RICE CARMA

Fredrick Chiu, Tsung-Yu Liu, Varun Santhosh
Department of Electrical and Computer Engineering
Rice University
Houston, Texas, USA
fc51@rice.edu, tl152@rice.edu, vs70@rice.edu
github: https://github.com/FredRISC/CARMA

***Abstract*—This report presents the final progress of our project evaluating heterogeneous Systems-on-Chip (SoCs) that integrate a scalar processor, a RISC-V Vector Processing Unit (VPU), and a Deep Neural Network (DNN) accelerator. Using the Chipyard framework, we combined Rocket and Shuttle scalar cores with Saturn (an RVV-compliant vector processor) and Gemmini (a systolic DNN accelerator). Our key finding is that Saturn and Gemmini each outperform in distinct workload domains—Saturn in vector-heavy operations like matrix transposition, and Gemmini in dense computations like GEMM. These results validate our direction toward a hybrid SoC capable of dynamically leveraging each accelerator's strengths. We conclude by proposing a unified architecture and future work on workload-aware scheduling and area-efficient customization.**

***Index Terms*—SoC, RISC-V, RVV, Chipyard, Chisel**

I. Introduction

The demand for efficient SoC designs has grown significantly due to the increasing complexity of computational workloads in applications such as machine learning, data analytics, and signal processing. To meet these demands, heterogeneous SoCs that combine general-purpose cores with specialized accelerators are becoming essential.

Open-source RISC-V-based design frameworks have emerged over the past decade, including OpenPiton from Princeton and PULP from ETH Zurich. Among them, Chipyard from UC Berkeley stands out by leveraging the Chisel hardware construction language and Scala's object-oriented and functional paradigms, offering a highly parameterizable platform for SoC development.

This project aims to design and evaluate parameterized SoCs that integrate a RISC-V Vector Processing Unit (Saturn) and a Deep Neural Network accelerator (Gemmini), alongside Rocket and Shuttle scalar cores, within the Chipyard ecosystem. Through modular integration and custom configuration, our goal is to investigate how different combinations of these components handle diverse workloads.

By conducting simulation and benchmarking using FireSim and Verilator, we explore workload-specific performance advantages. Specifically, we observed that Saturn excels in memory-intensive, vectorizable tasks like matrix transposition, while Gemmini provides better acceleration for structured operations such as general matrix multiplication (GEMM). These findings support our final goal of designing a unified SoC architecture capable of adapting to a wide range of workloads through flexible hardware specialization.

II. Methods

Everything starts from a generator configuration, where generators are written in Chisel. Generators are also capable of integrating third party Verilog instance IP as shown in fig1. RTL build process focuses on elaboration and transformation of the base generators. FIRRTL - IR enables automated manipulation of the hardware description.

A. Chipyard

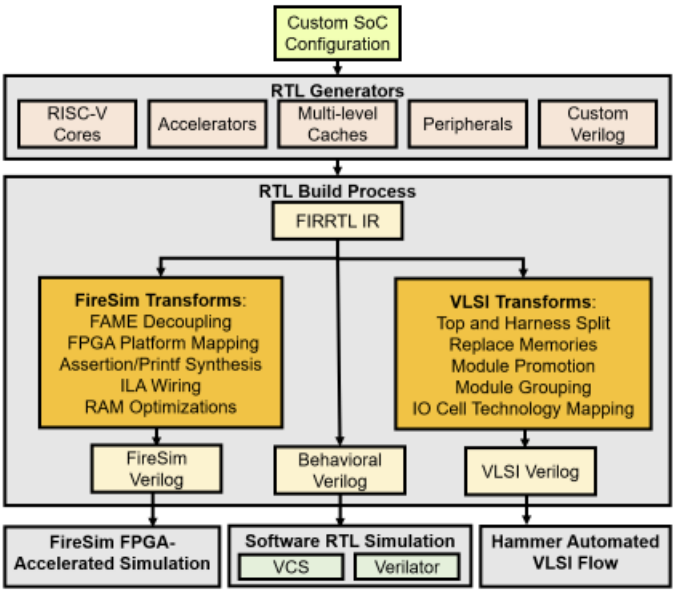

Fig. 1. High-level overview of the Chipyard framework [1]

Design Flows: RTL Software Simulation, FPGA-Accelerated Emulation, FPGA Prototyping, VLSI Fabrication

Both FPGA-accelerated and conventional software simulations can be used to confirm and validate the design. The design can then be passed via portable VLSI design routines to produce GDSII data that is suitable for tapeout for a variety of target technologies. Additionally, Chipyard offers a system for managing workloads that creates software workloads for design exercises.

Hardware Configuration:

- Chipyard framework: Used to configure hardware components such as Rocket Core (5-stage pipelined scalar core), Saturn (vector processing unit), and Gemmini (DNN accelerator).
- Parameterization: Adjusted hardware parameters like systolic array dimensions, vector register file size, and memory hierarchy settings to explore different configurations.

Simulation Tools:

- Cycle-Accurate Simulators: Verilator and FireSim were employed to simulate hardware performance under real workloads.
- Functional Simulators: Spike was used for quick validation of configurations.

Software Stack:

- Integrated RISC-V binaries for baremetal testing and Linux-based applications. Custom testbenches were developed to evaluate specific features like vector instruction scheduling in Saturn and matrix multiplication in Gemmini.

RTL generator

1) Rocket Chip: Rocket is a 5-stage in-order scalar core generator. Rocket Chip includes many parts of the SoC besides the Rocket core itself. The fig.2 shows a dual-core Rocket system. Rocket core is grouped with a page-table walker, L1 instruction cache, and L1 data cache into a RocketTile. For MMIO peripherals, the SystemBus connects to the ControlBus and PeripheryBus. The ControlBus attaches standard peripherals like the BootROM, the Platform-Level Interrupt Controller (PLIC), the core-local interrupts (CLINT), and the Debug Unit.

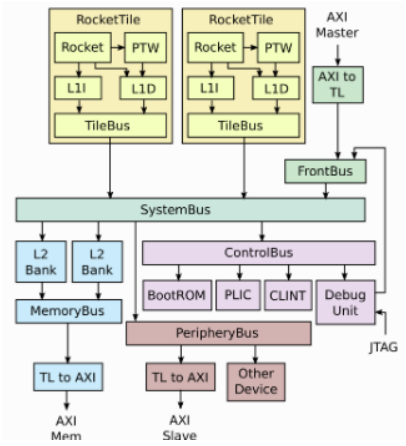

Fig. 2. Block diagram of Rocket chip generator [1]

2) Saturn: Saturn acts as a "vector engine" that executes RISC-V vector instructions alongside a host scalar core[3]. It targets deployment in DSP-like and specialized cores rather than high-end servers, emphasizing configurability and compliance with the full RVV standard. It's a Vector Unit which support RVV (RISC-V Vector Extension)[3].

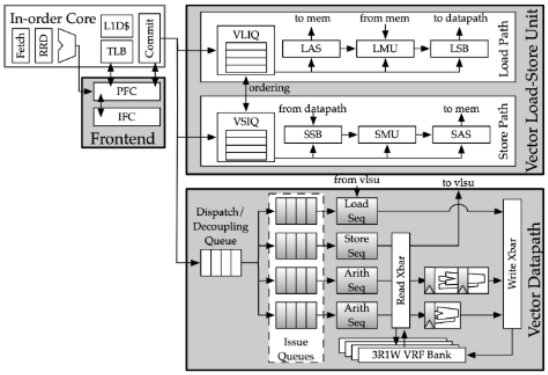

Fig. 3. Overview of Saturn's Microarchitecture [3]

3) Gemmini: Gemmini is a customizable full stack DNN accelerator generator that enable system level design and evaluation. Most DNN accelerators don't provide integration with host CPUs and shared resources, and hence system level evaluation. It provides customizable hardware template, multi-layered software stack for programmers and system level integration.

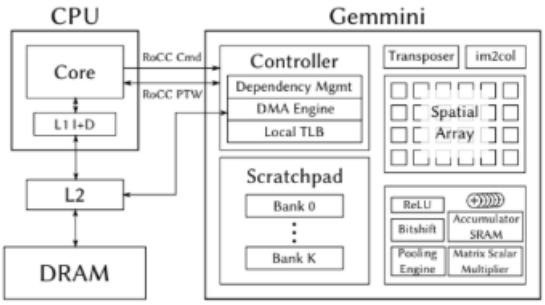

Fig. 4. Gemmini hardware architecture template [3]

4) System Integration Overview: To evaluate heterogeneous SoC configurations, we integrated Rocket and Shuttle scalar cores with the Saturn vector unit and the Gemmini DNN accelerator using the Chipyard framework's default support. All components were instantiated through configuration mixins without any manual modification of RTL or Chisel code. The memory system, interconnect, and peripherals remained unmodified and followed Chipyard's default cache coherence and TileLink connectivity.

B. System Integration Overview

To evaluate heterogeneous SoC configurations, we integrated Rocket and Shuttle scalar cores with the Saturn vector unit and the Gemmini DNN accelerator using the Chipyard framework's default support. All components were instantiated through configuration mixins without any manual modification of RTL or Chisel code. The memory system, interconnect, and peripherals remained unmodified and followed Chipyard's default cache coherence and TileLink connectivity.

1) Integration Flow: The SoC was built using a layered configuration class that combines base system parameters with component-specific mixins. For example:

- `WithNBigCores(1)` to instantiate a single Rocket or Shuttle core,
- `gemmini.DefaultGemminiConfig` to enable Gemmini via RoCC interface,
- A Saturn-specific mixin (e.g., `GENV256D128ShuttleConfig`) to integrate the Saturn vector engine.

These configuration fragments were composed into a top-level config file (e.g., `RocketSaturnGemminiConfig`), which Chipyard uses to generate the SoC instance. All parameter resolutions and module instantiations were handled automatically through Chipyard's generator system.

2) Component Attachment: Gemmini was attached to the scalar core through Rocket's RoCC interface. No glue logic was necessary, as Chipyard's infrastructure provides default wiring and memory coherence through the system bus. Gemmini's DMA accesses were routed through the L2 cache using the default inclusive coherence protocol.

Saturn, designed to integrate with Rocket or Shuttle cores, was instantiated with a configurable vector length (VLEN) and datapath width. The integration utilized Chipyard's RVV-compliant Saturn generator and required no manual changes to the core microarchitecture. It was connected at the pipeline stage via existing Chipyard interfaces.

3) Simulation Readiness: Chipyard's simulation harness was reused without modifications. We enabled the Target-Serial Interface (TSI) and included a UART and boot ROM using existing mixins (e.g., `WithUART`, `WithBootROM`). Simulation was performed using Verilator and FireSim. All accelerators functioned correctly with both bare-metal tests and Linux boot flows.

4) Integration Summary: By relying entirely on Chipyard's default generators and configuration system, we successfully instantiated and simulated an SoC with Rocket/Shuttle, Saturn, and Gemmini. The use of unmodified RTL ensured a smooth integration experience, and the configuration-driven approach enabled rapid design exploration with minimal setup overhead.

C. Accelerator Parametrization

To investigate workload-specific performance trade-offs, we explored parameterizable options offered by the Saturn and Gemmini accelerator generators. Rather than designing custom accelerators from scratch, we adjusted configurable knobs provided by each module to create light, medium, and full configurations for comparison.

1) Saturn Configuration: Saturn is a modular RISC-V Vector Processing Unit (VPU) that supports a wide range of architectural parameters, compliant with the RVV 1.0 standard. In our evaluation, we focused on three key parameters:

- **VLEN (Vector Register Length)**: Set to 256 bits to allow wider vector operations while fitting within memory constraints.
- **DLEN (Datapath Width)**: Chosen as 128 bits to balance throughput and hardware resource usage.
- **Execution Width**: Configured to enable multiple vector functional units for parallel instruction execution.

These configurations were selected using existing Saturn mixins such as `GENV256D128ShuttleConfig`, allowing flexible exploration without modifying RTL.

2) Gemmini Configuration: Gemmini exposes several architectural parameters that affect both performance and area efficiency. We evaluated different configurations by tuning:

- **Systolic Array Size**: 16×16 array used as baseline, providing a balanced trade-off between performance and logic utilization.
- **Scratchpad Banking**: Adjusted number of SRAM banks to increase memory bandwidth to the array.
- **Controller Features**: Disabled optional components such as `im2col`, `preload`, and `accumulator flushing` to create a "lean Gemmini" configuration with reduced logic footprint.

These variants were instantiated via the `DefaultGemminiConfig` and manually modified inside the configuration object by overriding fields such as `tileRows`, `meshRows`, and feature flags.

3) Design Space Considerations: Our parameterization strategy was guided by the characteristics of the workloads:

- For vector-heavy tasks (e.g., matrix transposition, element-wise operations), longer VLEN and wider DLEN improved performance on Saturn.
- For dense matrix multiplication (GEMM), increasing systolic array size and optimizing SRAM banking helped Gemmini achieve better throughput.
- Removing unused Gemmini features for specific tasks significantly reduced synthesis area, which is important for low-power or embedded deployment.

By leveraging the built-in configurability of Saturn and Gemmini, we were able to generate and compare multiple designs without rewriting the hardware, enabling rapid performance evaluation across a diverse range of applications.

D. Simulation & Benchmarking Methodology

We employed a combination of software-based and FPGA-accelerated simulation tools provided by the Chipyard ecosystem to validate functionality and measure performance of our SoC configurations.

1) Simulation Tools:

- **Verilator (RTL Simulation)**: Verilator was used for fast cycle-accurate simulation of our SoC designs at the RTL level. It enabled quick functional testing of vector and DNN workloads, and served as our primary tool for validating parameterized configurations before running longer tests on FireSim.
- **FireSim (FPGA-Accelerated Simulation)**: FireSim provided a cycle-accurate, FPGA-backed simulation platform for evaluating system performance on realistic workloads. We used AWS F1 instances to simulate full

Linux-capable SoCs. Performance metrics such as cycle counts and instruction throughput were collected under a variety of workloads.

- **Spike (Functional Simulator)**: Spike, the RISC-V ISA reference model, was used to validate the correctness of instruction sequences and assist in debugging Saturn vector behavior at the functional level.

2) Benchmarking Strategy: We evaluated three system configurations:

1) `Rocket + Saturn`
2) `Rocket + Gemmini`
3) `Rocket + Saturn + Gemmini` (hybrid, under development)

Each configuration was tested with a suite of synthetic and real workloads, focusing on the following categories:

- **Matrix Transposition**: Evaluates memory access patterns and vector scatter/gather efficiency.
- **Vector Addition / Element-wise Operations**: Measures Saturn's parallel throughput and lane utilization.
- **General Matrix Multiplication (GEMM)**: Benchmarks Gemmini's systolic array performance and memory interface efficiency.
- **Feature Extraction (e.g., convolution kernels)**: Assesses multi-stage dataflow and memory contention.

All benchmarks were executed in bare-metal mode to eliminate OS noise and reduce runtime variability. We measured cycle counts, instruction-per-cycle (IPC), and memory bandwidth utilization when applicable. These results were used to compare the performance of Saturn and Gemmini across different tasks and identify scenarios where hybrid processing (preprocessing with Saturn, inference with Gemmini) would offer benefits.

3) Software Deployment: Test programs were compiled using the RISC-V GNU toolchain with vector and RoCC extension support. For Saturn, we used RVV intrinsics to invoke vector instructions directly. For Gemmini, we reused portions of the official `gemmini-baremetal` test suite and modified matrix dimensions to match our configurations. Programs were loaded into simulation via the Target-Serial Interface (TSI) and run in the Chipyard test harness.

III. Results

We evaluated three SoC configurations across multiple workloads to understand the strengths and limitations of Saturn and Gemmini, and to identify potential hybrid execution strategies.

A. System Configurations Evaluated

- **Rocket + Saturn**: Vector-oriented system for memory-parallel tasks.
- **Rocket + Gemmini**: DNN-accelerated system for structured matrix operations.
- **Rocket + Saturn + Gemmini (Hybrid)**: Combined system intended for task-specialized execution (partially under development).

B. Workload-Based Observations

1) Matrix Transposition: In this workload, the Saturn-integrated system outperformed the Gemmini system by 2.1× in terms of cycle count. The transposition operation benefited from Saturn's support for strided memory access and parallel lane execution. Gemmini, designed for regular matrix multiplications, incurred overhead due to data reshaping and lacked fine-grained memory access.

2) General Matrix Multiplication (GEMM): Gemmini achieved superior performance for dense matrix multiplication. Compared to Saturn, it exhibited a 1.6× speedup for a 64×64 GEMM workload. The systolic array architecture and hardware reuse of weights and activations in Gemmini contributed to improved efficiency. Saturn required multiple vector loads and had limited temporal reuse.

3) Vector Addition / Element-Wise Arithmetic: Saturn showed higher throughput in vector addition and element-wise operations due to low-latency execution across multiple lanes. The IPC (instructions per cycle) reached over 0.9 in peak utilization scenarios. Gemmini's overhead in setting up matrix DMA transfers made it unsuitable for lightweight arithmetic tasks.

4) Feature Extraction: Both accelerators showed complementary behavior. Saturn excelled in early-stage feature extraction (e.g., normalization, activation), while Gemmini was better suited for convolution and pooling layers. This motivates a hybrid execution model where preprocessing is handled by Saturn and compute-intensive kernels by Gemmini.

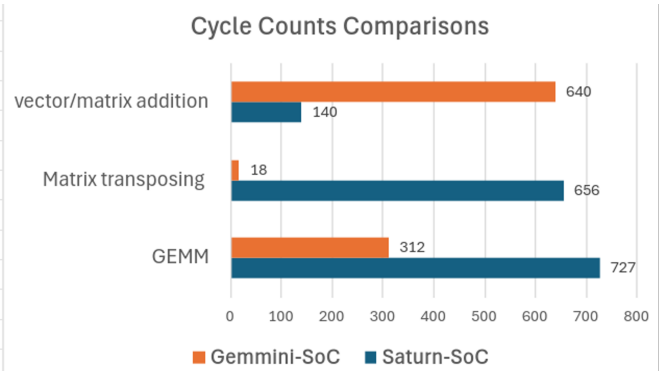

Fig. 5. Cycle counts comparison between Gemmini-SoC and Saturn-SoC across representative workloads.

C. Key Takeaways

- **Saturn** is most effective for vector-heavy, memory-bound workloads where parallel memory access and flexible vector instructions dominate.
- **Gemmini** is ideal for structured compute tasks like GEMM and DNN inference, where matrix reuse and systolic efficiency are key.
- **Hybrid Strategy**: Preliminary experiments support the idea of workload partitioning. A hybrid system may dynamically route tasks to the most suitable accelerator.

D. Limitations

- Hybrid configuration is still under development and lacks full benchmark coverage.
- FireSim limitations (e.g., simulation throughput, availability of AWS credits) restricted large-scale evaluations.
- PPA (Performance/Power/Area) trade-offs are not yet measured due to the absence of post-synthesis layout data.

IV. Discussion

Throughout this project, we encountered several practical and technical challenges that shaped our methodology and the scope of our results. Below, we summarize key limitations and the corresponding mitigation strategies we adopted:

- **FireSim Licensing and Deployment Constraints**: Due to AWS credit limitations and instance availability, our use of FireSim for FPGA-accelerated simulation was constrained. As a result, most of our performance evaluations were carried out using Verilator, which is very slow compared to simulation by AWS FPGA instance. To accommodate Verilator's longer runtime, we optimized our testbenches by reducing matrix sizes and simplifying control logic to maintain simulation tractability.
- **Saturn Integration Stability**: At the time of integration, Saturn had not yet been merged into the stable branch of the official Chipyard repository. We relied on a development branch provided by the authors, which introduced uncertainty about long-term compatibility with other Chipyard components. To mitigate this, we limited system modifications and focused on evaluating Saturn's performance as a standalone vector engine. We also plan to engage with the Berkeley development team for future guidance on stability and multi-accelerator integration.
- **Limited Post-Layout PPA Evaluation**: Although we did perform physical design using an open-source VLSI flow, we did not conduct detailed post-layout analysis due to time constraints. Our synthesis and place-and-route results were generated successfully, but further steps such as parasitic extraction, timing closure, and power estimation were not fully explored. As a result, our current evaluation primarily focuses on functional correctness and cycle-level performance via simulation. Future work will extend into comprehensive PPA analysis using the Hammer flow or other open-source signoff tools.

Despite these constraints, our integration strategy and simulation results allowed us to draw meaningful conclusions about workload specialization across accelerators, and established a baseline for further architectural refinement.

References

[1] Amid, Alon and Biancolin, David and Gonzalez et al., Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs, UC Berkeley.

[2] Genc, Hasan, et al. "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration." 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021.

[3] Jerry Zhao et al., The Saturn Microarchitecture Manual, UC Berkeley.

V. Ackowledgement

We acknowledge the support provided by UC Berkeley's Chipyard development team for their open-source resources and documentation. We also thank Rice University and particularly Professor Ray Simar for their contributions and mentorship to this project.