYTIC APPROACH RESULTS

	Avg. Error (%)	Std. Dev. (%)
Area	1.85	1.51
Performance	3.50	2.78
IFMAP Read	1.22	2.73
OFMAP Read/Write	0.00	0.00
Acc. Power (core)	7.00	6.21
Total Energy	0.66	4.34

The power estimation has a larger error than the area and performance estimation. Two reasons explain this mismatch:

- The power reference is layer L0, with its switching activity. The switching activity of other layers is different, affecting the power estimation. The switching activity is a function of the input data, not being possible to capture it in the analytic model.
- The buffered dataflows has an error induced by the interpolation method.

The overall power estimation error presents an average error of **7.00**% with a standard deviation of 6.21% (minimal error: 0.05%, maximum error: 23.55%).

*5) **Total Energy Estimation**:* Table V presents results for the energy estimation, considering the accelerators and the memory accesses. Memory accesses are responsible for most of the consumed energy. According to [10], the memory energy can spend 200 times more energy than the accelerator array. As observed, there is a mismatch higher than 1% only in the IS dataflows due to the errors in the memory readings estimations (Section VI-C.3). The errors in the IS dataflows occur due the following reasons:

- The IS dataflow presents a small energy error estimation because the number of OFMAP accesses is higher

TABLE VII
ANALYTIC AND STATE-OF-THE-ART RESULT ERRORS COMPARISON

	Aladdin [43]			MLPAT [30]			Accelergy [25]			STONNE [24]			Our Proposal		
|error %|	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg
area	4.30	10.60	6.60	—	—	5.00	—	—	—	—	—	—	0.14	5.17	1.85
performance	0.20	2.60	0.90	—	—	—	—	—	—	11.00	19.00	15.00	0.12	9.29	3.50
accel. power	2.30	8.30	4.90	—	—	10.00	—	—	—	—	—	—	0.05	23.55	7.00
accel. energy	—	—	—	—	—	—	—	—	5.00	—	—	—	0.12	28.65	8.11

than the IFMAP readings (where the estimation presents errors), resulting in a small energy estimation error, below 3%.
- The bufferd IS dataflow makes more IFMAP readings (with an estimation error equal to 9.38%) than OFMAP writes. The result is a higher energy estimation error.

The overall performance energy error stays below 7%, with an average error of **0.66**% with a standard deviation of 1.54% (minimal error: 0%, maximum error: 6.26%).

*6) **Analytic Model Summary**:* The proposed analytic flow enabled an accurate PPA estimation. Table VI summarizes the results. The main source of error is in estimating the number of IFMAP readings for the dataflow IS, impacting the average power estimation. Also, improvements may be done in the interpolation approach to estimate the area and consumption of output buffers.

Besides the PPA accuracy, the method is *fast*. Below we compare the computing time of each flow.

- Physical synthesis flow. The physical synthesis of one accelerator can take up to 45 minutes (Intel i9-7940X@3.10GHz, 28 cores, 64 GB memory). The DSE of a given CNN configuration takes several hours, considering all layers and channels.
- Analytical flow. It is necessary to synthesize the first convolutional layer of each dataflow, with the remaining layers estimated from the first layer results. The obtained PPA data is used to extract the model. The DSE of a given CNN configuration took 0.025 seconds for the CNN evaluated in this Section.
- MAC-based flow. It also requires a synthesis step for the MAC and registers. The area and power related to the synthesis step are the values used to estimate the accelerator cost.

For comparison purposes, Shao *et al.* [43] mention that their simulator, Aladdin, takes 7 minutes to execute a full DSE of a given set of accelerators against 52 hours for the synthesis flow.

D. Analytic Model Compared to the State-of-the-Art

Table VII compares the analytic model results with results available in the literature. Power and energy consider only the accelerator, not the memories. The Table presents for each work the min/max/average error, when it is available.

Few works in the literature present a comprehensive estimation as the method proposed in this paper. MLPAT and Accerlergy present a limited evaluation of area and energy. Stone evaluates only performance with higher errors compared to our method. Aladdin presents the most complete evaluation compared to the other methods. However, the proposed method has an area evaluation more accurate than Aladdin, while the evaluation of the other metrics has errors of the same order.

As a conclusion, the proposed flow presents a more comprehensive evaluation of more metrics with lower errors, and can provide a large set of estimates for each dataflow.

VII. CONCLUSION

This work presented a fast, comprehensive, and accurate design space exploration analytic method for CNN hardware accelerators. The method integrates front-end frameworks (such as TensorFlow) to a hardware back-end design flow. Despite the coupling of the DSE framework to the accelerator architecture, the flow presented in this work is a guideline for executing DSE for other dataflows. We showed that methods based on basic components, such as MACs, are not enough to have accurate results and present errors between 12.51% and 44.68% for area and power. The average error comparing the analytical model to the data obtained from the physical synthesis is smaller than 7%. Compared to the literature, the proposed method shows a more accurate area evaluation, while the assessment of power and performance have errors of the same order.

Accelerators source code, synthesis and DSE scripts are available at the following GitHub repository:

https://github.com/leorezende93/acc_dse_env

Future work covers accelerator and system levels. At the accelerator level, we plan to: (i) optimize the accuracy of the PPA results, as the performance of the IS dataflow estimation; (ii) add other dataflows, such as NLR (No Local Reuse), RS (Row Stationary), and FG (Fine-Grained), and implement in hardware other CNN functions, such as max and average pooling; (iii) implement larger arrays as 16×16, and integrate the DSE flow with the Imagenet dataset to allow simulation of the accelerators using more complex CNNs; (iv) use both SRAM and DRAM, implementing a memory hierarchy scheme. At the system level, we plan to combine the DSE method with system simulators to perform DSE regarding an entire system composed of CPUs, DMA, and CNN accelerators.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.
[2] Facebook. (2022). *Facebook Horizon*. [Online]. Available: https://www.oculus.com/horizon-worlds/
[3] Google. (2022). *Google Assistant, Your Own Personal Google*. [Online]. Available: https://assistant.google.com

[4] ServiceNow. (2022). *Enterprise Chatbot—Virtual Agent*. [Online]. Available: https://assistant.google.com/

[5] Tesla. (2022). *Autopilot*. [Online]. Available: https://www.tesla.com

[6] S. S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. London, U.K.: Pearson, 2009.

[7] (2022). *Caffe*. [Online]. Available: https://caffe.berkeleyvision.org/

[8] (2022). *PyTorch*. [Online]. Available: https://pytorch.org/

[9] (2022). *TensorFlow*. [Online]. Available: https://www.tensorflow.org/

[10] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "EyeRiss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2016.

[11] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. Interspeech*, Sep. 2015, pp. 1488–1492.

[12] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Commun. ACM*, vol. 63, no. 7, pp. 48–57, Jun. 2020, doi: 10.1145/3361682.

[13] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An architecture for ultralow power binary-weight CNN acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 48–60, Jan. 2018.

[14] S. Shivapakash, H. Jain, O. Hellwich, and F. Gerfers, "A power efficient multi-bit accelerator for memory prohibitive deep neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[15] S.-F. Hsiao, K.-C. Chen, C.-C. Lin, H.-J. Chang, and B.-C. Tsai, "Design of a sparsity-aware reconfigurable deep learning accelerator supporting various types of operations," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 376–387, Sep. 2020.

[16] F. Spagnolo, S. Perri, F. Frustaci, and P. Corsonello, "Reconfigurable convolution architecture for heterogeneous systems-on-chip," in *Proc. 9th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2020, pp. 1–5.

[17] S.-F. Hsiao and H.-J. Chang, "Sparsity-aware deep learning accelerator design supporting CNN and LSTM operations," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–4.

[18] TESLA. (2019). *Autopilot and Full Self-Driving Capability*. [Online]. Available: https://analyticsindiamag.com/under-the-hood-of-teslas-ai-chip-that-takes-the-driverless-battle-to-nvidias-home-turf/

[19] Apple. (2022). *iPhone 11*. [Online]. Available: https://www.apple.com/iphone-11/

[20] S. Das, A. Roy, K. K. Chandrasekharan, A. Deshwal, and S. Lee, "A systolic dataflow based accelerator for CNNs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[21] C. Heidorn, F. Hannig, and J. Teich, "Design space exploration for layer-parallel execution of convolutional neural networks on CGRAs," in *Proc. 23th Int. Workshop Software Compilers Embedded Syst. (SCOPES)*, 2020, pp. 26–31.

[22] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, "DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 1593–1597.

[23] A. Parashar *et al.*, "Timeloop: A systematic approach to DNN accelerator evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2019, pp. 304–315.

[24] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STONNE: A detailed architectural simulator for flexible neural network accelerators," *Comput. Res. Repository*, vol. 2006, no. 1, pp. 1–8, 2020.

[25] Y. Nellie Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2019, pp. 1–8.

[26] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni, "ESP4ML: Platform-based design of systems-on-chip for embedded machine learning," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1049–1054.

[27] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Enabling automated FPGA accelerator optimization using graph neural networks," *Comput. Res. Repository*, vol. abs/2111.08848, pp. 1–12, Jun. 2021.

[28] G. Datta and P. A. Beerel, "Can deep neural networks be converted to ultra low-latency spiking neural networks?" in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 718–723.

[29] P. Panda, S. A. Aketi, and K. Roy, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," *Frontiers Neurosci.*, vol. 14, pp. 1–18, Aug. 2020.

[30] T. Tang and Y. Xie, "MlPat: A power area timing modeling framework for machine learning accelerators," in *Proc. IEEE Int. Workshop Domain Specific Syst. Archit. (DOSSA)*, Aug. 2018, pp. 1–3.

[31] H. Kwon, M. Pellauer, and T. Krishna, "MAESTRO: An open-source infrastructure for modeling dataflows within deep learning accelerators," *Comput. Res. Repository*, vol. abs/1805.02566, no. 1, pp. 1–5, 2018.

[32] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, May 2019, pp. 754–768.

[33] NVIDIA. (2022). *NVDLA*. [Online]. Available: http://nvdla.org/

[34] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. ACM/IEEE Design Automat. Conf. (DAC)*, Mar. 2019, pp. 1–6.

[35] X. Zhang, H. Ye, and D. Chen, "Being-ahead: Benchmarking and exploring accelerators for hardware-efficient AI deployment," *Comput. Res. Repository*, vol. abs/2104.02251, no. 1, pp. 1–12, Jun. 2021.

[36] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 769–774.

[37] K. Asanovic *et al.*, "The rocket chip generator," Univ. California, Los Angeles, CA, USA, Tech. Rep. UCB/EECS-2016-17, 2016. [Online]. Available: https://aspire.eecs.berkeley.edu/wp/wp-content/uploads/2016/04/Tech-Report-The-Rocket-Chip-Generator-Beamer.pdf

[38] X. Yang *et al.*, "Interstellar: Using halide's scheduling language to analyze DNN accelerators," in *Proc. ACM Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, 2020, pp. 369–383.

[39] S. D. Manasi and S. S. Sapatnekar, "DeepOpt: Optimized scheduling of CNN workloads for ASIC-based systolic deep learning accelerators," in *Proc. ACM/IEEE Asia South Pacific Design Automat. Conf. (ASPDAC)*, May 2021, pp. 235–241.

[40] A. Karbachevsky *et al.*, "Early-stage neural network hardware performance analysis," *MDPI Sustainability*, vol. 13, no. 2, p. 717, 2021.

[41] C. Baskin *et al.*, "UNIQ: Uniform noise injection for non-uniform quantization of neural networks," *ACM Trans. Comput. Syst.*, vol. 37, nos. 1–4, pp. 1–15, 2021.

[42] M. Ferianc *et al.*, "Improving performance estimation for design space exploration for convolutional neural network accelerators," *MDPI Electron.*, vol. 10, no. 4, pp. 1–14, 2021.

[43] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proc. ACM Int. Symp. Comput. Archit. (ISCA)*, 2014, pp. 97–108.

[44] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-SIM: Systolic CNN accelerator," *Comput. Res. Repository*, vol. abs/1811.02883, no. 1, pp. 1–11, 2018.

[45] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Aug. 2020, pp. 58–68.

[46] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM Special Interest Group Program. Lang. Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[47] S. Kim *et al.*, "Transaction-level model simulator for communication-limited accelerators," *Comput. Res. Repository*, vol. abs/2007.14897, no. 1, pp. 1–11, 2020.

[48] S. Cao, W. Deng, Z. Bao, C. Xue, S. Xu, and S. Zhang, "SimuNN: A pre-RTL inference, simulation and evaluation framework for neural networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 2, pp. 217–230, 2020.

[49] Keras. (2022). *PReLU layer*. [Online]. Available: https://keras.io/api/layers/activations/

[50] L. R. Juracy, M. T. Moreira, A. M. Morais, A. Hampel, and F. G. Moraes, "A high-level modeling framework for estimating hardware metrics of CNN accelerators," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4783–4795, 2021.

[51] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, no. 1, pp. 1–26, 2021.

[52] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Comput. Res. Repository*, vol. abs/1409.1556, no. 1, pp. 1–14, 2014.

[53] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "Cacti-IO: Cacti with off-chip power-area-timing models," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1254–1267, 2014.

[54] N. J. Kim and H. Kim, "FP-AGL: Filter pruning with adaptive gradient learning for accelerating deep convolutional neural networks," *IEEE Trans. Multimedia*, early access, Jul. 1, 2022, doi: 10.1109/TMM.2022.3189496.

Leonardo Rezende Juracy received the bachelor's degree in computer engineering and the M.Sc. degree in computer science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree. His research interests include design for testability, fault-tolerant designs, asynchronous designs, resilient designs, and machine learning hardware accelerators.

Alexandre de Morais Amory received the Ph.D. degree in computer science from UFRGS, Brazil, in 2007. His professional experience include the Lead Verification Engineer at CEITEC Design House from 2007 to 2009, a Post-Doctoral Fellow at PUCRS from 2009 to 2012; and a Professor at PUCRS from 2012 to 2020. He is currently a Research Fellow at Scuola Superiore Sant'anna, Italy. His research interests include design, test, fault-tolerance, and safety-critical systems.

Fernando Gehm Moraes (Senior Member, IEEE) received the bachelor's degree in electrical engineering and the M.Sc. degree from UFRGS, Brazil, in 1987 and 1990, respectively, and the Ph.D. degree from the Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier, France, in 1994. Since 2002, he has been a Full Professor at PUCRS University. He has authored and coauthored 48 peer-refereed journal articles in the field of VLSI design. His primary research interests include microelectronics, security, MPSoCs, NoCs, and hardware accelerators.