LiteAIR5: A System-Level Framework for the Design and Modeling of AI-extended RISC-V Cores

Yimin Gao*, Sergiu Mosanu*, Mohammad Nazmus Sakib*, Vaibhav Verma*, Xinfei Guo†, Mircea Stan*
*Department of Electrical and Computer Engineering, University of Virginia
†University of Michigan – Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University
{yg9bq, sm7ed, ms3xy, vv8dn}@virginia.edu, xinfei.guo@sjtu.edu.cn, mircea@virginia.edu

2023 IEEE 36th International System-on-Chip Conference (SOCC) | 979-8-3503-0011-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/SOCC58585.2023.10257058

***Abstract*—The rapid evolution of machine learning applications along with the exponential growth in the Internet of Things (IoT) has driven a surge in demand for high-performance and energy-efficient hardware solutions for Edge AI applications. While traditional Edge AI hardware either falls short in performance or incurs massive power/area overhead, this paper presents a novel RISC-V processor core architecture with instruction set architecture (ISA) extensions that achieves up to 37x performance improvement for a General Matrix Multiply (GEMM) kernel compared to a baseline RISC-V core. Furthermore, this paper proposes an end-to-end framework named LiteAIR5 that encompasses system-level modeling of ISA-extended RISC-V cores, System-on-Chip Generation, Compiler Support, and FPGA emulation. This all-encompassing framework provides users with the ability to develop optimized AI hardware while also taking into account the effects on the entire system.**

***Index Terms*—AI hardware, FPGA emulation, ISA extensions, Neural networks, RISC-V**

I. Introduction and Background

AI and machine learning have significantly reshaped application landscape in recent years, providing practical solutions for real-world problems like image classification, natural language processing, and autonomous cars. However, conventional AI accelerators like Graphics Processing Units (GPUs) [1] and Google's Tensor Processing Units (TPUs) [2] were designed mainly for large-scale cloud data centers, limiting their deployment in edge computing where costs and energy efficiency are of top priority. Moreover, the exponential increase in data generated, transmitted, processed, and stored by the Internet of Things (IoT) and Artificial Intelligence of Things (AIoT) has posed significant challenges to conventional cloud computing paradigm in AI ecosystems due to high latency, limited bandwidth, and increased costs [3]. This has led to a growing interest in Edge AI, which deploys AI near data-generating devices, offering real-time response, enhanced data security, and improved power efficiency. However, current Edge AI computations are limited by the computing capabilities of all-purpose, generic microcontroller units (MCUs). Furthermore, current AI accelerators operate as decoupled standalone units that share resources with the CPU via a data bus, which is suboptimal for edge scenarios. Edge computing demands low latency, operates within constrained hardware budgets, and calls for more tightly integrated designs. Therefore, there is a need for hardware solutions that are Pareto optimal in terms of performance, power, and area for AI/ML applications at the edge.

A. The Need for AI Extended RISC-V Cores

To address these challenges, one promising solution is to enrich the CPU ISA with custom instructions that accelerate common operations in AI applications [4]. Similar to vector extensions, this approach integrates small in-pipeline AI units that allow for fine-grained offloading of AI tasks, resulting in a more efficient and effective solution for edge computing. Recently proposed AI-optimized scalable RISC-V processors [5], [6] offer energy-efficient edge hardware with custom AI instructions and tightly integrated AI accelerators within the processor pipeline. However, extending a RISC-V core requires concurrent modifications to both hardware and software components, necessitating hardware-software co-design to ensure synergy between the software and compiler with the current hardware. This co-design process also enables verification and evaluation of new instructions when extending a RISC-V processor and saves a significant amount of development time.

In this work, we leverage Application-specific instruction-set processors (ASIP) Designer [7] ISA definition to configure AI-extended RISC-V cores and the accompanying compiler support, together with LiteX [8] for System-on-Chip (SoC) integration and FPGA deployment to devise LiteAIR5 - a comprehensive framework for hardware and software co-design and evaluation. We briefly describe a spectrum of design strategies for Edge AI in Table I. In the following sections, we emphasize how the proposed technique compares against existing methods, augmenting architectural modeling and evaluation.

B. AI ISA Extensions with ASIP Tools

Prior research [5], [6] follows a hardware/software co-design methodology, utilizing a two-step compilation strategy that incorporates a modified TVM [14] as the front-end compiler and Synopsys ASIP Designer [7] as the back-end compiler. ASIP Designer is preferred due to its user-friendly hardware/software co-design framework, simplifying the extension of RISC-V cores with custom AI instructions and allowing for direct core benchmarking in simulation. However, the initial focus is on core development and does not include System-on-Chip integration and thus fails to address real-world constraints such as memory latency, bus connections, interrupt handling, and other system-level factors.

TABLE I
COMPARISON OF FRAMEWORKS FOR EXTENDING RISC-V PROCESSORS WITH AI ISA EXTENSIONS

Approach	HW/SW co-design	AI Extension(s)	Full System Integration	Flexibility and Features	Implementation Language
CFU Playground [9]	Limited to accelerator	Accelerator interface	LiteX SoC, firmware	Custom AI function unit & firmware, Fixed SoC, Limited benchmarking software, Limited FPGA boards	Verilog, Migen
PULP Platform [10]	Core and accelerator	Xpulp SIMD	ASIC	Performance limitation, Limited libraries	SystemVerilog, VHDL
Rocket Customization [11]	Split, accelerator	Possible	LiteX, other SoC	Complex Design, Agile HDL with Chisel, Flexible SoC architecture, Limited microarchitecture, memory configuration	Chisel (Scala) [12]
ASIP Designer [7]	Core and compiler	Custom Instructions	Core only	Flexible processor customization, Custom Instructions, Flexible libraries, Flexible ML models	nML
OpenASIP [13]	Limited	Custom Instructions	Core only	Flexible processor customization, Custom Instructions, Limited library	XML-based
LiteAIR5	Core, SoC, memory, compiler	Custom Instructions	Core, memory hierarchy, I/O, FPGA	Flexible processor customization, Custom Instructions, SoC architecture, ML models, Flexible bus connection, Cache configuration, FPGA board, Flexible number of cores, Flexible memory configuration, Flexible libraries, ML models	nML, Verilog, Migen

For instance, the RISC-V processor in ASIP Designer assumes a system without a memory hierarchy, always returning the requested data within a single clock cycle. The core also lacks cache integration and essential handshaking signals, such as “valid” and “acknowledgment” signals. Additionally, it does not provide sufficient stall signals to deactivate parts of the core during a memory miss. Lastly, it does not support System-on-Chip generation/integration or FPGA modeling. Such limitations can lead to an overly-optimistic estimation of the processors’ performance, resulting in sub-optimal designs in real-world systems. Furthermore, for benchmarking, prior research [5], [6], [15] employs cycle-accurate C++ / SystemC-based simulation, which, while faster than gate-level simulation, is considerably slower than a hardware modeling approach wherein the workload operates on an FPGA prototype of the System-on-Chip.

C. AI ISA Extensions with Soft Core Approach

Soft-core approaches like CFU Playground [9] and Rocket customization [11] provide another way to extend RISC-V cores for AI. While CFU Playground offers a user-friendly framework with System-on-Chip support for extending a RISC-V core for AI, it only allows for a restricted hardware/software co-design methodology with limited flexibility, as users can only design the processor in the provided region of the core and not have access to control the whole processor. Additionally, CFU Playground lacks support for front-end ML compilers such as TVM [14] and only supports evaluation with faked/canned ML inference. Rocket customization [11] presents a viable option for extending RISC-V cores for AI due to its extensive open-source FPGA and System-on-Chip framework support. Despite its advantages, Rocket customization presents a steep learning curve and requires extensive expertise in both hardware design and compiler knowledge. The soft-core approach lacks simplicity when adding new instructions, and implementing AI ISA extensions on these soft cores typically requires expertise in digital design and hardware description languages such as SystemVerilog, Chisel, SpinalHDL, and Migen.

D. AI ISA Extensions with ASIP Design and Soft Core Hybrid Approach

To design an optimal AI accelerator for Edge AI applications, it is crucial to have both a user-friendly hardware-software co-design framework that simplifies the process of extending RISC-V cores, and a user-friendly System-on-Chip support that adopts a soft-core approach. This allows for efficient modeling, performance evaluation, activity tracking, and design optimization as the core interacts with the system as a whole. Therefore, a system-level framework with an ASIP-based processor design and soft-core-based System-on-Chip integration is needed to enable a simplified and optimized process of extending RISC-V cores for Edge AI applications. However, as of now, there is no known framework that can fully harness the flexibility of ASIP Designer to extend the ISA of RISC-V cores and also provide complete system-level support through a soft-core approach.

To fill this research gap, we propose LiteAIR5. LiteAIR5 is an end-to-end system-level framework for designing and modeling AI-extended RISC-V cores that incorporates the user-friendly hardware/software co-design of ASIP Designer, soft-core-based system-on-chip integration, system-level modeling on FPGA, and provides custom RISC-V compiler support. This enables full-system design and modeling of AI-extended RISC-V cores combining the best of both worlds. We summarize the contributions of this paper as follows:

- We introduce a user-friendly framework for end-to-end AI-extended RISC-V core design and modeling, facilitating both hardware-software co-design and complete System-on-Chip architecture generation. The framework includes ASIP Designer, LiteX SoC builder framework, and a customized RISC-V GNU toolchain.
- To improve user-friendliness and flexibility, the proposed framework for AI-extended processors has been integrated into the LiteX framework. This integration simplifies interfacing, system creation, implementation of the target system on a chosen FPGA board, and expanding system-level design space explorations.

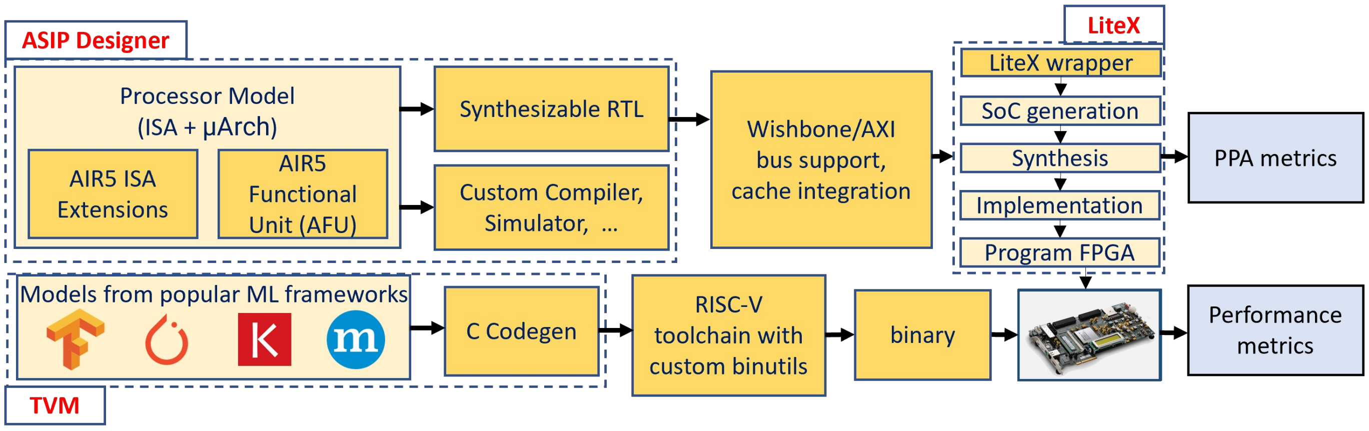

ASIP Designer
Processor Model (ISA + μArch)
AIR5 ISA Extensions
AIR5 Functional Unit (AFU)
Synthesizable RTL
Custom Compiler, Simulator, ...
Wishbone/AXI bus support, cache integration
LiteX
LiteX wrapper
SoC generation
Synthesis
Implementation
Program FPGA
PPA metrics
Models from popular ML frameworks
C Codegen
RISC-V toolchain with custom binutils
binary
Performance metrics
TVM

Fig. 1. Design Process of the proposed framework for system-level validation and evaluation of AI-argument RISC-V cores

- We develop AIR5, a 32-bit RISC-V processor designed with custom AI instructions and hot-swappable in-pipeline AI Functional Units. The inclusion of these specialized units yields a remarkable 37x improvement in performance for a General Matrix Multiply (GEMM) kernel, making AIR5 a highly scalable and efficient solution for Edge AI workloads.

II. System-level framework for modeling AI-extended RISC-V cores

In this section, we present a comprehensive overview of the design process of the proposed framework, LiteAIR5. The proposed framework incorporates ASIP cores into the LiteX SoC generator framework [8] and provides additional software and compiler support for further validation and evaluation of the AI-extended RISC-V cores. LiteAIR5 enables a synergistic hardware/software/ISA co-design to extend a RISC-V core with AI instructions. Additionally, we provide detailed descriptions of the proposed AIR5 processor architecture and the SoC architecture surrounding the AIR5 processor, which has been constructed using the proposed framework.

A. LiteAIR5 Design Flow

The design process of LiteAIR5 is illustrated in Figure 1. The ISA extensions and AI functional units are designed in ASIP Designer which generates the synthesizable Register Transfer Level (RTL) of the core and the complete software development kit including compiler, simulator, etc. To ensure compatibility of the generated RTL with LiteX SoC interconnections, Wishbone/AXI bus interface support and L1 cache integration are added to the RTL of the AI core if required. Next, a LiteX wrapper is created in Migen to connect the core to the system data bus. The design then undergoes a standard FPGA design flow including synthesis, implementation, and FPGA deployment. During the FPGA design flow, power, performance, and area metrics are gathered for the generated System-on-Chip.

To enable the evaluation of popular ML models on our proposed LiteAIR5 framework, we utilize the TVM open-source compiler as the front-end compiler, following a similar software flow to prior research [5], [6]. This allows us to compile Tensorflow, PyTorch, Keras, and MXNet models into custom C code. While previous studies [5], [6] used the custom compiler from ASIP Designer as the back-end tool for evaluation in the ASIP Designer simulation environment, for this work we chose the open-source RISC-V toolchain with custom *binutils* which is supported by the LiteX framework to compile the C program into binary with custom instructions.

B. Hardware-Software Co-design Flow

To design the ISA extensions and microarchitecture of the AI processor, we follow a similar approach in [5], [6] to utilize the hardware/software codesign from ASIP Designer. The processor architecture is described using a C-like high-level language called nML which describes both custom ISA extensions and pipeline structure. This language allows for the generation of the hardware RTL and the complete software stack, including the compiler, assembler, linker, simulator, debugger, and profiler. These software tools are compatible with any ISA and microarchitecture changes, making it easy to program and test applications for each processor version. The generated hardware and software also enable benchmark evaluations in the generated cycle-accurate simulator, providing a comprehensive evaluation of the designed processor.

To verify and evaluate the custom instructions of the AI processor, a C program must be compiled and executed before moving to full-system integration. ASIP Designer simplifies this process by generating software that is always compatible with the hardware. Compiling and executing a custom C test program can be achieved with a single click in the ASIP Designer interface. This ease of use ensures that the software can be efficiently generated and executed for each version of the processor with varying ISA and microarchitecture changes. By facilitating the software generation process, ASIP Designer streamlines the overall design and validation process for the AI processor.

C. AIR5: 32-bit RISC-V Processor with AI instructions

However, previous work on AI processors [5], [6] lacked support for Control and Status Register (CSR) instructions which are essential for full-system integration. To overcome this limitation, we present the design of AIR5, a 32-bit scalable RISC-V processor with support for both AI and CSR instructions, using the proposed LiteAIR5 framework. AIR5 integrates AI accelerators inside processor pipeline to enable real-time response and power-efficiency. With the hardware/software co-design methodology, these in-pipeline AI functional units are hot-swappable that can be easily switched on or off based on the application scenarios.

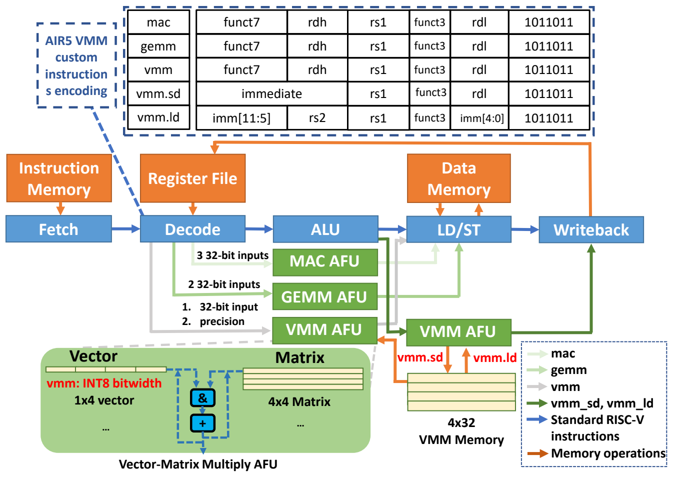
Figure 2 illustrates the detailed architecture of the AIR5 processor, including the encoding of custom instructions. All five instructions share the same opcode, 10110, and are decoded using funct7 and funct3 fields. The `mac` instruction that coupled with the MAC AI Functional Unit during the execute stage reads three 32-bit inputs from the register file (`a, b, c`), performs a multiply-and-accumulate operation, and writes the result (`axb+c`) back to the register file during the Write Back stage.

Similarly, the `gemm` instructions which couple with the GEMM AI functional unit during the execute stage read two 32-bit values from the register file. The unit interprets the 32-bit value as a 2x2 matrix with 8 bits per entry, performs a 2x2x2 matrix multiplication and stores the results in the register file during the Write Back stage.

The `vmm` instruction is used to perform Vector-Matrix Multiplication (VMM) and encodes the quantization bit-width in `funct7`. This instruction supports INT4 and INT8 quantization bit-widths. To perform VMM within the execute stage, an 8x32-bit VMM memory is used to store the matrix values. The VMM unit reads the precision from the decode stage, a 32-bit value from the register file and a Matrix from the VMM memory. It then performs Vector-Matrix multiplication during the execute stage based on the quantization bit-width. For instance, with an INT8 quantization bit-width, the 32-bit input is interpreted as a 1x4 vector with 4 bits per entry, and a 4x4 matrix is read from the VMM memory to perform the VMM operation. The result is subsequently passed through the memory stage and stored back in the register file during the Write Back stage. The final two instructions, `vmm.sd` and `vmm.ld` function as VMM load and store instructions. While they share similarities with the standard RISC-V load/store instructions, they differ in that they read from and write to the VMM memory instead of the main memory.

D. Full System Integration

While the ASIP Designer can generate the RTL of the AI processor we design, the basic RISC-V core's native interface from ASIP Designer lacks stall or handshaking signals, which means that it cannot halt instruction fetching in case of a cache miss in the instruction memory or pause the core's operation before the memory stage when a memory miss occurs in the data memory. To enable basic handshaking with the instruction



mac	funct7	rdh	rs1	funct3	rdl	1011011
gemm	funct7	rdh	rs1	funct3	rdl	1011011
vmm	funct7	rdh	rs1	funct3	rdl	1011011
vmm.sd	immediate		rs1	funct3	rdl	1011011
vmm.ld	imm[11:5]	rs2	rs1	funct3	imm[4:0]	1011011

Fig. 2. AIR5: 32-bit RISC-V processor with AI extensions

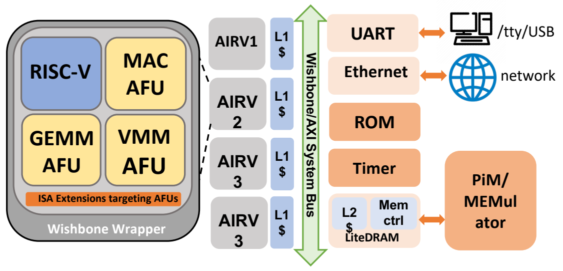


Fig. 3. LiteAIR5 System-on-Chip architecture

memory and data memory, a processor stall signal and a fetch stage stall signal are added to the core in ASIP Designer.

To work with a Wishbone system data bus in the Litex environment, a Wishbone Wrapper is built on top of the AIR5 processor in Verilog. On the LiteX side, a wrapper is built in Migen for interconnection between the core and the system bus. The next step involved generating a System-on-Chip (SoC) around the AIR5 cores as shown in Figure 3, by following the rest of the framework described in Figure 1.

The LiteAIR5 SoC, depicted in Figure 3 consists of the AIR5 processor integrated into a Wishbone wrapper, a UART module, an EtherNet module, Read-only memory (ROM), a DRAM controller called LiteDRAM which comes with a configurable L2 cache, a timer, and a memory module as either a traditional DDR3 memory or PiMulator, a flexible Processing-in-Memory Emulation Platform [16]. The UART module enables serial communication with the host to interact with the SoC and load applications into DRAM, while the Ethernet module enables parallel communication with the local network. On the memory side, the LiteDRAM memory controller makes it possible to access DRAM which could fit in much larger ML models. The system has software support with a configurable memory space with four regions: ROM, SRAM, main RAM, and CSR. The system also enables multi-core features where we can integrate multiple numbers of AIR5 processors into a system.

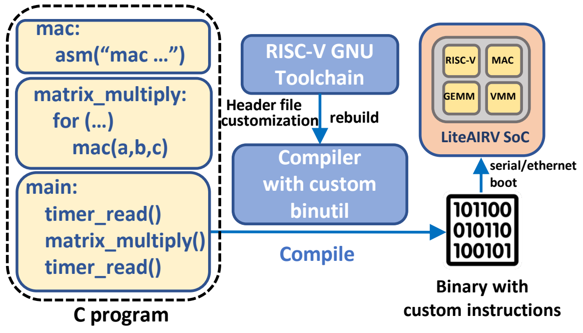


Fig. 4. Back-end compile methodology in LiteAIR5

E. *Software and Compiler support*

The back-end software methodology of LiteAIR5 is detailed in Figure 4, following the front-end compiling with TVM. LiteAIR5 customizes the *binutil* in a RISC-V GNU toolchain to compile C programs to binary files with custom AI instructions. To incorporate the custom AI instructions into the RISC-V GNU toolchain, we define the encoding of the custom instructions in the header files of the toolchain. Then, we rebuild the toolchain to support these instructions in the *binutil*. Since the customized *binutil* can only map custom assembly code to custom instructions instead of translating general C code to custom instruction itself, an extra step of writing C programs with incorporated inline assembly code is needed. For example, as shown in Figure 4, a C function with inputs and outputs incorporates the assembly code of a mac instruction. This function leverages the custom hardware unit in the AIR5 processor. In this example, the mac function is used to conduct a GEMM kernel. After the compilation, the resulting binary file is then sent to the LiteAIR5 SoC through serial communication.

III. Evaluations

The LiteAIR5 framework not only enables full System-on-Chip integration and provides end-to-end software support but also facilitates FPGA modeling. In our evaluation, we leveraged this framework to assess the 32-bit AIR5 processor's performance, power, and energy efficiency. Specifically, we ran C programs that performed matrix multiplication on matrices of different sizes (4x4, 8x8, 16x16) to obtain these metrics.

A. *Experimental Setup*

We considered four versions of the AIR5 processor to comprehensively evaluate its real-world capabilities: a plain RV32I RISC-V processor, a RISC-V processor with MAC extension (R5+MAC), a RISC-V processor with both MAC and GEMM extensions (R5+MAC+GEMM), and a RISC-V processor with MAC, GEMM, and VMM instructions (R5+MAC+GEMM+VMM). All AI functional units can be easily enabled or disabled, providing greater flexibility in the evaluation process and enabling more thorough analysis of the processor's capabilities.

The System-on-Chip surrounding the AIR5 core was built using LiteAIR5's design process and includes an AIR5 processor, a Wishbone system bus, a UART module, Read-only Memory, a Timer, a DRAM controller with an L2 cache, 2GB DDR3 memory. The integrated timer enables accurate measurement of real-world timing for specific code segments. We measured the matrix multiplication performance using the integrated timer by recording its values before and after the matrix multiplication function to capture the cycle count and elapsed time. We used the integrated UART module to transmit the program to the FPGA and the results from the FPGA board to the host, while the 2GB DDR3 memory expands the memory in addition to the limited SRAM on the FPGA, providing more memory space for larger ML models. We generated the bitstream for the SoC using LiteAIR5's design process and booted it on a Xilinx VC707 FPGA board.

To evaluate the performance of the AIR5 processor, we ran a GEMM kernel in C with various matrix sizes on the FPGA using the LiteAIR5 framework. To incorporate custom instructions in the C program, we followed the back-end software methodology illustrated in Figure 4. We inserted inline assembly code to define the custom instructions with inputs and outputs and passed them to C functions to perform dense matrix multiplication operations. We then defined a GEMM kernel in C with different matrix sizes using these custom functions. We compiled the test C program to binary using the RISC-V GNU toolchain with customized binutils and loaded it onto the 2GB DDR3 memory of the SoC on the FPGA for full-system evaluation.

B. *Experimental Results*

The AIR5 processor's real-world capabilities were comprehensively evaluated in LiteAIR5 using four different versions of the processor, each with varying AI functional units. The performance results were captured for each version, including cycle count, total time elapsed for the GEMM

TABLE II
PERFORMANCE IMPROVEMENT IN THE NUMBER OF CYCLES AND SPEEDUP OF VARIOUS VERSIONS OF THE AIR5 CORES RUNNING GEMM WITH DIFFERENT MATRIX TYPES COMPARED TO THE BASELINE RV32I RISC-V PROCESSOR

Processor	4x4	8x8	16x16
RISC-V	6203(1.0x)	47982(1.0x)	451578(1.0x)
R5+MAC	2975(2.08x)	21697(2.21x)	167772(2.69x)
R5+M+GEMM	1365(4.52x)	8869(5.41x)	66037 (6.84x)
R5+M+G+VMM	225(27.57x)	1647(29.13x)	12225(36.94x)

TABLE III
FPGA UTILIZATION OF THE AIR5 CORE WITH VARIOUS AI FUNCTIONAL UNITS SETUP

Processor	LUT	FF	DSP
RISC-V	11308	6514	6
R5+MAC	11766 (+4%)	6589 (+1%)	9
R5+M+GEMM	13228 (+16%)	6626 (+2%)	9
R5+M+G+VMM	17179 (+51%)	6905 (+6%)	9

TABLE IV
ENERGY COMPARISON WITH VARIOUS VERSIONS OF AIR5 PROCESSORS TO RV32I RISC-V PROCESSOR BASELINE RUNNING GEMM WITH DIFFERENT MATRIX TYPES (IN MICROJOULES)

Processor	**4x4**	**8x8**	**16x16**
RISC-V	42 (100%)	323 (100%)	2540 (100%)
R5+MAC	20 (47.6%)	146 (45.2%)	1123 (44.2%)
R5+M+GEMM	9 (21.4%)	60 (18.6%)	445 (17.5%)
R5+M+G+VMM	2 (4.7%)	11 (3.4%)	85 (3.3%)

kernel with different matrix size. Power and utilization results were also captured during the FPGA design flow of the LiteAIR5 process. The performance results show that AIR5 achieves up to 36.94x speedup compared to a baseline RV32I RISC-V processor. As the processor is scaled up with more AI functional units, the performance also increases, with the R5+MAC+GEMM+VMM version achieving the highest speedup for 16x16 matrix multiplication. Moreover, as the matrix size is increased from 4x4 to 8x8 and 16x16, AIR5 consistently outperforms the baseline processor, demonstrating its effectiveness in handling high computational requirements. These results validate the capabilities of AIR5 in a real-world System-on-Chip, taking into account memory hierarchy, cache latency, and data bus communication. The performance results are summarized in Table II.

Table III demonstrates that integrating AI units into the core increases its utilization, with the VMM unit utilizing the most hardware resources. However, as Table IV shows, integrating AI units significantly improves the performance of the GEMM kernel, resulting in a substantial decrease in total energy consumption compared to the baseline RV32I processor. Despite the slight increase in utilization and power caused by integrating AI units, the overall energy savings are significant, making them a valuable addition to the LiteAIR5 framework. As the core scales up, these improvements in energy efficiency will become increasingly crucial.

IV. CONCLUSIONS AND FUTURE WORK

To enable comprehensive hardware and software co-design, ISA extensions, and seamless integration of AI cores within an SoC, we utilize the power of ASIP Designer and LiteX to develop LiteAIR5, an end-to-end framework. This allows us to create a robust and efficient solution that can tackle complex AI workloads while being highly flexible and customizable. This approach provides a straightforward and efficient way to generate custom hardware solutions for Edge AI applications while offering software and compiler support for further validation and evaluation.

In the future, our goal is to make the LiteAIR5 platform fully open-source by employing open-source ASIP tools like OpenASIP instead of proprietary ASIP Designer. To achieve this, we plan to modify the Wishbone Wrapper and LiteX wrapper to support the back-end software in LiteAIR5, enabling seamless integration with the OpenASIP toolchain. This will allow for more accessible and customizable ISA extension and AI core integration within an SoC, promoting wider adoption of the platform.

ACKNOWLEDGEMENT

This work was funded in part by a grant from the SRC AI Hardware program as well as by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[2] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[3] J. Zhang and D. Tao, "Empowering things with intelligence: a survey of the progress, challenges, and opportunities in artificial intelligence of things," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 7789–7817, 2020.

[4] V. Verma, "AI-RISC: Scalable RISC-V Processor for IoT Edge AI Applications," *University of Virginia, Electrical Engineering - School of Engineering and Applied Science, PHD (Doctor of Philosophy)*, 2022. [Online]. Available: https://doi.org/10.18130/nj7t-7j93

[5] V. Verma, T. Tracy II, and M. R. Stan, "EXTREM-EDGE—EXtensions To RISC-V for Energy-efficient ML inference at the EDGE of IoT," *Sustainable Computing: Informatics and Systems*, vol. 35, p. 100742, 2022.

[6] V. Verma and M. R. Stan, "AI-PiM-Extending the RISC-V processor with Processing-in-Memory functional units for AI inference at the edge of IoT," *Frontiers in Electronics*, p. 22.

[7] "Synopsys ASIP Designer," https://www.synopsys.com/dw/ipdir.php?ds=asip-designer.

[8] F. Kermarrec *et al.*, "LiteX: an open-source SoC builder and library based on Migen Python DSL," in *OSDA/DATE*, 2019.

[9] S. Prakash, T. Callahan, J. Bushagour, C. Banbury, A. V. Green, P. Warden, T. Ansell, and V. J. Reddi, "Cfu playground: Full-stack open-source framework for tiny machine learning (tinyml) acceleration on FPGAs," *arXiv preprint arXiv:2201.01863*, 2022.

[10] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*. IEEE Computer Society, 2015, pp. 1–39.

[11] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, 2016.

[12] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a Scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[13] K. Hepola, J. Multanen, and P. Jääskeläinen, "OpenASIP 2.0: co-design toolset for RISC-V application-specific instruction-set processors," in *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2022, pp. 161–165.

[14] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," *arXiv preprint arXiv:1802.04799*, 2018.

[15] A. Roelke and M. R. Stan, "Risc5: Implementing the RISC-V ISA in gem5," in *First Workshop on Computer Architecture Research with RISC-V (CARRV)*, vol. 7, no. 17, 2017.

[16] S. Mosanu, M. N. Sakib, T. Tracy, E. Cukurtas, A. Ahmed, P. Ivanov, S. Khan, K. Skadron, and M. Stan, "PiMulator: A fast and flexible processing-in-memory emulation platform," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1473–1478.