NNASIM: An Efficient Event-Driven Simulator for DNN Accelerators with Accurate Timing and Area Models

Xiaoling Yi[1], Jiangnan Yu[1], Zheng Wu[1], Xiankui Xiong[3,4], Dong Xu[3,4], Chixiao Chen[1,2], Jun Tao[1] and Fan Yang[1*]
[1]*State Key Lab of ASIC & System, School of Microelectronics, Fudan University, China*
[2]*Frontier Institute of Chip and Systems, Fudan University, China*
[3]*ZTE Corporation, China*
[4]*State Key Laboratory of Mobile Network and Mobile Multimedia Technology, China*

2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937422

***Abstract*—In this paper, we propose NNASIM, an efficient timing and area accurate event-driven simulator for custom DNN accelerators. NNASIM is a highly-modular and highly parameterized modeling framework. We build accurate timing and area models for common accelerator modules like GEMM, ALU array, and crossbar using ASIC synthesis flows. These models are fed into the event-driven simulator for fast simulation. NNASIM is integrated with a RISC-V simulator. This approach guarantees the functional correctness of the accelerator simulation at the instruction level. The experimental results show that our model evaluates the performance and area of DNN accelerators with less than 0.76% and 2.83% error, respectively, compared to RTL implementations. NNASIM allows designers to model the performance and area of the accelerator at a high level, and thus enables the systematic microarchitecture design space exploration of the custom accelerators.**

***Index Terms*—Event-Driven Simulation, Neural Network Accelerator, Timing and Area Models**

I. Introduction

With scaling benefits ending, more and more designers are now working on Domain-Specific Architectures (DSAs). Especially, because of the recent phenomenal popularity of Artificial Intelligence, the dedicated hardware accelerators with both performance enhancement and energy efficiency for Deep Neural Networks (DNNs) have received tremendous interest [1] [2] [3] [4] both in academia and industry. Among previously proposed custom DNN accelerator architectures, 2D spatial array architecture is a prominent choice [1] [2]. Besides the systolic array [1], the other option is the parallel vector-based execution array used in such as NVDLA [5].

Meanwhile, accelerators tend to be coupled with a CPU in heterogeneous System-on-Chip (SoC) designs [6] [7]. RISC-V ecosystem allows developers to easily build such a system for its high support for DSAs [8] [9] [10]. RISC-V based DNN accelerators and software stack have gained great attention in recent years [11] [12] [13].

Traditionally, architectural simulators have been wildly applied in the computer architecture research and design process [14] [15] [16], since they permit rapid evaluation of the architecture enhancements, leveraging high-level simulation frameworks. In contrast, the simulators for DNN accelerators have not been well developed. Current DNN accelerator related research usually relies on traditional ASIC synthesis flow, which requires significant expertise and takes hours even days to evaluate the performance and area of a single design. Only a few high-level DNN accelerator simulators are reported, for example, STONNE [17] and SMAUG [18]. The simulators such as SCALE-Sim [19], Timeloop [20] and NNest [21] act as analytical models for evaluating dataflow or mapping strategy, but they can not ensure the functional correctness of simulation. To the best of our knowledge, there is still no detailed open-source architectural simulator for the vector-based spatial array accelerators with accurate timing and area models, functional correctness, and RISC-V integration.

*Corresponding author: {yangfan}@fudan.edu.cn

TABLE I
Comparison of high-level simulators for DNN architectures

	Simulation Support	Array Type	RISC-V Integration
Timeloop	✗	systolic/vector-based	✗
NNest	✗	vector-based	✗
SMAUG	✓	vector-based	✗
STONNE	✓	flexible	✗
SCALE-Sim	✗	systolic-based	✗
NNASIM	✓	vector-based	✓

To address this issue, in this paper, we present NNASIM (**N**eural **N**etwork **A**ccelerator **SIM**ulator), an efficient, timing and area accurate, highly-modular, and highly parameterized simulation framework for DNN accelerators. Technically, we use an event-driven simulation mechanism to model the custom DNN accelerator's behavior. We build accurate timing and area models for common hardware modules like vector-based GEMM, ALU array, and crossbar in the accelerator using ASIC synthesis flows. These models are fed into the event-driven simulator for fast simulation. Moreover, NNASIM provides functional correctness at the instruction level and a full software stack as an end-to-end evaluation method for hardware-software co-design, leveraging the compatibility with the RISC-V ecosystem. Table I gives a qualitative comparison of our proposed simulator NNASIM with the contemporary publicly available high-level simulators for DNN accelerators.

The remainder of this paper is developed as follows. In section 2, we introduce the proposed NNASIM with detailed modeling methodology described. The NNASIM validation results are presented in section 3. Then in section 4, we demonstrate an execution time decomposition analysis and a fast accelerator microarchitecture design space exploration enabled by NNASIM, which identify the design insights. Finally, we conclude this paper in section 5.

II. Modeling methodology

In this section, we describe the detailed NNASIM modeling methodology, including discrete event-driven simulation framework, timing and area models, RISC-V integration, software stack, and the user interface.

A. Discrete Event-Driven Simulation Framework

The foundation of the NNASIM infrastructure is the discrete event-driven simulation methodology which describes the hardware behavior of the DNN accelerator. The discrete event-driven mechanism models the dynamic and time-dependent real-world processes as discrete events. Between these events, the system state is approximated as fixed. Without considering all the detailed execution processes of the specific DNN workloads, we focus on the performance, area consumption, hardware utilization, and memory bandwidth requirements, etc. of the DNN accelerators. Therefore, such a discrete event simulation approach helps to enhance simulation performance and efficiency.

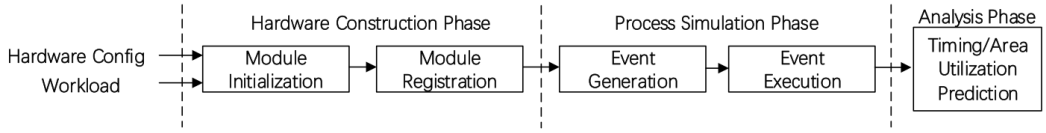


Fig. 1. High-level diagram of the NNASIM framework

1) Discrete Event-Driven Simulation: Figure 1 illustrates the overall structure of NNASIM. In the construction phase, NNASIM takes in the user-defined DNN accelerator configuration file and specific workload. The hardware modules are constructed and registered according to the configuration file. In the simulation phase, the workload is divided into DNN operators, which are composed of custom instructions. The custom instructions are finally mapped to two types of events, either data transformation event, or execution event. One instruction can be converted to one or several events. For instance, one custom GEMM instruction is translated to two source data transformation events, one execution event, and one destination data transformation event. Each event consists of event type, execution time, starting time, and two module pointers. NNASIM has a scheduler to generate the starting time and execution time of each event based on a real-world timing model, hardware resources, and the data dependency. The data transformation event focuses on the read behavior of the first module and the write behavior of the second module. For the execution event, it focuses on the execution behavior of the first module and the second module is null. All the events are pushed into an event priority queue. The comparison element of the priority queue is the event's starting time, so all the events are sorted automatically in a unidirectional timeline. With the events executed one by one, NNASIM simulates the real-world circuit state-changing process and accumulates the accurate cycle consumed simultaneously. NNASIM decides whether to update the current time or not for every event according to its termination time, which is the event's starting time pluses execution time. If the current time is numerically smaller than the current event termination time, then the current time updates to the current event termination time, otherwise it remains the same. This manner simulates the hardware parallelism. The outcome of these two phases is then fed into the analysis phase to get a pre-RTL timing and area evaluation as well as analytical data such as module utilization rate. For supporting full-stack evaluation and hardware-software co-design, such as DNN quantification and pruning, NNASIM does not just analyze but truly executes the events to simulate the real-world process thus ensuring functional correctness.

2) Module Behavior and Characteristic Abstraction: After analyzing several common DNN accelerator architectures, we design several highly parameterized hardware components including vector-based GEMM, ALU Array, DMA, On-chip Buffer, Accumulator, crossbar, etc. in the NNASIM framework. We have configurable connection support for all hardware modules to enhance dataflow and mapping strategy exploration. NNASIM has been developed in C++. Considering all modules have read, write, execution behavior, and some other characteristic in common, we design a base class that abstracts these features. We take advantage of the object-oriented programming principle to derive specific module classes.

B. Timing and Area Models

We now elaborate on the construction method of the NNASIM timing and area models which aim to evaluate the performance and resource requirements of the custom DNN accelerators. To offer flexible parameterization and configurability, we design the hardware module generators at the RTL level using Chisel [22] hardware construction language. We run the ASIC synthesis flow of the Chisel-generated Verilog to get the real timing and area data under a specific technology node. All the possible design points are swept to further support exhaustive microarchitecture design space exploration. The simulation data are directly plugged into the NNASIM to build timing and area libraries.

1) Timing Model: To model the timing of the DNN accelerators accurately, we require accurate timing characterization of different NNASIM components. We characterize the timing for each type of NNASIM functional component (GEMM, ALU, DMA, Controller, etc.) from the commercial EDA tool, i.e., Synopsys Verilog Compiler Simulator (VCS).

In particular, we construct all possible representative microbenchmarks that cover the computation and data movement operations under various conditions. Then we use VCS to simulate the functional components under different parameters with our microbenchmarks and obtain the execution cycles of each functional component by checking the generated waveform files to build our timing library. NNASIM models the performance of specific accelerator architecture with the discrete event-driven simulation framework based on our real-world timing library.

2) Area Model: We construct a detailed area model by synthesizing our functional units in conjunction with a commercial 28nm standard cell library in Synopsys Design Compiler (DC). Similar to the previously described timing model, we synthesize each hardware module separately to get the accurate module area and enhance modularity. We traverse all the possible and valuable parameters to enhance extensibility and configurability. Furthermore, this synthesis flow is fully automated so that it can be easily migrated to new technologies. For unconsidered parameters, we also construct a parameter-area linear interpolation model for each module using the Matlab cftool with our simulation data, thus ensuring that our area model can handle arbitrary module parameters.

We organize the simulation data into a lookup table. When a new accelerator is constructed with all the hardware modules initialized and registered, NNASIM accumulates the area of each module by indexing the lookup table. In the case of miss retrieve of the lookup table data, NNASIM estimates a module area from the linear interpolation model. Furthermore, considering there are some functional units to connect these separated modules in the real hardware, NNASIM uses a prior cubic function to fine-tune the cumulative module area and then finally gets the overall accelerator area. We get the prior function and validate our area model using different hardware configurations, ensuring our model's generality.

3) Integration with Memory: NNASIM has its on-chip memory model. We design a variety of SRAM blocks with different word width and word depth using Memory Compiler. For the large data width required by computational units such as GEMM and ALU Array, we combine small SRAM instances into SRAM arrays. We design complicated data and address selection strategies at the RTL level to ensure that the read/write operation is conducted in one cycle. The SRAM area and timing model are based on a commercial 28nm Memory Compiler that accompanies our standard cell library.

To enable the system-level evaluation, we provide a simple DRAM timing model. In this paper, the term "DRAM timing model" is defined as the memory system timing model outside the accelerator. We conduct a large number of simulations of a realistic Chipyard [9] memory system including L1, L2 Cache, and DRAM. We develop an accurate statistical analysis of the simulation data to build our DRAM timing model.

C. RISC-V Integration

We provide an ideal DRAM behavior model, enabling NNASIM to conduct simulation independently. However, the optimal performance design point at the accelerator level might result in suboptimal performance at the system level [23].

Considering this issue, NNASIM integrates with a RISC-V simulator to conduct an end-to-end and overall evaluation. In particular, we integrate NNASIM with the Spike [24], a RISC-V ISA Simulator, using the Rocket Custom Coprocessor Interface (RoCC) [25]. In this situation, NNASIM acts as a co-processor cooperating with the main processing system. NNASIM implements the RISC-V custom instruction programming interface and loads/stores data through a global memory block. With Spike integrated, NNASIM can execute a full DNN workload in RISC-V binary format to evaluate the overall performance.

D. Software Stack

NNASIM can be programmed through an easy-to-use C/C++ programming interface. To enhance the developers' productivity, we produce a hand-tuned software stack with frequently used DNN operators, such as tiled matrix multiplication, convolution, fully-connected, pooling, im2col, etc. At runtime, according to the hyper-parameters of the DNN layers and the hardware parameters, the NNASIM software stack automatically determines the tiled computing pattern and the maximum amount of data moved into the buffer in each iteration. Our software stack shares the same accelerator configuration file with NNASIM, facilitating rapid software-hardware co-design. Furthermore, we implement an example software library based on the above-mentioned operators, including bare-metal tests, multi-layer perceptions, and several common neural networks, including LeNet5, AlexNet, etc.

E. User Interface

NNASIM takes in the specified accelerator configuration file and DNN workloads. It outputs the modeling results (such as prediction results), performance, area consumption, as well as component utilization. NNASIM enables highly parameterized module instantiation and connection configuration. Users can conveniently describe a custom accelerator architecture and evaluate it.

III. SIMULATOR VALIDATION

A. Validation Methodology

We begin this section with a detailed description of the methodology used to validate NNASIM, as illustrated in Figure 2. In the RTL implementation flow, we develop a real DNN accelerator hardware generator using the Chipyard [9] framework. The NNASIM framework and the RTL implementation flow take in the same configuration and workload. The area evaluation of NNASIM is compared against synthesized results generated by DC using commercial 28nm standard cells and Memory Compiler. The NNASIM timing model is validated against VCS simulation results.

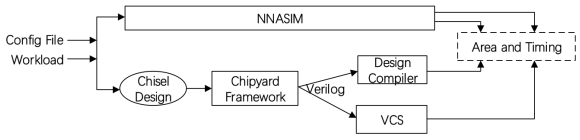


Fig. 2. The NNASIM framework validation flow.

B. Architecture and Workload

We implement a Chisel hardware generator for an in-house accelerator, which can be viewed as a typical contemporary DNN accelerator, to demonstrate NNASIM's modeling capabilities. A system-level overview of the validation accelerator architectural template is illustrated in Figure 3. Leveraging the highly parameterized architectural template, we can easily instantiate a DNN accelerator to validate NNASIM. We use the architectural template with the parameters listed in Table II to validate the NNASIM timing and area models.

For workload, we implement some test programs, including basic bare-metal instruction tests, Mat-Mat multiplication of matrices, and several multi-layer perceptions, as the benchmarks to validate the NNASIM. We integrate the accelerator with a RISC-V host CPU [25] in the Chipyard framework to provide the same high-level programming interface as NNASIM so that the two platforms take in the same test binary programs for validation.

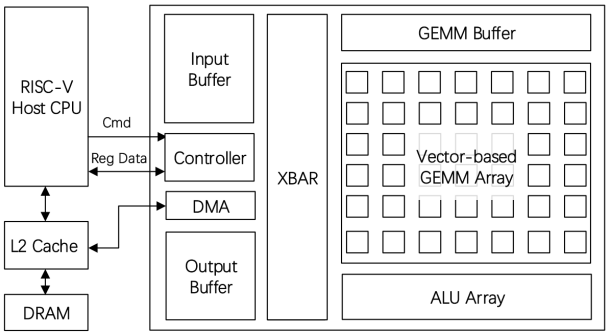


Fig. 3. A system-level overview of the validation accelerator architectural template

TABLE II
VALIDATION HARDWARE CONFIGURATIONS

Config Name	GEMM Array	ALU Unit	On-chip Buffer
Big	16*16	128	512K
Base	8*8	32	128K
Small	4*4	8	128K

C. Validation Results

Validation results show that NNASIM ensures functional correctness and has modest performance and area evaluation relative error within 0.76% and 2.83%. Moreover, NNASIM provides these evaluations over 100× faster compared to ASIC synthesis flow, showing NNASIM's efficiency.

1) Functional Correctness Validation: We compare the execution result of NNASIM and the RTL implementation with equal bare-metal and neural network benchmarks. The experimental results indicate that our simulator achieves equal numerical value with the RTL implementation, demonstrating the functional correctness of the NNASIM.

2) Timing and Area Models Validation: Figure 4 depicts the accelerator performance given by the NNASIM and the RTL implementation across different hardware configurations and benchmarks. The percentage number above each pair of the bar is the performance relative error between the NNASIM evaluation and the RTL implementation of each benchmark. Across all the presented benchmarks, the average relative error is 0.76%. Table III shows the average area relative error is 2.83% across different hardware configurations. These experimental results indicate that NNASIM models the timing and area of the accelerator with high accuracy.

TABLE III
NNASIM AREA MODEL VALIDATION

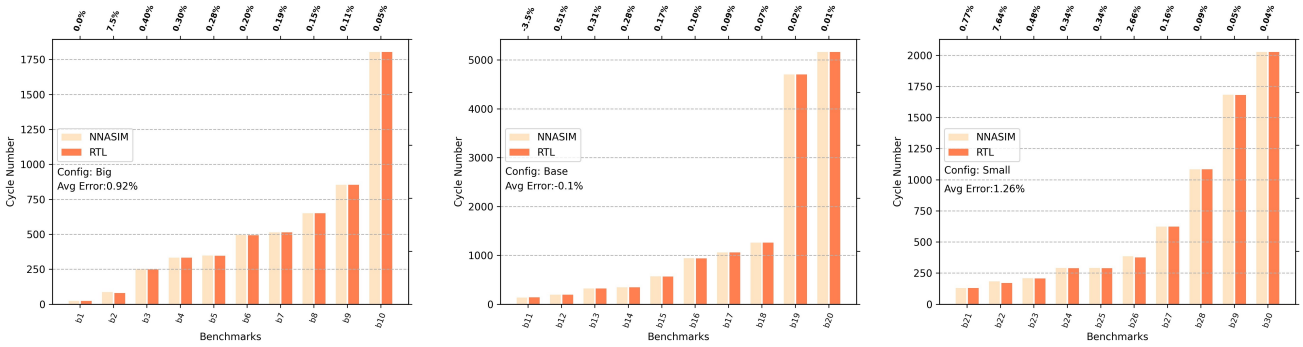
Config Name	NNASIM (um^2)	RTL (um^2)	Relative Error
Big	6994469.87	6924062.84	1.02%
Base	1869196.10	1818318.70	2.80%
Small	1782205.80	1702429.70	4.68%

IV. DESIGN INSIGHTS USING NNASIM

We now present two case study that demonstrates the insights that NNASIM can bring to DL hardware designers. We focus on DNN accelerator holistic performance analysis based on several representative DNN workloads.

A. Execution Time Decomposition

So far, NNASIM has been viewed as a standalone DNN accelerator simulator without considering the outside memory access cycles because obtaining data cycle might not always be an ideal one cycle in real designs with large workloads [26]. However, the actual efficiency of accelerators highly depends on the memory system. To quantify the memory system's effect, we provide a DRAM timing model by extracting the



(a) For Config Big (b) For Config Base (c) For Config Small

Fig. 4. NNASIM timing validation

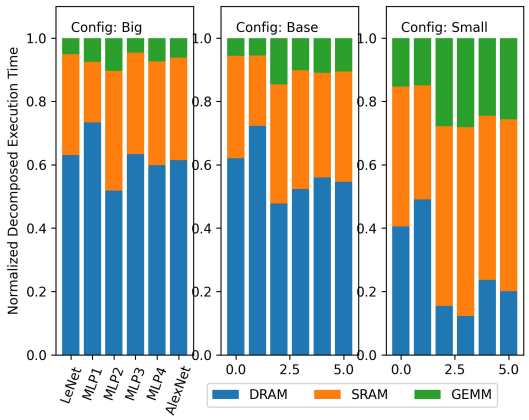


Fig. 5. Execution time decomposition

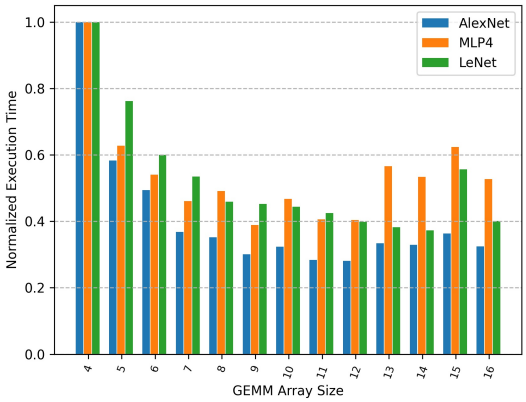


Fig. 6. Execution cycles under different GEMM array size

Chipyard memory system simulation statistical characteristic. Experimental results show the error between our DRAM timing model and Chipyard RTL implement with an average of 6.3% under the above mentioned benchmarks, demonstrating the reliability of our DRAM timing model.

We divide the accelerator's overall execution time into the computing time which is the functional unit execution time, and the memory time which consists of On-chip Buffer data movement time (SRAM time) and DRAM access time (DRAM time). Figure 5 illustrates the decomposition execution time of several DNN workloads under hardware configurations listed in Table II. We find that with Base Config, the memory time occupies 85.4%-94.5% of the overall time, and DRAM access consumes 47.8%-72.3% of the overall time, causing low GEMM utilization of only 5.5%-14.6%. With the Small Config, SRAM time becomes predominant. The GEMM utilization improves to 14.9%-28.1%. The overall low GEMM utilization uncovers the fact that system memory can not supply sufficient operands to keep all the computation units busy. The off-chip memory and the on-chip SRAM bandwidth need to be optimized dedicatedly.

B. Accelerator Design Space Exploration

NNASIM allows designers to conduct comprehensive accelerator microarchitecture design space exploration. For instance, NNASIM can generate 1 billion more microarchitecture design points for a single accelerator architecture combined with different parameters and connection relationships. To illustrate NNASIM's support for accelerator architecture optimization, we use NNASIM to sweep one parameter with other parameters fixed, using the Base Config in Table II. Figure 6 shows several workloads' overall execution time under different GEMM array configurations. We notice that big GEMM is not always the best choice. After carefully checking the execution trace, we find that the reason lays in that interacting large block data may cause an unstable DRAM communication time. Loading/storing a large block of data may consume more time than the computation itself. Figure 6 also points out that for different workloads, the best hardware configuration varies. Thus, the accelerator needs to be subtly optimized for the target DNN applications, demonstrating the significance of hardware-software co-design.

V. Conclusion

In this paper, we propose NNASIM, a fast, timing and area accurate, highly-modular, and highly parameterized simulator, enabling functional correctness and integration with the RISC-V ecosystem. NNASIM adopts discrete event-driven simulation methodology and builds the timing and area models from ASIC synthesis flows. We rigorously validate NNASIM against RTL implement under end-to-end evaluation with diverse workloads. Experimental results show that the average performance and area errors are within 0.76% and 2.83%, respectively. Furthermore, our execution time decomposition and accelerator design space exploration experiment show that NNASIM highlights system bottlenecks and uncovers that how parameters affect the accelerator microarchitecture design trade-offs.

Acknowledgment

This research is supported partly by National Key R&D Program of China 2019YFA0709600, 2019YFA0709602, partly by National Natural Science Foundation of China (NSFC) 61822402, 62090025, 62011530132 and 61929102.

REFERENCES

[1] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[2] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[3] Derek Chiou. The microsoft catapult project. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 124–124. IEEE Computer Society, 2017.

[4] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801. IEEE, 2021.

[5] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. Research on nvidia deep learning accelerator. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 192–195. IEEE, 2018.

[6] Farzad Farshchi, Qijing Huang, and Heechul Yun. Integrating nvidia deep learning accelerator (nvdla) with risc-v soc on firesim. In *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 21–25. IEEE, 2019.

[7] Giuseppe Desoli, Nitin Chawla, Thomas Boesch, Surinder-pal Singh, Elio Guidetti, Fabio De Ambroggi, Tommaso Majo, Paolo Zambotti, Manuj Ayodhyawasi, Harvinder Singh, et al. 14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 238–239. IEEE, 2017.

[8] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovi. The risc-v instruction set manual. volume 1: User-level isa, version 2.0. Technical report, California Univ Berkeley Dept of Electrical Engineering and Computer Sciences, 2014.

[9] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[10] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, et al. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[11] Wenqi Lou, Chao Wang, Lei Gong, and Xuehai Zhou. Rv-cnn: flexible and efficient instruction set for cnns based on risc-v processors. In *International Symposium on Advanced Parallel Processing Technologies*, pages 3–14. Springer, 2019.

[12] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. Pulp-nn: accelerating quantized neural networks on parallel ultra-low-power risc-v processors. *Philosophical Transactions of the Royal Society A*, 378(2164):20190155, 2020.

[13] Pengcheng Xu and Yun Liang. Automatic code generation for rocket chip rocc accelerators.

[14] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[15] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480, 2009.

[16] Todd Austin, Eric Larson, and Dan Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.

[17] Francisco Muñoz-Martínez, José L Abellán, Manuel E Acacio, and Tushar Krishna. Stonne: A detailed architectural simulator for flexible neural network accelerators. *arXiv preprint arXiv:2006.07137*, 2020.

[18] Sam Xi, Yuan Yao, Kshitij Bhardwaj, Paul Whatmough, Gu-Yeon Wei, and David Brooks. Smaug: End-to-end full-stack simulation infrastructure for deep learning workloads. *ACM Transactions on Architecture and Code Optimization (TACO)*, 17(4):1–26, 2020.

[19] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[20] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 304–315. IEEE, 2019.

[21] Liu Ke, Xin He, and Xuan Zhang. Nnest: Early-stage design space exploration tool for neural network inference accelerators. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 1–6, 2018.

[22] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*, pages 1212–1221. IEEE, 2012.

[23] Yakun Sophia Shao, Sam Likun Xi, Vijayalakshmi Srinivasan, Gu-Yeon Wei, and David Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[24] A.Waterman and Y.Lee. Spike-risc-v isa simulator. https://github.com/riscv/riscv-isa-sim. Accessed April 4, 2010.

[25] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[26] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 97–108. IEEE, 2014.