FIONA: Photonic-Electronic CoSimulation Framework and Transferable Prototyping for Photonic Accelerator

Yinyi Liu[1,†], Bohan Hu[2,†], Zhenguo Liu[2], Peiyu Chen[2], Linfeng Du[1], Jiaqi Liu[1], Xianbin Li[1], Wei Zhang[1], Jiang Xu[2,*]
[1]Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology
[2]Microelectronics Thrust, The Hong Kong University of Science and Technology (Guangzhou)

Abstract

Recent advances in the architecture design for photonic accelerators have demonstrated great promise to accelerate deep neural network (DNN) applications, and also allude to the essential collaboration of the electronic subsystems for efficient logic arithmetic and memory access. However, available tools to design and evaluate photonic accelerators usually neglect the cross-stack effects or low-level details in real-world scenarios, ranging from programming-stack inefficiency to electronic peripheral implementation complexity. This frustrating fact makes it difficult to holistically estimate the performance metrics of a practical photonic-electronic collaborative computing system. In addition, until now, no toolchain can provide programmable, hardware-reconfigurable, and end-to-end rapid verification for photonic accelerators.

Here we present FIONA, a Full-stack Infrastructure for Optical Neural Accelerator, which comprises a photonic-electronic co-simulation framework for multi-level design space exploration (DSE), and a transferable hardware prototyping template for physical verification. Specifically, the co-simulation framework consists of a functional simulator at the instruction set architecture (ISA) level to agilely verify the programming software stack and a register-transfer level (RTL) cycle-accurate simulator to precisely profile the overall system. We also demonstrate *LightRocket* as a case study of the FIONA toolchain to show the full workflow of designing a Turing-complete photonic accelerator system that supports arbitrary DNN workloads and on-chip training. The toolchain is open-sourced and available at *https://github.com/hkust-fiona/*.

Keywords: Photonic Accelerator, Full-stack Implement, Simulation, Transferable Prototyping, Design Space Exploration

1 Introduction

As Moore's law slows down, computing systems must pivot towards more powerful computational capabilities and more rational architectures to enable continuous performance growth. Recent years have seen a trend that photonic-based processing units are promising to perform vector operations faster with higher energy efficiency than their electronic counterparts, while the optimal architecture of the photonic accelerator is yet to be determined.

With the emergence of research at either the architecture level or the device level, we observe a design philosophy shift: from inference-only to supporting hardware-aware training, from accelerating only a single task to supporting multiple tasks of different DNN workloads. In other words, compared to the early-stage photonic accelerators that focused on improving the inference-phase performance on a single application, such as convolutional neural network (CNN), nowadays, photonic accelerators are expected to support general-purpose computation for various neural applications and adapt to process variation and quantization error through on-chip training.

We conduct a comparison among recent representative works about photonic computing, as shown in Table 1. They can be categorized into two main sets: (i) the architecture exploration and (ii) the device prototyping of standalone photonic accelerators. [1, 2] focus on the arrangement of photonic components *inside* a photonic core and do well in minimizing the footprint and energy consumption to achieve complex arithmetic in the optical domain. However, they ignore the low-level details such as memory transaction and scheduling overheads *outside* the photonic core. They adopt formulaic estimation to roughly investigate the uncore units, which is fast but results in optimism bias in evaluation. [3, 4] propose architecture designs involving the interplay between electronic and optical domains, but still modeling in an abstract event granularity. [5, 6] introduce two valuable tools to simulate DNN workloads on MZI meshes at the algorithm level. However, the those mentioned above are more likely functional design kits and do not yet lower to the register-transfer level (RTL) or hardware level. Therefore, they are not physically available for verification. [7–9] do tape-out the chip and are ready to run in a real environment, but they are highly customized for specific DNN workloads and lack programmability.

In summary, we observe a gap between the system-level design and device-level prototyping in the fields of photonic accelerators. No available toolchain can provide a complete solution spanning software and hardware stacks to enable rapid end-to-end photonic accelerator design.

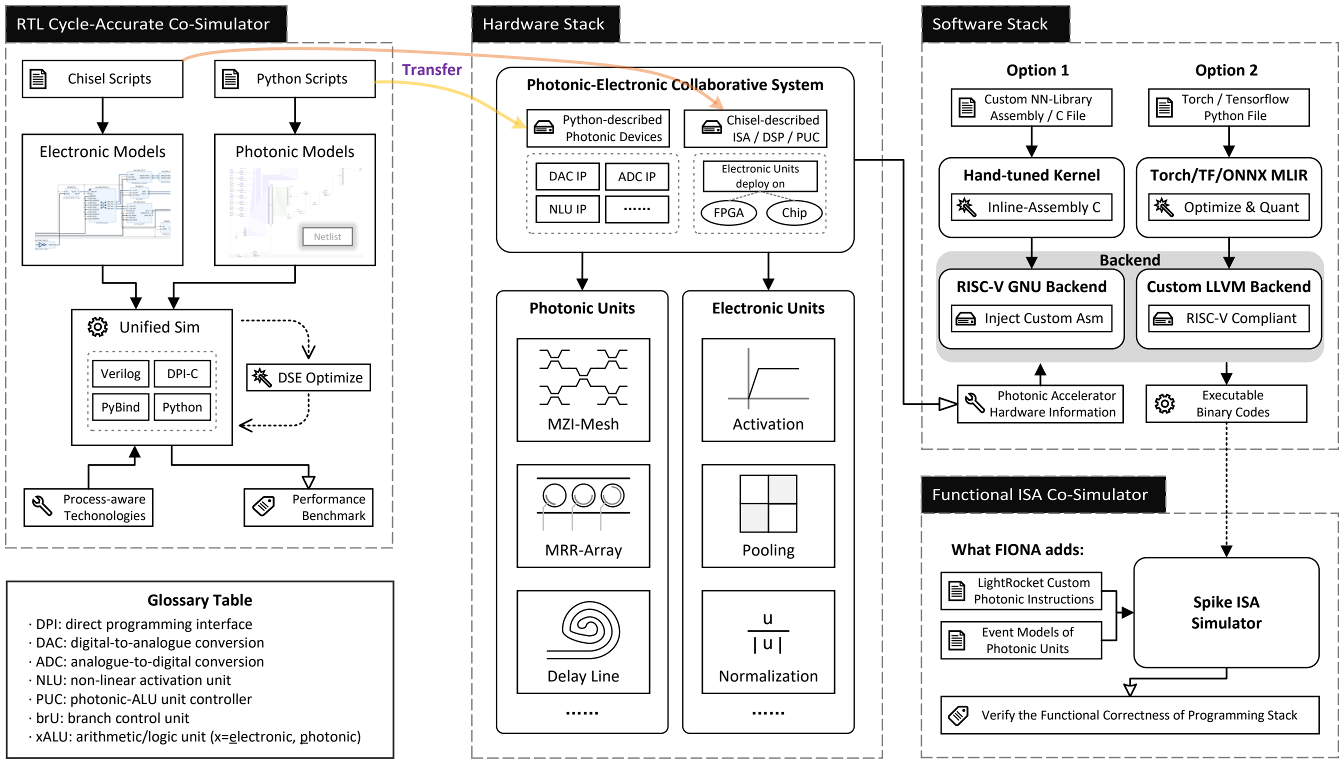
Table 1. Comparison: Progress in Photonic Computing

Related Works	Training Support	Multi-Task Reconfigurable	Simulator for DSE	Physically Executable
[1, 2]	✔		✔	
[3, 4]	✔	✔		
[5, 6]	✔	✔	✔	
[7–9]	✔	❍		✔
Ours	✔	✔	✔	✔

*Note: ❍ denotes it only supports specific DNN or no software stack.

To bridge the gaps mentioned above, we propose FIONA, a full-stack infrastructure for optical neural accelerators. FIONA comprises a photonic-electronic collaborative simulation framework for multi-level design space exploration

†Both authors contributed equally to this work.
*Corresponding author: *jiang.xu@ust.hk*
This work is supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No.2023A03J0013).



Glossary Table

- · DPI: direct programming interface
- · DAC: digital-to-analogue conversion
- · ADC: analogue-to-digital conversion
- · NLU: non-linear activation unit
- · PUC: photonic-ALU unit controller
- · brU: branch control unit
- · xALU: arithmetic/logic unit (x=electronic, photonic)

Figure 1. Overview of FIONA Toolchain.

and a transferable hardware prototyping template for rapid physical verification. Specifically, the co-simulation framework consists of a functional ISA-level simulator to verify the correctness of the software stack, and an RTL cycle-accurate simulator to profile the overall computing system in detail. We also demonstrate *LightRocket* as a case study of the FIONA toolchain. *LightRocket* is compatible with the RISC-V ecosystem and can perform custom photonic instructions. We disclose the complete design-and-transfer workflow of developing a Turing-complete photonic accelerator system that supports arbitrary DNN workloads and on-chip photonic-electronic collaborative training.

To our best knowledge, FIONA is the first toolchain spanning both software and hardware stacks for photonic accelerator systems. It enables hierarchical simulation, painlessly transfering the implementation to hardware, and end-to-end rapid prototyping.

The remainder of this paper is organized as follows. Section II illustrates the constitution of the FIONA toolchain. Section III conducts a case study, *LightRocket*, to exemplify how to transfer an architecture design from simulation to physical prototyping. Section IV discusses the applicable scope covered by FIONA. Section V concludes this work.

2 FIONA Toolchain

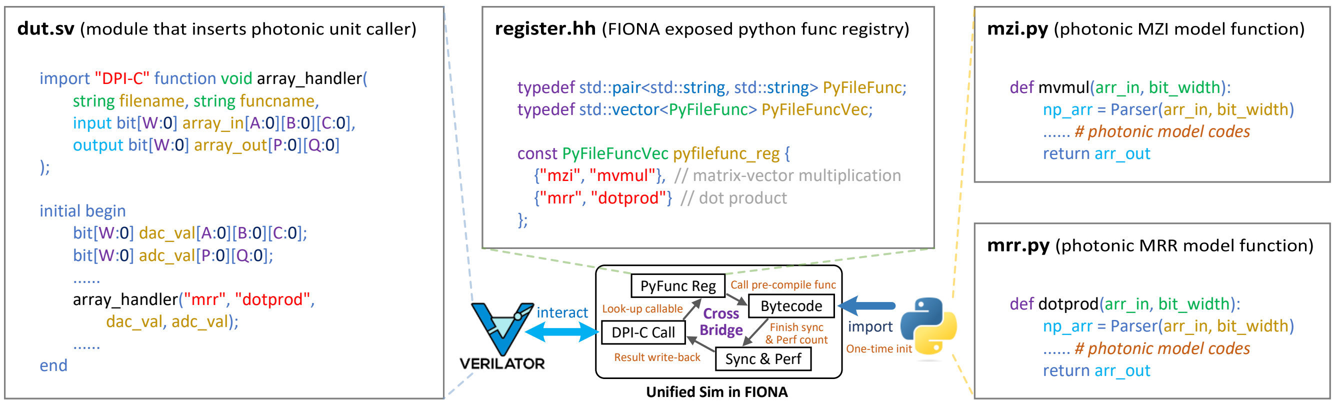
This section elucidates the proposed FIONA toolchain, including hardware and software stacks. Note that the RTL co-simulator and the functional ISA co-simulator (Co-Sim) are the tools to simulate for two stacks, respectively. The overview of FIONA is shown in Figure 1.

The design philosophy of FIONA is: *one-time development, multi-way deployment.* (i) As for the hardware stack and its RTL Co-Sim, the simulation and the prototyping share the same design source files of photonic and electronic units. The hardware prototyping template is organized in a modular design manner. (ii) As for the software stack, the generated machine codes can verify on its functional ISA Co-Sim or run on physical prototypes in a bare-metal mode.

2.1 RTL Photonic-Electronic Co-Simulator

Under this simulator framework, we focus on the collaborative simulation of electronic and photonic units. Most previous works focus on designing the photonic computing units in isolation without considering the interaction with their electronic peripherals. However, when designing accelerators, it is essential to pay attention to the timing of the datapath, memory access pattern, and dataflow optimization to gain higher utilization and throughput. Such kinds of optimizations are at the micro-architecture level and are susceptible to datapath design such as pipeline stages segmentation, which requires cycle-accurate simulation to profile the metrics such as pipeline stalls.

The emergence of photonic computing devices brings more significant challenges to the design and implementation of the architecture. When designing the datapath and control logic, we need to consider more factors like compute


dut.sv (module that inserts photonic unit caller)

```
import "DPI-C" function void array_handler(
      string filename, string funcname,
      input bit[W:0] array_in[A:0][B:0][C:0],
      output bit[W:0] array_out[P:0][Q:0]
);

initial begin
      bit[W:0] dac_val[A:0][B:0][C:0];
      bit[W:0] adc_val[P:0][Q:0];
      ......
      array_handler("mrr", "dotprod",
            dac_val, adc_val);
      ......
end
```

register.hh (FIONA exposed python func registry)

```
typedef std::pair<std::string, std::string> PyFileFunc;
typedef std::vector<PyFileFunc> PyFileFuncVec;

const PyFileFuncVec pyfilefunc_reg {
   {"mzi", "mvmul"}, // matrix-vector multiplication
   {"mrr", "dotprod"} // dot product
};
```

mzi.py (photonic MZI model function)

```
def mvmul(arr_in, bit_width):
      np_arr = Parser(arr_in, bit_width)
      ...... # photonic model codes
      return arr_out
```

mrr.py (photonic MRR model function)

```
def dotprod(arr_in, bit_width):
      np_arr = Parser(arr_in, bit_width)
      ...... # photonic model codes
      return arr_out
```

Figure 2. Cross-Domain Simulation Workflow of Unified-Sim in FIONA.

latency, conversion across optical-electrical domains, etc. Making photonic and electronic devices co-integrate better requires more exploration at the micro-architecture level.

To address the lack of a co-simulation framework, we propose a simulator that aims to perform the cycle-accurate simulation at the RTL to profile the hardware-aware metrics before transferring to a physical prototype.

The existing dominant way to describe electronic circuits is Hardware Description Language (HDL). To facilitate the system design and smooth DSE, we adopt Chisel [10] HDL to implement parameterizable circuits and produce simulatable and synthesizable code. Since we are projecting that FIONA can serve as the toolchain to perform general photonic-electronic collaborative computing that is extensible and compatible with the current state-of-the-art ecosystem in electronics, the RISC-V instruction set and the rocket-chip framework becomes the best candidate to set off. The RISC-V ISA reserves custom instruction slots, and the rocket chip has a uniform interface, RoCC [11], for accelerators. We extend a typical RISC-V ecosystem and propose our RTL photonic-electronic co-simulator framework, which consists of the following parts:

1. **Photonic Accelerator Interface**. Photonic cores at the developing stage usually do not lithograph the on-chip electronic circuits of controllers. Therefore, the controller is integrated with the vector processor.
2. **FIONA-V Vector Processor**. FIONA-V is a ready-to-run baseline photonic-electronic vector processor designed for fast system-level prototyping. Since the photonic chips show great capability in performing vector operations, we implement a baseline vector processor in the electrical domain and provide vector datatype supports. FIONA-V is customizable and pluggable for new photonic instructions. The ISA and architecture of FIONA-V are described in Section 3.2.2.

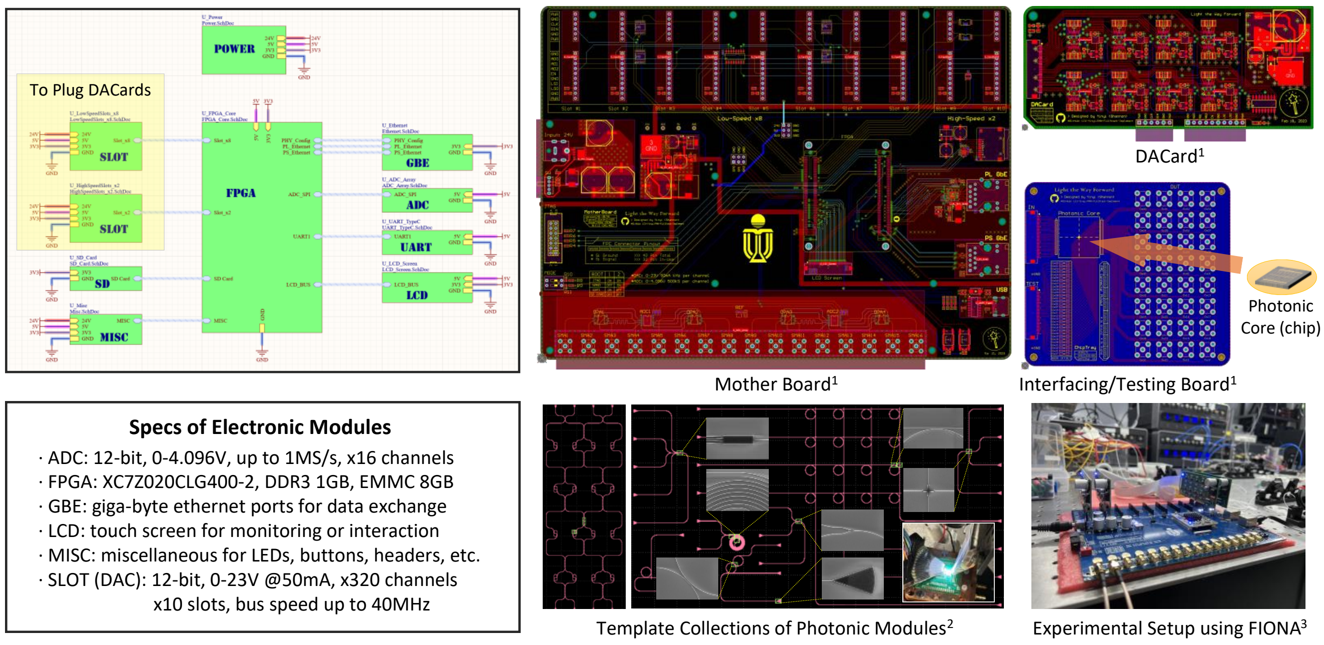
The detailed implementation and the instantiation of our example design are elaborated in Section 3. The RTL cycle-accurate simulation is supported by the commonly-used RTL simulators, *e.g.*, Verilator [12] and etc. The simulation will produce architectural performance metrics like throughput, latency, and execution cycles, as well as technology-aware metrics such as power consumption.

2.1.1 Photonic Computing Units Modeling.

The photonic models are described in Python. The input arguments are the quantized data fetched from the *cross bridge* as shown in the bottom center of Figure 2. We provide a FIONA built-in utility class to automatically unpack the data into *numpy* or *torch* supported forms. The models can be constructed by either formula from optic theories or look-up table data from experimental measurements. Commonly-used photonic units are included in FIONA built-in libraries: Mach-Zehnder Interferometer (MZI) Mesh, Micro-Ring Resonator (MRR) Array, Optical Delay Line (ODL), etc. The mentioned units are attached with the parametric GDSII generators using the python package *gdsfactory* [13], which enables the chip layout generation for prototyping and the simulation in a multi-physics simulator such as *Lumerical* [14].

2.1.2 Cross-Domain Simulation and Synchronization.

The electronic and photonic models are simulated in the corresponding domains, respectively. To bridge the data and align the clock between Verilator and Python-Photonics, FIONA's Unified-Sim provides a cross-domain synchronization framework, as shown in Figure 2. Unified-Sim provides built-in *Registry*, *Scheduler*, and *Monitor*. Once an instruction that belongs to the photonic operations is triggered, the cross bridge will evoke a handler through the direct programming interface (DPI) C/C++. The handler passes the name of the target Python-model function and the multi-dimension data of input operands as arguments to *Registry*, which will look up the callable objects from imported Python bytecodes. *Scheduler* stalls the simulation in the electrical domain, waits for the computational results from Python models, and writes the quantized data back to the electrical domain through DPI-C. *Monitor* counts the latency and the energy consumption of DAC, ADC, and photonic models.


To Plug DACards
POWER
FPGA
SLOT
SLOT
SD
MISC
GBE
ADC
UART
LCD
DACard1
Mother Board1
Interfacing/Testing Board1
Photonic Core (chip)
Template Collections of Photonic Modules2
Experimental Setup using FIONA3

Specs of Electronic Modules

- ADC: 12-bit, 0-4.096V, up to 1MS/s, x16 channels
- FPGA: XC7Z020CLG400-2, DDR3 1GB, EMMC 8GB
- GBE: giga-byte ethernet ports for data exchange
- LCD: touch screen for monitoring or interaction
- MISC: miscellaneous for LEDs, buttons, headers, etc.
- SLOT (DAC): 12-bit, 0-23V @50mA, x320 channels x10 slots, bus speed up to 40MHz

Figure 3. Transferable Prototyping Template: (1) schematics and PCB layouts of electronic modules, (2) chip layouts and scanning electron microscope (SEM) images of photonic modules, (3) experimental setup using FIONA prototyping toolchain.

2.1.3 Practicality Discussion. There is skepticism about the simulation speed and the necessity to run at the RTL. Here we address the two questions from the aspects of the software engineering and the manufacture, respectively:

1. **Performance Consideration:** We notice that the Python-model functions are frequently called during the simulation process. To ameliorate the performance, the Python scripts are dedicatedly designed to load and parse only at the initial stage. The interpreted bytecodes are stored as C/C++ objects in the memory. Therefore, the simulation speed of Python-Photonics would be as fast as that of C/C++.
2. **Simulation Granularity:** Photonic computing units are highly customized and their timing quality is delicate. Therefore, we need to carefully inspect the functionality and timing cycle-accurately. Furthermore, we hope that through RTL simulation, architects will have the opportunities to observe the micro-architecture level interaction between photonic and electronic units. This mechanism provides designers an approach to integrate their photonic models seamlessly with the electrical-domain RTL. Moreover, the framework also produces synthesizable codes that can be implemented on FPGAs for fast prototyping. RTL simulation is the last step of the Pre-Silicon phase, which is indispensable for verifying the timing requirements and profiling the system performance before fabrication. We decouple the fast pre-RTL functional simulation to the ISA Co-Sim of the software stack.

2.2 Transferable Prototyping Template

The prototyping template aims to provide reusable modules for flexible migration from the schematic design and serves as the key part of rapid prototyping. **Transferable** refers to the capability of seamless migration from simulation environment to physical prototypes thanks to the *one-time development, multi-way deployment* philosophy. The overview of electronic and photonic modules is shown in Figure 3.

2.2.1 Electronic Modules. To enable transferring the DSE optimized solution in the simulator to the physical modifications of the system architecture, electronic modules are intended to decouple and design for painless customization:

- Cross-module interfaces are maximally devised in a pluggable way, except for the integrity-sensitive ones. Changes in the inter-module connectivity of the system architecture in the simulator correspond to handily unplug-and-reconnect the modules' topology.
- Intra-module customization is also allowed. It just requires unplug-and-replace the dedicated design by following the protocols of cross-module interfaces.

The electronic module template encompasses four types of printed circuit boards (PCBs): FPGA board, mother board, digital-to-analogue (DA) card, interfacing/testing board.

- **FPGA Board:** The Verilog codes generated by Chisel scripts are implemented on the FPGA. It is plugged into the slot on Mother Board (center), and handles all the electronic digital instructions and data.

- **Mother Board:** It provides proper driven power, the signal bus towards slots of DAC and ADC arrays, and the IO pins for all the peripheral communications.
- **DA Card:** There are 10 DAC slots on Mother Board (top). Each slot can hold one DA Card. Each card has 32 output channels in a form of FPC connectors (left).
- **Interfacing/Testing Board:** Wire bonding with the photonic chip. It also serve DAC-ADC loopback test.

2.2.2 Photonic Modules.

FIONA provides a set of built-in parametric GDSII layout generators for general devices, *e.g.*, MZI and MRR. Adding a customized device generator is also applicable. The photonic modules are building blocks to construct the photonic units, which perform the photonic instructions. For example, properly arranged MZIs constitute an MZI mesh that can perform matrix-vector multiplication. More photonic instructions are listed in Section 3.

2.3 Software Stack and Functional ISA Simulator

Besides the hardware stack, FIONA also provides an easily extendable programming software stack to boost developers' productivity. Inspired by Gemmini [15], we develop a multi-level software flow. (i) At the high level, the DNN models described in PyTorch/Tensorflow/ONNX are converted to the multi-level intermediate representation (MLIR), and our custom LLVM backend will interpret the MLIR to LLVM IR and later the machine binaries. (ii) At the middle level, FIONA provides the hand-tuned kernels of commonly-used photonic units. By specifying the backend macros in C/C++ files, the kernel will execute instructions on the corresponding FIONA custom units. Table 2 lists the supported kernels and their backends. (iii) At the low level, developers can also program through C/C++ with inline assembly macros.

Table 2. Supported Hand-tuned Kernels and Backends

Kernel (Math Operator)	Backend (Target)	Kernel (DNN Module)	Backend (Target)
tiled_dotprod	MRR	nn_linear	MRR / MZI
tiled_mvmul	MRR / MZI	nn_conv	MRR / MZI
tiled_{eALU}	{eALU}.V	nn_batchnorm	{eALU}.V
tiled_{NLU}	{NLU}.V	nn_padding	SHUFFLE.V
tiled_{MISC}	{MISC}.V	nn_maxpooling	MAX.V
tiled_dropout	VMASK	nn_mhattention	MRR / MZI
residual_connect	ADD.V	nn_embedding	RV

****Note:** RV denotes RISC-V standard ISA. The rest refers to Section 3.*

3 Application and Case Study

In this section, we demonstrate *LightRocket* as a case study of the FIONA toolchain to exemplify the full workflow of designing a photonic accelerator system: (i) the ISA design with functional software simulation, (ii) the cross-domain RTL simulation and profiling, and (iii) the rapid prototyping migration. We also discuss how to implement advanced DNN operators and on-chip training.

3.1 Statement of Application Scenario

Suppose we come up with an idea to leverage the micro-ring resonator (MRR) weight bank [16] structure to perform the dot-product operations, as shown in Figure 4. In the following subsections, we will give a step-by-step guide to how to build up a photonic accelerator system from scratch using the FIONA toolchain. This demo is entitled as *LightRocket*.

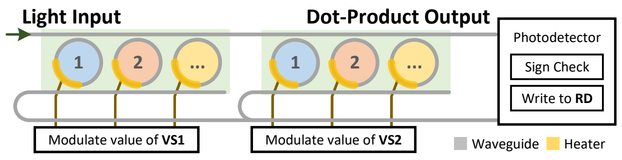


Figure 4. Schematic of Thermo-Optic MRR Weight Banks to perform Dot Product. The modulation process at each wavelength is equivalent to multiplication. The wide-spectral photodetection approximates the accumulation.

3.2 ISA Design

Table 3 shows the FIONA baseline ISA of photonic-electronic vector processing instructions.

3.2.1 Instruction.

As the name suggests, this custom ISA comprises vector-oriented photonic operations (pOps) and electronic operations (eOps). Each vector instruction can be implemented in either an electrical or optical domain, depending on the functionality of the photonic core. In FIONA-V, the function unit routing rule can be easily customized. For example, the instructions in the non-linear unit (NLU) category can also remap to photonic units, if any. The FIONA baseline ISA summarizes the most common-used operations while keeping them as neat and reduced as possible. Developers are free to extend their new custom instructions, either pOps or eOps, onto the FIONA baseline ISA. We take MRR-based *DotProd* instruction affiliated to the **pALU** category as the example to demonstrate the *LightRocket* design process.

3.2.2 Register Set.

Besides the vector registers, the FIONA baseline ISA also defines a special group of register sets for configuration (CFG). The *STRIDE* register, with the default value of 1, only affects the address jumps of the memory (MEM) load/store instructions. The *VLEN* register determines the effective execution length of all vector instructions. Only the elements of which index lies between 0 and *VLEN*-1 will commit to compute. It tackles the remainder segments when reaching the margin of a long vector. Alternatively, the *VMASK* is used for discrete element selection by a bit mask. The *MAT* is dedicated to *MVMul* instruction.

3.3 System Architecture

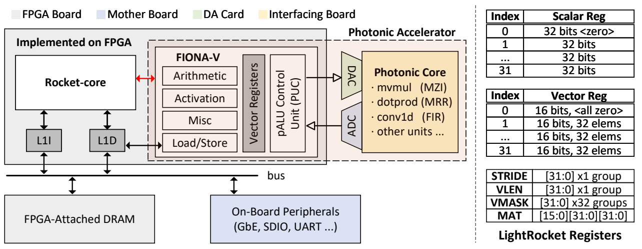
The system architecture design of *LightRocket* is shown in Figure 5. The system consists of a rocket core to serve execution flow control, a photonic accelerator to accelerate vector

Table 3. Overview of LightRocket Custom Instruction Set Architecture

Category	Instruction	Operands								Description
		Funct7 [31:25]	VS2/RS2 [24:20]	VS1/RS1 [19:15]	Funct3 [14:12]			VD/RD [11:7]	OpCode [6:0]	
					xd	xs1	xs2			
pALU	DotProd	41H	V	V	1	0	0	S	0x0B	MRR: RD = VS1[i] · VS2[i]
	MVMul	42H	U	V	0	0	0	V		MZI: VD[i] = MAT @ VS1[i]
	Conv1D	43H	V	V	0	0	0	V		FIR: VD[i] = VS1 ⊛ VS2
eALU	ADD.V	01H	V	V	0	0	0	V		VD[i] = VS1[i] + VS2[i]
	SUB.V	02H	V	V	0	0	0	V		VD[i] = VS1[i] - VS2[i]
	ADD.VS	03H	S	V	0	0	1	V		VD[i] = VS1[i] + RS2
	SUB.VS	04H	S	V	0	0	1	V		VD[i] = VS1[i] - RS2
	MUL.VS	05H	S	V	0	0	1	V		VD[i] = VS1[i] * RS2
	DIV.VS	06H	S	V	0	0	1	V		VD[i] = VS1[i] / RS2
MISC	SHUFFLE.V	0AH	V	V	0	0	0	V		VD[i] = VS1[VS2[i]]
	MAX.V	0BH	0H	V	1	0	0	S		RD = Max(VS1[i])
	MIN.V		1H	V	1	0	0	S		RD = Min(VS1[i])
NLU	PRELU.V	0FH	S	V	0	0	1	V		Leaky param: α = RS2
	TANH.V		1H	V	0	0	0	V		VD[i] = f(VS1[i])
	SIGMOID.V		2H	V	0	0	0	V		where f is nonlinear function
MEM	LOAD.V	10H	U	S	0	1	0	V		VD[i] = Mem[RS1+i*STRIDE]
	STORE.V	11H	V	S	0	1	0	U		MEM[RS1+i*STRIDE] = VS2[i]
CFG	SET.R	18H	U	S	0	1	0	STRIDE		Reg: STRIDE = RS1
			U	S	0	1	0	VLEN		Reg: VLEN = RS1 (i = 0 to VLEN-1)
			S	S	0	1	1	VMASK		Reg: VMASK[RS2] = RS1
			S	S	0	1	1	MAT		Reg: MAT[RS2+i] = Mem[RS1+i]

*Note: V, S, U denote Vector Register, Scalar Register, and Unused, respectively. STRIDE, VLEN, VMASK, MAT are the FIONA custom registers.

operations, external memory, and peripherals. More specifically, the photonic accelerator (PA) comprises the FIONA-V, the photonic accelerator interface, and the photonic core. We will illustrate PA in the following parts.



Index	Scalar Reg
0	32 bits <zero>
1	32 bits
...	32 bits
31	32 bits

Index	Vector Reg
0	16 bits, <all zero>
1	16 bits, 32 elems
...	16 bits, 32 elems
31	16 bits, 32 elems

STRIDE	[31:0] x1 group
VLEN	[31:0] x1 group
VMASK	[31:0] x32 groups
MAT	[15:0][31:0][31:0]

Figure 5. System Architecture and Board Assignment of *LightRocket*. The red arrows denote the RoCC interface.

3.3.1 FIONA-V.

The FIONA-V vector processor described in Section 2.1 utilized the RISC-V [17] custom instruction slots to support basic vector operations. It is a baseline design mainly for fast prototyping and design space exploration. FIONA-V has a standalone dual-banked vector register file (VRF) accessible by electronic and photonic units. In the electrical domain, FIONA-V contains an arithmetic unit to execute addition, subtraction, multiplication, and division in both vector-vector and vector-scalar fashion. We also implemented a hardware activation function unit utilizing the table lookup and interpolation methods. With these function units, FIONA-V can handle most computational tasks covered in the DNN workloads.

FIONA-V is dedicated to providing vector computation capability in both electrical and optical domains. Hence, it also reserves datapath for photonic operations. The PUC in Figure 5 is the default interface that handles the data interaction between electrical and optical domains. It decodes the photonic instructions, convert the vectors into a photonic-compatible format, and feeds them to the photonic core.

Since FIONA-V is designed for accelerating computation, it does not implement the complete RISC-V instruction set. To support execution flow control and scalar operations, it works with Rocket-core through the RoCC interface. They share the frontend, integer register file, decoder, and the L1 data cache. FIONA-V has direct access to the L1 cache to move the data between VRF and the L1 data cache independently without interfering with the scalar core to diminish the need for scalar load/store instructions for data transfer.

3.3.2 Photonic Accelerator Interface.

For *LightRocket*, the photonic accelerator interface in FIONA is instantiated as the PUC, ADC, and DAC in Figure 5. The DAC/ADC interfaces handle signal conversions. The PUC is customized to program the DAC using the vector values to modulate the on-chip photonic components and then program the ADC to sample the outputs from the photonic core.

3.3.3 Photonic Core.

The photonic core contains many kinds of photonic computing units, for example, the MZI mesh that supports matrix-vector multiplication, the MRR weight bank that supports dot product, and the FIR array that approximates 1D convolution. Developers can also design and add new photonic computing units on the photonic core.

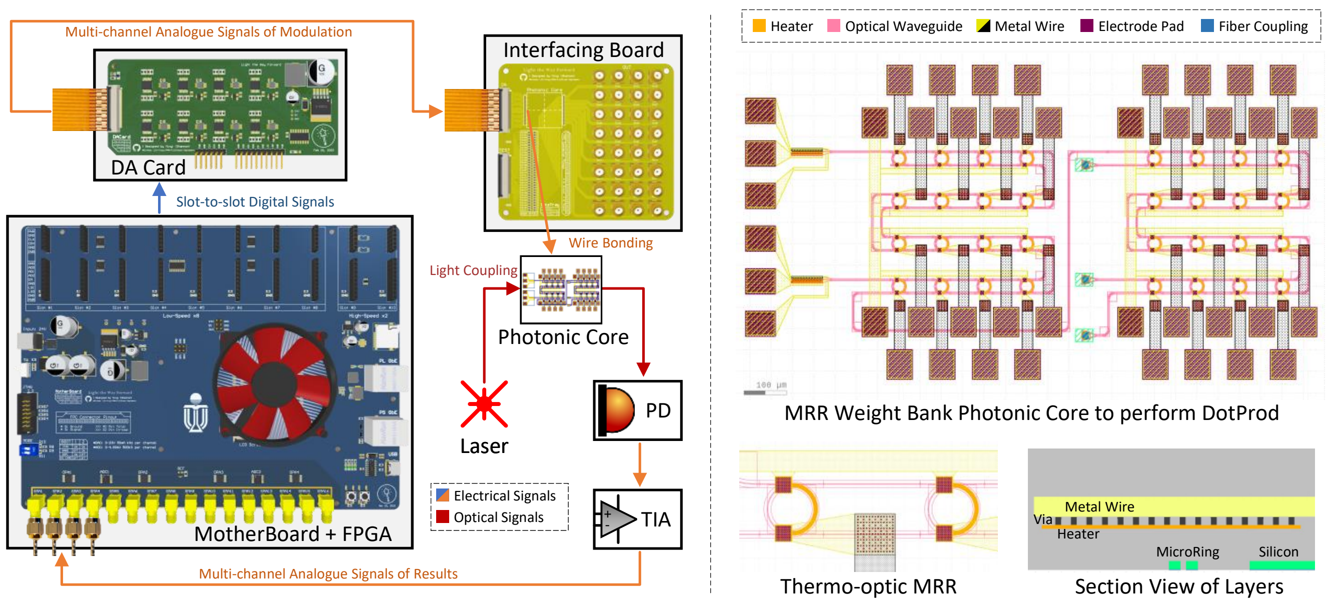


Figure 6. LightRocket Hardware Setup using FIONA Toolchain.

3.4 Software Stack and Pre-RTL Simulation

We modify the Spike [18] simulator to support the baseline FIONA ISA. Spike is an ISA-level simulator that can execute RISC-V binary executables. We implement the behavior-level functional models, decode logic in C++ and integrate them with the Spike simulator. Following the *one-time development, multi-way deployment* design philosophy of FIONA, we provided a wrapped interface that the behavior model developed for the RTL simulator in Python, C, or C++ can be reused in the functional simulator.

3.4.1 Hand-tuned Kernel Design. The flow to wrap a hand-tuned kernel follows: (i) add C/C++ macros as the interface to assembly-form instructions, (ii) properly setup the configuration (CFG) register set and handle vector memory access for the preparation of the custom photonic instructions, and (iii) tile the arbitrary array and schedule to fit the computation patterns on targeted photonic computing unit backends. The hierarchical wrapping is shown in Figure 7.

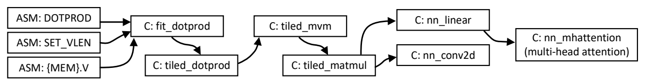


Figure 7. Progressively Wrap Kernels for Targeted Backend.

3.4.2 Functional ISA Simulation. The next step is to profile the DNN workloads of interest at the instruction level. A statistic tool is added to Spike for counting the call usage of different instructions. We build several prevailing DNN models using the hand-tuned kernels pointing to the custom MRR-based DotProd backends. In Figure 8(a), the execution cycle breakdown reveals the percentage of each kind of operation, which provides a clear metric for developers to further optimize the holistic system.

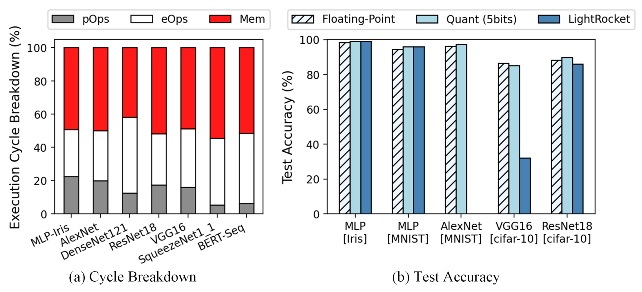
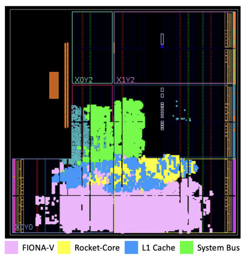


Figure 8. Profile DNN Workloads on MRR DotProd Backend.

We can also involve the cross-domain dataflow analysis if the response models of photonic computing units are available. As mentioned in the previous section, photonic models can be acquired from theoretical derivation or experimental data. Once the models are ready, FIONA ISA Co-Sim can give a fast simulation to estimate the quantity of difference due to the change of backends. Figure 8(b) shows the difference and loss of test accuracy under various situations: floating-point units and 5-bit quantized processing units at a modern electronic computer, and photonic computing units at a *LightRocket*. We can also locate and resolve the software bugs before physically running at a real system. For example, AlexNet almost produces wrong results at the *LightRocket*. After printing the outputs of each layer in AlexNet, the reason is found: AlexNet has 11×11 big convolution kernels, and their summation results are so large that they trigger the register overflow.

3.5 RTL Implementation Result

We implemented part of the LightRocket system on Xilinx Zynq-7000 FPGA xc7z020clg400. We configured the FIONA-V to have two vector lanes. The electronic part of the prototyping system runs at 50MHz and can be entirely fit into the selected FPGA. The implementation result on the FPGA and the resource utilization breakdown is shown in Figure 9.



(a) FPGA Implementation Results

Function Unit	LUTs (%)	Regs (%)	Block RAM	DSP
VRF	4%	4%	**15**	0
Activation	8%	10%	1	**4**
ALU	**38%**	22%	0	0
Config	10%	**31%**	0	0
LSU	3%	11%	0	0
MDU	13%	12%	0	2
PUC	23%	11%	7.5	0
Total	**100%**	**100%**	**23.5**	**6**

(b) FIONA-V Resource Utilization Breakdown

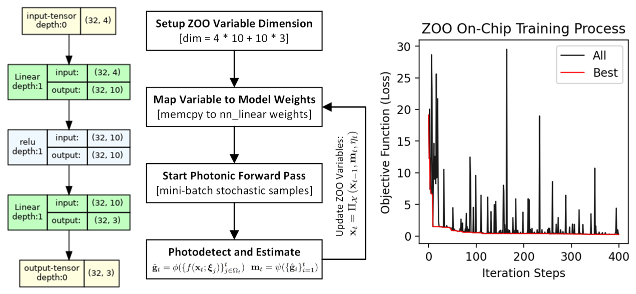
Figure 9. Implementation Result of *LightRocket*.

3.6 Transfer to Hardware Prototyping

The migration process contains: (i) implementing the design on FPGA, (ii) instantiating and assigning modules to boards, and (iii) generating photonic cores for tape-out. The hardware setup is shown in Figure 6. From the board-assignment perspective, let us retrospect the *LightRocket* architecture in Figure 5. When transferring to the hardware prototyping, PUC is implemented on FPGA together with the electronic core and the L1 cache. The photonic accelerator interface is instantiated as DAC and ADC data bus.

3.7 On-Chip Training Supports

Many previous works adopt the optical-forward and electrical-backward flow as the on-chip training scheme. In other words, they compute the forward pass using the photonic units and perform the backward propagation through the electronic DSP. Although *LightRocket* is Turing-complete and supports such a training flow, such gradient-based optimization is extremely expensive. Alternatively, we experimentally implement the on-chip zeroth-order optimization (ZOO) [19] for tuning the pre-trained models, as shown in Figure 10. It only requires random sampling through the forward pass to approximate a quasi-gradient. The drawback is that it suffers higher variance and takes more steps to converge. Therefore, applying ZOO at the on-chip fine-tuning phase is more suitable than training from scratch.


input-tensor depth:0 | (32, 4)
Linear depth:1 | input: (32, 4) | output: (32, 10)
relu depth:1 | input: (32, 10) | output: (32, 10)
Linear depth:1 | input: (32, 10) | output: (32, 3)
output-tensor depth:0 | (32, 3)
Setup ZOO Variable Dimension [dim = 4 * 10 + 10 * 3]
Map Variable to Model Weights [memcpy to nn_linear weights]
Start Photonic Forward Pass [mini-batch stochastic samples]
Photodetect and Estimate $\hat{\mathbf{g}}_t = \phi(\{f(\mathbf{x}_t;\boldsymbol{\xi}_j)\}^t_{j\in\Omega_t})$ $\mathbf{m}_t = \psi(\{\hat{\mathbf{g}}_i\}^t_{i=1})$
Update ZOO Variables: $\mathbf{x}_t = \Pi_{\mathcal{X}}(\mathbf{x}_{t-1}, \mathbf{m}_t, \eta_t)$
ZOO On-Chip Training Process
All
Best
Objective Function (Loss)
Iteration Steps
0
5
10
15
20
25
30
0
100
200
300
400

(a) MLP-Iris Model (b) On-Chip ZOO Training Flow (c) Loss versus ZOO Iteration Steps

Figure 10. Zeroth-Order Optimization On-chip Training.

4 Applicable Scope Covered by FIONA

FIONA toolchain provides a complete end-to-end solution for photonic accelerator systems spanning both software and hardware stacks. Since FIONA decouples software and hardware stacks by functional virtualization, researchers in the field of heterogeneous compilation can use software stacks to investigate the relationship among ISA, simulator, and compiler, without painfully mastering the hardware details as a prerequisite. From the photonic device designers' perspective, FIONA is an out-of-the-box toolkit to handle all the high-level details.

Table 4. Specs of Various Thermo-Optic Devices

Device Types	Footprint (μm^2)	Power (mW)	Switching Voltage (V)	Switching Time (μs)
AB MZI [20]	~5000	12.7	~11.9	2.2
DCI MZI [21]	50×30	28	~2.8	2.16
PCN MZI [22]	150×30	0.16	~0.6	4.5
DC MRR [23]	51×17	21	~0.4	9
LN MRR [24]	400×400	14.9	~5	53
MI [25]	400×130	0.05	~6.5	780

* AB: adiabatic bend-based, DCI: direct carrier injection, PCN: photonic crystal nanobeam, LN: lithium niobate, MI: Michelson interferometer.

The software stacks are highly extendable and are supposed to support most scenarios of photonic-electronic collaborative simulation as long as new photonic models are correctly described and inserted. The hardware stacks, however, require targeted customization and optimization according to the interacting types of photonic devices. We conduct a specification survey of thermo-optic (TO) devices in Table 4. The upper bound of photonic TO devices is approximately 500kHz. Currently, the presented hardware template can fulfill the requirements of tuning TO devices. As we step further, electro-optic devices that work at the frequency of tens of gigahertz are beyond the capability of the current-version FIONA prototyping toolchain. The bottleneck lies in the data transfer and signal conversion between electrical and optical domains. We are expecting to use PCIe interfaces with more advanced DAC/ADC components to build up the next-generation FIONA hardware template.

5 Conclusion

We present the FIONA toolchain with *LightRocket* as a case study demonstrating the entire workflow of designing a Turing-complete photonic accelerator system. Compared to state-of-the-art commercial electronic accelerators that mainly adopt the systolic-array scheme, the photonic accelerators fully exploit the analogue computing paradigm and therefore are more promising but sophisticated. Any math-meaningful optical phenomena can serve operators. We anticipate that FIONA can bring up an ecosystem that all the photonic operators can build and share in a unified standard to enable fast migration and exploration for futuristic high-performance photonic computing.

References

[1] Jiaqi Gu, Zheng Zhao, Chenghao Feng, Zhoufeng Ying, Mingjie Liu, Ray T Chen, and David Z Pan. Toward hardware-efficient optical neural networks: Beyond fft architecture via joint learnability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(9):1796–1809, 2020.

[2] Kyle Shiflett, Avinash Karanth, Razvan Bunescu, and Ahmed Louri. Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 860–873. IEEE, 2021.

[3] Febin Sunny, Asif Mirza, Mahdi Nikdast, and Sudeep Pasricha. Crosslight: A cross-layer optimized silicon photonic neural network accelerator. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1069–1074. IEEE, 2021.

[4] Kyle Shiflett, Dylan Wright, Avinash Karanth, and Ahmed Louri. Pixel: Photonic neural network accelerator. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 474–487. IEEE, 2020.

[5] Yinyi Liu, Jiaxu Zhang, Jun Feng, Shixi Chen, and Jiang Xu. A reliability concern on photonic neural networks. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1059–1064. IEEE, 2022.

[6] Jiaqi Gu, Hanqing Zhu, Chenghao Feng, Zixuan Jiang, Ray Chen, and David Pan. L2ight: Enabling on-chip learning for optical neural networks via efficient in-situ subspace optimization. *Advances in Neural Information Processing Systems*, 34:8649–8661, 2021.

[7] Hui Zhang, Mile Gu, XD Jiang, Jayne Thompson, Hong Cai, S Paesani, R Santagati, A Laing, Y Zhang, MH Yung, et al. An optical neural chip for implementing complex-valued neural network. *Nature communications*, 12(1):457, 2021.

[8] Johannes Feldmann, Nathan Youngblood, Maxim Karpov, Helge Gehring, Xuan Li, Maik Stappers, Manuel Le Gallo, Xin Fu, Anton Lukashchuk, Arslan Sajid Raja, et al. Parallel convolutional processing using an integrated photonic tensor core. *Nature*, 589(7840):52–58, 2021.

[9] Farshid Ashtiani, Alexander J Geers, and Firooz Aflatouni. An on-chip photonic deep neural network for image classification. *Nature*, 606(7914):501–506, 2022.

[10] 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 *Proceedings of the 49th Annual Design Automation Conference*, pages 1216–1225, 2012.

[11] 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*, 4, 2016.

[12] Wilson Snyder. Verilator and systemperl. In *North American SystemC Users' Group, Design Automation Conference*, 2004.

[13] gdsfactory: a python library to design chip layout. [Online] Available at: *https://gdsfactory.github.io/gdsfactory/components.html*.

[14] Lumerical Inc. Lumerical: Photonic multiphysics simulation tools. [Online] Available at: *https://optics.ansys.com/*.

[15] 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 *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 769–774. IEEE, 2021.

[16] Alexander N Tait, Allie X Wu, Thomas Ferreira De Lima, Ellen Zhou, Bhavin J Shastri, Mitchell A Nahmias, and Paul R Prucnal. Microring weight banks. *IEEE Journal of Selected Topics in Quantum Electronics*, 22(6):312–325, 2016.

[17] 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.

[18] Spike: a RISC-V ISA simulator. [Online] Available at: *https://github.com/riscv-software-src/riscv-isa-sim*.

[19] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O Hero III, and Pramod K Varshney. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020.

[20] Michael R Watts, Jie Sun, Christopher DeRose, Douglas C Trotter, Ralph W Young, and Gregory N Nielson. Adiabatic thermo-optic mach–zehnder switch. *Optics letters*, 38(5):733–735, 2013.

[21] Manuel Mendez-Astudillo, Masaki Okamoto, Yoshiaki Ito, and Tomohiro Kita. Compact thermo-optic mzi switch in silicon-on-insulator using direct carrier injection. *Optics Express*, 27(2):899–906, 2019.

[22] Huanying Zhou, Ciyuan Qiu, Xinhong Jiang, Qingming Zhu, Yu He, Yong Zhang, Yikai Su, and Richard Soref. Compact, submilliwatt, 2× 2 silicon thermo-optic switch based on photonic crystal nanobeam cavities. *Photonics Research*, 5(2):108–112, 2017.

[23] Po Dong, Wei Qian, Hong Liang, Roshanak Shafiiha, Ning-Ning Feng, Dazeng Feng, Xuezhe Zheng, Ashok V Krishnamoorthy, and Mehdi Asghari. Low power and compact reconfigurable multiplexing devices based on silicon microring resonators. *Optics express*, 18(10):9852–9858, 2010.

[24] Xiaoyue Liu, Pan Ying, Xuming Zhong, Jian Xu, Ya Han, Siyuan Yu, and Xinlun Cai. Highly efficient thermo-optic tunable micro-ring resonator based on an lnoi platform. *Optics letters*, 45(22):6318–6321, 2020.

[25] Zeqin Lu, Kyle Murray, Hasitha Jayatilleka, and Lukas Chrostowski. Michelson interferometer thermo-optic switch on soi with a 50-μw power consumption. *IEEE Photonics Technology Letters*, 27(22):2319–2322, 2015.