A System for Validating Resistive Neural Network Prototypes

Brian Hoskins, Mitchell Fream, Matthew Daniels, Jonathan Goodwill, Advait Madhavan, Jabez McClelland
National Institute of Standards and Technology
Gaithersburg, Maryland, USA

Osama Yousuf, Gina Adam
George Washington University
Washington, DC, USA

Wen Ma, Tung Hoang, Mark Branstad, Muqing Liu, Rasmus Madsen, Martin Lueker-Boden
Western Digital Research
San Jose, California, USA

ABSTRACT

Building prototypes of heterogeneous hardware systems based on emerging electronic, magnetic, and photonic devices is an important area of research. The novel implementation of these systems for artificial intelligence poses new and unforeseen challenges in mixed signal data acquisition, hyperparameter optimization, and hardware co-processing. Many emerging devices exhibit unpredictable and stochastic behavior as well as poorly repeatable hysteretic effects or performance degradation. Dealing with these device challenges on top of more traditional hardware problems, like quantization errors, timing constraints, and even hardware and software bugs is an enterprise fraught with pitfalls. Equally important to the construction of the physical prototype is the co-development and integration of a design verification framework that can extensibly allow for predictable behavior of not only the entire system but also all of its parts in a modular way, allowing for seamless integration in both simulation and implementation. This work discusses Daffodil-lib, a Python based prototyping framework which, from hardware to software, enables everything from a script-based simulation to a compiled hardware-timed experiment, to everything in between with no syntactical changes for the end user.

CCS CONCEPTS

• **Hardware → Emerging tools and methodologies**; **Memory and dense storage**; **Emerging simulation**.

KEYWORDS

design verification, neural networks, prototyping, hardware

ACM Reference Format:
Brian Hoskins, Mitchell Fream, Matthew Daniels, Jonathan Goodwill, Advait Madhavan, Jabez McClelland, Osama Yousuf, Gina Adam, and Wen Ma, Tung Hoang, Mark Branstad, Muqing Liu, Rasmus Madsen, Martin Lueker-Boden. 2021. A System for Validating Resistive Neural Network Prototypes. In *International Conference on Neuromorphic Systems 2021 (ICONS 2021), July 27–29, 2021, Knoxville, TN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3477145.3477260

ICONS 2021, July 27–29, 2021, Knoxville, TN, USA

ACM ISBN 978-1-4503-8691-3/21/07.
https://doi.org/10.1145/3477145.3477260

1 INTRODUCTION AND BACKGROUND

Increasing numbers of novel neuromorphic prototypes are becoming available. These include traditional silicon technologies that span from implementing novel architectures, such as Loihi, to more conventional deep neural networks [9, 18]. These conventional hardware systems benefit from a long history of design verification, which includes industry practice in classic digital system design, as well as practice in software development [4, 11, 12, 20]. Traditionally, design verification took place within the space of commercial electronic design automation tools, but now a growing number of open source tools – especially those based on Verilator[1], a tool for compiling ("verilating") Verilog into accessible C++ libraries – have facilitated an explosion of alternatives. These alternatives have primarily been used for hardware description and verification using high-level languages [3, 8, 13, 17, 22]. These recent advancements have expanded the utility and interoperability of digital system design with high-level modeling, making it easier to simulate the interaction of a digital system with the world or an analog system.

In parallel to these advancements, interest has grown in the development of analog systems based on both conventional silicon as well as on unconventional devices like resistive memories (ReRAM), phase change memories (PCM), magnetic tunnel junctions (MTJs), and even photonic modulators [1, 5, 10, 14]. In addition to an expanding number of prototypes, there have also been new high-level modeling and simulation tools being developed, but these have only just begun to grow into design verification frameworks [2, 6, 19, 21]. In general, design verification for analog hardware systems is not as mature as it is for digital systems [7], and this is doubly the case for systems that must also incorporate complex dynamics such as weight updates or spike timing dependent plasticity [5].

Consequently, as prototypes develop, it is important to evolve in parallel the tools, methodologies, and software frameworks necessary to ensure that the system functions correctly, accurately predicts a system's performance, and encompasses the total span of hardware/model isomorphisms required for a prototype to be directly translated into an integrated system on a chip. To that end, we introduce Daffodil, a modular end-to-end system capable of simulating and executing experiments on arrays of up to 20,000 resistive devices. The system is composed of an integrated circuit, a mixed-signal daughterboard, a field-programmable gate array (FPGA) development board, and a software framework including a compiled CPU, embedded Linux distribution, FPGA hardware drivers, and Python-based application programming interfaces (APIs).

[1] https://www.veripool.org/verilator/

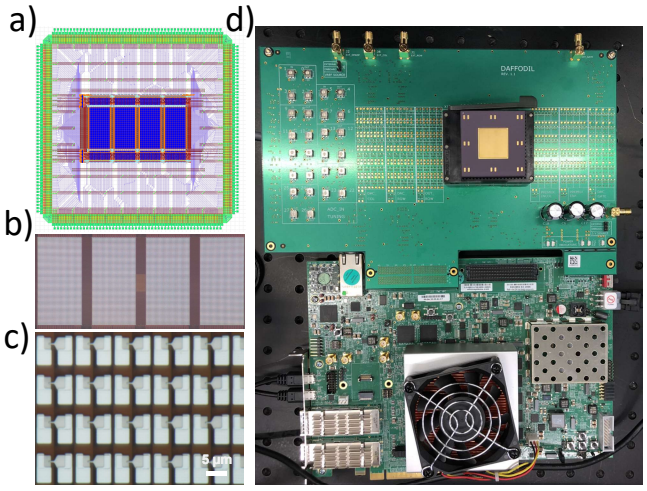

Figure 1: Details of prototyping platform. a) GDS file of the 20,000 device chip. b) 20,000 devices fabricated on the chip. c) Micrograph of a smaller subarray. d) Custom ADC/DAC daughterboard and packaged chip interfacing to an FPGA unit.

The system design leans heavily on conforming to modular interfaces, software-hardware polymorphisms within the APIs, and functionally exact reconstructions of the hardware operations in simulation [12]. The goal of the verification framework is not only to model the system's behavior, but also to model the behavior of programming and algorithmic mistakes so that the simulation predicts even unintended side effects.

2 HARDWARE AND OPERATING SYSTEM

2.1 Integrated Circuit

The integrated circuit, called the Daffodil Chip, is a flexible-interface platform specifically designed for the research and development of two-terminal resistive memory and selector devices. It is designed in a commercial 3.3 V, 180 nm complementary metal oxide semiconductor (CMOS) process and contains via and access points for direct integration of 20,000 resistive devices in a research foundry. The chip is accessed by 403 pads, including three digital CMOS logic configuration pads, 50 gate-access pads, 50 column-access pads, and 200 row-access pads for each side of a double connected row. The remaining pads are for electrical power. Two of the digital pads are used for accessing internal 4:1 multiplexers, which allow for access to one of four internal 50x100 2T-1R arrays. Consequently, the maximum number of devices accessed at a time is 5,000.

2.2 Mixed Signal Daughterboard

The chip interfaces with a mixed-signal daughterboard, which we call the Daffodil Board. The Daffodil Board is designed to access any of the up to 32 unique subarrays, or *kernels*, within the 20,000 array chip. Each kernel, which can be considered as a 25×25 2T-1R crossbar, is accessed with 75 unique digital-to-analog converter (DAC) channels, giving each of the rows, columns, and gate columns a unique bias. For read operations, the row or column DACs can be replaced with tunable transimpedance amplifiers feeding to an array of 25 analog to digital converter (ADC) channels. Negative current biases can be emulated in the transimpedance amplifiers by raising their reference inputs from ground using one of the DAC channels. For diagnostic purposes, the rows, columns, and gates can also be globally or individually connected to ground or an external source through coaxial connectors. Logically, the board's selection amongst the 32 kernels is mediated by five external CMOS logic signals. Individual rows and columns can be activated by any of 50 unique CMOS logic signals. Three sets of three CMOS logic signals configure the DAC, ADC, ground, external configurations, or disable the access configuration multiplexers completely. All the 96 CMOS logic configuration signals, the ADC and DAC serial-peripheral interface (SPI) lines, clock signals, and the board power are sourced from a 400-pin FPGA mezzanine connector (FMC).

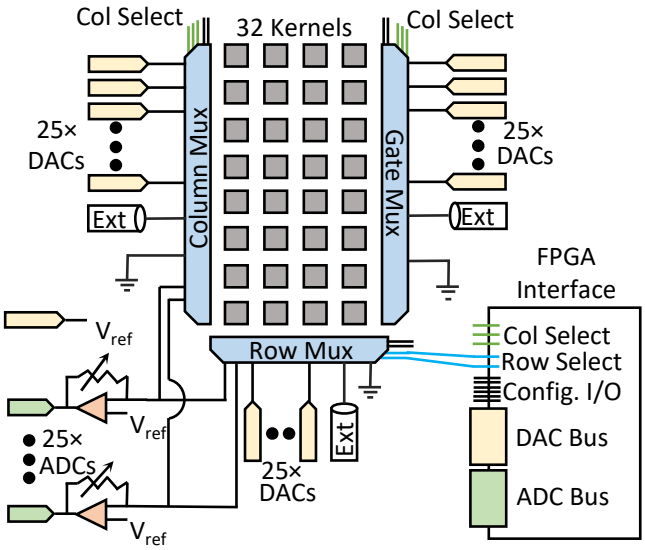

Figure 2: Logical diagram of the prototyping architecture. Five CMOS logic signals access any of 32 ReRAM kernels. Each of the 25 rows, columns, and gates can be independently biased with DACs, accessed externally, or grounded. Transimpedance amplifiers allow for readout of the array.

3 SOFTWARE

3.1 Operating System and Drivers

The Daffodil Board is designed to be plugged into a commercial FPGA development board to act as the online host. The FPGA host operates primarily from a compiled Xilinx Microblaze[2] soft-processor which interfaces to the board drivers and to a superhost Linux PC via Ethernet. The processor operates with a custom embedded Linux distribution, Daffodil Linux, built from the Yocto Project toolset[3]. All of the digital control signals on the Daffodil Board, driven by 1.8V CMOS logic, are accessible as general purpose input/output (GPIO) signals from the Daffodil Linux operating

[2]Certain commercial processes and software are identified in this article to foster understanding, and. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the processes and software identified are necessarily the best available for the purpose.
[3]https://www.yoctoproject.org/

system and can be asserted individually. This allows for the DACs and ADCs to be controlled using off-the-shelf drivers.

In addition to the hardware drivers, the Daffodil Linux distribution contains hooks to call any hardware compiled function on the FPGA. These functions can control any precise, hardware timed operation including pulse generation, read operations, kernel selection, random number generation, random number generation, or algorithmic coprocessing of batch and gradient information.

3.2 Daffodil-lib Base and Simulation Classes

Once the physical structures of the Daffodil Chip and Board are set and all driver definitions are encoded, the Daffodil library (Daffodil-lib) operates as the system's simulation engine, control software, and design verification framework. Written in Python, the library is engineered to describe the board operation in a predictive and a descriptive manner. It is descriptive in the sense that the code written under Daffodil-lib must set the relevant 1.8V logic lines, modeled by integers or Booleans, to select a kernel or specify inputs/outputs—but it is also predictive in the sense that the system must check what the existing logic lines are set to before executing a simulation of the expected behavior. The key action of the predictive behaviors is to set the correct simulated values into the DACs and ADCs as well as the correct voltages and currents in the ReRAM memory.

In the case of the DACs and ADCs, each is modeled with a part class that reproduces the specified transfer functions from the digital to the analog domain, which, in the case of the Daffodil board, is 12-bit digitization. Consequently, any mapping from floating point abstract layers down to the analog layers are automatically rounded and any communication from analog to the abstract neural net layers are also digitized.

The ReRAM memory model interface exposes only input voltages, output currents, and time values as accessible parameters. Consequently, the internals of the model could be anything: a Python model, a C model, or a full SPICE simulation of the system including noise, or even commands to execute operations on a real device. In this section, we discuss both a Python simulation class and a physical operation class. Our implemented ReRAM model and interface does not distinguish between inference and learning operations, and so automatically checks for the possibility of device conductance changes based on the inputted biases. One concrete class descended from the base class is our Python model, named the *simulation class*. As a child of the base class, the simulation class retains the descriptive ADC and DAC models, e.g., it considers the 12-bit. Additionally, the class implements its predictive behavior through a Python wrapped ReRAM model.

The key operation of the simulation is the *event* operation, which maps the correct biases to and from the DACs and ADCs to the ReRAM memory model. Using the input pulse time duration, the simulator predicts the ReRAM behavior, including operationally erroneous behaviors that a user might invoke in higher level code. For example, an inference operation with too-high bias is indistinguishable from a write operation, and improperly set gate biases will not generate any current. Improperly set CMOS logic lines would generate the wrong behavior in the simulation exactly as they would in the experiment. The potential to capture mistakes introduced in the higher level code that would occur in the experiment is the critical element of the predictive/descriptive model that is crucial for the design verification utility of the base class.

3.3 Daffodil-lib Hardware Class

Of the many concrete classes derived from the Daffodil-lib base class, one of them is uniquely privileged in that it controls the hardware directly, rather than running simulations. We call this the *hardware class*. The Daffodil-lib hardware class, like the simulation class, inherits its interface from the base class and keeps the descriptive behaviors. The predictive behaviors, such as the routing of signals, placement of voltages on the DACs/ADCs, or ReRAM response characteristics are now spontaneous reactions of the physical system rather than mathematical models.

The descriptive behaviors, such as setting the CMOS logic signals, programming the DACs/ADCs, and timing the operation of events must each invoke either individual GPIO lines or otherwise call device drivers from the device tree, which may either access individual components or orchestrate a complex series of hardware-timed actions. To avoid confusing syntax, Python's operator overloading features are used to expand the get attribute and set attribute actions of the base class to also simultaneously invoke the associated driver calls. Critically, none of the software interfaces between the hardware class and the base class are changed, allowing the higher level functions to operate with either the Daffodil-lib base or hardware class without modification. In addition, any programming errors that would lead to undesired behavior in the hardware class are predicted by the operations of the base class, allowing both model and simulation to co-verify one another.

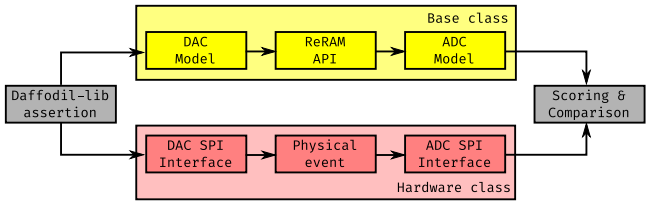

Figure 3: Polymorphism allows for commands to interface either to a simulation of the prototyping hardware or to be routed to the interface drivers.

4 DAFFODIL-APP, VERILATOR, AND NEURAL NETWORK RESULTS

The highest level code is composed of Daffodil-app, a separate library which takes all of the primitives in Daffodil-lib and maps it into dimensionless network operations. These include constructing network layers from multiple kernels, performing outer-product operations on arbitrarily sized layers constructed from kernel-level operations, and running inference/backpropagation operations. The library has predefined values for read and write voltages; algorithmic hyperparameters are the only user-tunable values required. In addition to providing a consistent interface between the simulation and hardware classes that renders them indistinguishable, the app also uses pybind11 to produce a consistent Python interface between C-code, C++ libraries generated from verilated SystemVerilog

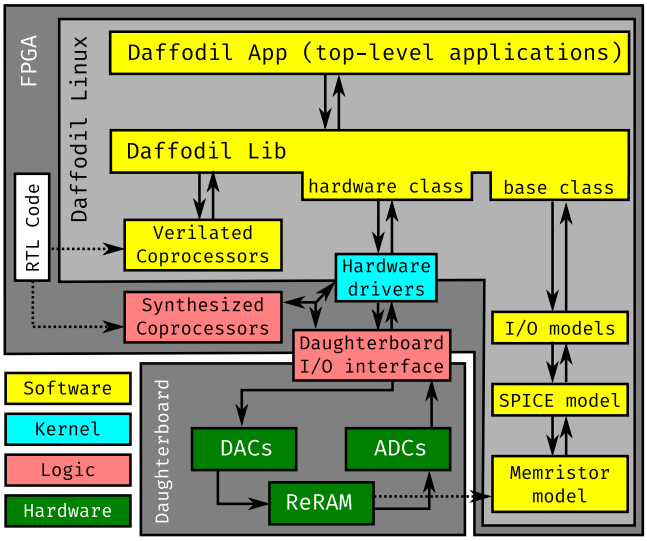

Figure 4: Depiction of the verification framework. The framework can be run from a simulation-only host, run within the FPGA CPU on the embedded OS, be partially synthesized on the FPGA, or interfaced into the physical hardware for the experiment. Solid arrows indicate dataflow; dashed arrows represent the derivation source/destination.

(which model the activity of our own custom compute accelerator or coprocessor modules), and compiled application-level drivers on the FPGA. Consequently, in addition to modeling/operating mixed signal computing, the Daffodil-app enables simulation of a hardware coprocessor from Python script models to a synthesized ASIC model. Using the Daffodil-app, we are able to engage with predictive modeling of network training on our hardware platform. In one model, we used 28 of the 32 available kernels to create a $324 \times 50 \times 10$ network to study a reduced-MNIST problem. Using an ideal ReRAM model with conductance from 1 µS to 100 µS, we show that an operationally exact hardware model of our system can train to acceptable accuracy. In addition to running basic mini-batch gradient descent, we simulate the operation of coprocessors implementing reduced-rank, stochastic training [15, 16].

5 DISCUSSION AND FUTURE WORK

Constructing a design verification platform requires not only building a model of the hardware system, but also a model which is operationally faithful across multiple layers of abstraction. Disciplined end-to-end isomorphism is critical to the tracking of errors and side effects which otherwise might be untraceable when comparing a hardware model to an entirely external reference model.

Daffodil-lib does this through every layer of abstraction. The code models coprocessor operations that describe signals on the board. The ReRAM API includes only details of time and voltage, and does not presuppose any mode of operation in which any of its side effects may be absent. By building and exploiting polymorphism between simulation-control and hardware-control codebases, experimental errors are avoided. Imbuing hardware-control code with syntax identical to that of its simulation not only establishes

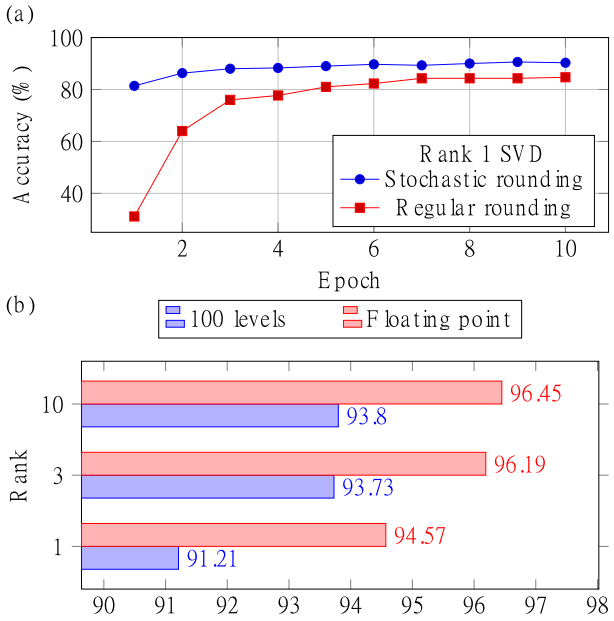

Figure 5: Plots of the simulated training on MNIST. Each point represents the test set accuracy for a single training. The networks were trained using a low rank training algorithm using both round nearest and stochastic rounding. a) Epochal performance of the training. b) Performance as a function of increasing the number of ranks.

this link for catching errors, but also reduces the cognitive load of moving from the modeling phase to implementation.

This system brings many advantages even though simulations are slower than they could otherwise be. Modelling the ReRAM interactions as mixed-signal quantities in this co-simulation environment allows the system to debug errors in the hardware drive routines, or in the application loops. In trouble-shooting bad results, such mistakes can be distinguished from, higher-level problems such as poorly optimized hyperparameters.

The strong organizational correspondence between the hardware architecture and the python code is meant to facilitate a staged design process. Once an algorithm has been validated in simulation and with real devices, the whole design can be streamlined. Individual methods can be replaced with FPGA hardware co-processors with the end goal of completely realizing the end-to-end training data-path in RTL hardware. This top-down design approach can be repeated many times, and adapted to handle idiosyncracies introduced by adjustments in media processing, choice of material, or even experiments using a different memory technology.

6 CONCLUSION

In the context of the Daffodil-lib and associated components, we have discussed a developed operation exact design for verification framework for a resistive neural network prototype. The concepts proposed in it, based on decades of research into design verification and project management, from interfaces, to descriptive modeling, to synaptic polymorphism, can be used to thoughtfully grow a neuromorphic concept from a model to a working experiment.

ACKNOWLEDGMENTS

This work has been partially supported by the DARPA / ONR grant No. N00014-20-1-2031 and GWU University Facilitating Fund. AM acknowledges support under the NIST-UMD Cooperative Research Agreement Award No. 70NANB14H209, through the University of Maryland. Authors acknowledge Sonia Buckley and Mark Stiles for their helpful review of the manuscript.

REFERENCES

[1] Syed Ahmed Aamir, Paul Müller, Gerd Kiene, Laura Kriener, Yannik Stradmann, Andreas Grübl, Johannes Schemmel, and Karlheinz Meier. 2018. A mixed-signal structured adex neuron for accelerated neuromorphic cores. *IEEE transactions on biomedical circuits and systems* 12, 5 (2018), 1027–1037.

[2] Sapan Agarwal, Robin B Jacobs Gedrim, Alexander H Hsia, David R Hughart, Elliot J Fuller, A Alec Talin, Conrad D James, Steven J Plimpton, and Matthew J Marinella. 2017. Achieving ideal accuracies in analog neuromorphic computing using periodic carry. In *2017 Symposium on VLSI Technology*. IEEE, T174–T175.

[3] Christiaan Baaij, Matthijs Kooijman, Jan Kuper, Arjan Boeijink, and Marco Gerards. 2010. Cλash: Structural descriptions of synchronous hardware using haskell. In *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. IEEE, 714–721.

[4] Tobias Bjerregaard and Shankar Mahadevan. 2006. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR)* 38, 1 (2006), 1–es.

[5] Geoffrey W Burr, Robert M Shelby, Severin Sidler, Carmelo Di Nolfo, Junwoo Jang, Irem Boybat, Rohit S Shenoy, Pritish Narayanan, Kumar Virwani, Emanuele U Giacometti, et al. 2015. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Transactions on Electron Devices* 62, 11 (2015), 3498–3507.

[6] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. 2018. NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 12 (2018), 3067–3080.

[7] Wen Chen, Sandip Ray, Jayanta Bhadra, Magdy Abadir, and Li-C Wang. 2017. Challenges and trends in modern SoC design verification. *IEEE Design & Test* 34, 5 (2017), 7–22.

[8] John Clow, Georgios Tzimpragos, Deeksha Dangwal, Sammy Guo, Joseph McMahan, and Timothy Sherwood. 2017. A pythonic approach for rapid hardware prototyping and instrumentation. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–7.

[9] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro* 38, 1 (2018), 82–99.

[10] Laura Fick, David Blaauw, Dennis Sylvester, Skylar Skrzyniarz, M Parikh, and David Fick. 2017. Analog in-memory subthreshold deep neural network accelerator. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–4.

[11] Tom Fitzpatrick. 2004. SystemVerilog for VHDL users. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, Vol. 2. IEEE, 1334–1339.

[12] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Design Patterns. 1995. Elements of Reusable Object-Oriented Software. *Design Patterns. massachusetts: Addison-Wesley Publishing Company* (1995).

[13] Per Haglund, Oskar Mencer, Wayne Luk, and Benjamin Tai. 2003. Hardware design with a scripting language. In *International Conference on Field Programmable Logic and Applications*. Springer, 1040–1043.

[14] Nicholas C Harris, Jacques Carolan, Darius Bunandar, Mihika Prabhu, Michael Hochberg, Tom Baehr-Jones, Michael L Fanto, A Matthew Smith, Christopher C Tison, Paul M Alsing, et al. 2018. Linear programmable nanophotonic processors. *Optica* 5, 12 (2018), 1623–1631.

[15] Brian D Hoskins, Matthew W Daniels, Siyuan Huang, Advait Madhavan, Gina C Adam, Nikolai Zhitenev, Jabez J McClelland, and Mark D Stiles. 2019. Streaming batch eigenupdates for hardware neural networks. *Frontiers in neuroscience* 13 (2019), 793.

[16] Siyuan Huang, Brian D Hoskins, Matthew W Daniels, Mark D Stiles, and Gina C Adam. 2020. Memory-efficient training with streaming dimensionality reduction. *arXiv preprint arXiv:2004.12041* (2020).

[17] Shunning Jiang, Peitian Pan, Yanghui Ou, and Christopher Batten. 2020. PyMTL3: a Python framework for open-source hardware modeling, generation, simulation, and verification. *IEEE Micro* 40, 4 (2020), 58–66.

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

[19] Malte J Rasch, Diego Moreda, Tayfun Gokmen, Manuel Le Gallo, Fabio Carta, Cindy Goldberg, Kaoutar El Maghraoui, Abu Sebastian, and Vijay Narayanan. 2021. A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays. *arXiv preprint arXiv:2104.02184* (2021).

[20] Rindert Schutten and Tom Fitzpatrick. 2003. Design for verification.

[21] John Sweeney. 2020. Techniques for performing design verification of an optical processor. https://medium.com/lightmatter/techniques-for-performing-design-verification-of-a-mixed-signal-digital-analog-and-optical-d0068c9ff868

[22] Cheng Tan, Yanghui Ou, Shunning Jiang, Peitian Pan, Christopher Torng, Shady Agwa, and Christopher Batten. 2019. PyOCN: A unified framework for modeling, testing, and evaluating on-chip networks. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 437–445.