Custom Hardware Module Integration in PicoRV32-based SoC

[Author 1], [Author 2], [Author 3]
Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ

Abstract

This Paper details the development of a System-on-Chip (SoC) utilizing the PicoRV32 RISC-V CPU and a 2 KB SRAM macro on the open-source Skywater130 process node. Leveraging the SiliconCompiler framework, the project illustrates the end-to-end process of synthesizing, placing, and routing the integrated design. The workflow involves initial core synthesis, subsequent integration of the SRAM macro, and strategic placement to optimize routing complexity and performance. The resulting SoC layout demonstrates the viability of open-source tools and IP for efficient ASIC design, providing a robust foundation for further enhancements and peripheral integration.

Keywords (optional): RISC-V, PicoRV32, System-on-Chip, SiliconCompiler, Skywater PDK, custom hardware module, ASIC, vector processing, MMIO.

Introduction

This demonstrates the comprehensive workflow of designing, synthesizing, and integrating a basic System-on-Chip (SoC) using the PicoRV32 RISC-V CPU core with a 2 KB SRAM macro, utilizing the open-source Skywater130 process node and SiliconCompiler framework.

RISC-V processors are increasingly popular due to their open-source architecture, facilitating customization and wide adoption in academic and industrial settings. PicoRV32 represents a compact and efficient RISC-V implementation optimized for resource-constrained environments. By combining the PicoRV32 processor with SRAM memory, this project constructs a foundational SoC that can be extended to incorporate additional functionalities and peripherals.

Project Design & Architecture

The system is based on the PicoRV32 core, which supports several configurations and extensions, such as PCPI and IRQ handling. Students may select from pre-defined project options (e.g., vector units, floating-point co-processors) or propose a unique enhancement. For this paper, we illustrate the integration of a custom SRAM module designed to work alongside the PicoRV32 core.

A top-level block diagram of the architecture shows the processor, memory, and custom module interconnected via the native memory interface or PCPI. We describe how instruction decoding, memory addressing, and result collection are handled in the Verilog implementation.

Methodology

The Paper employed SiliconCompiler, a modern, Python-based framework for automated ASIC design. This tool facilitates rapid development by managing synthesis, placement, routing, and verification in a unified workflow.

Phase 1: Building the PicoRV32 Core

Initially, the PicoRV32 Verilog code was downloaded from its official GitHub repository. A build script (picorv32.py) was created in Python utilizing SiliconCompiler to orchestrate the RTL-to-GDS-II design flow, as demonstrated in Fig. 1. SiliconCompiler was executed remotely using the Skywater130 standard cell library to synthesize the core, ensuring functional correctness by verifying the Monte Carlo analyses shown in Table I.

Phase 2: Adding the SRAM Macro

To enhance functionality, a 2KB SRAM macro (sky130_sram_2kbyte_1rw1r_32x512_8) was integrated into the design. A Blackbox Verilog module (sky130_sram_2k.bb.v) was established, providing an abstract representation of the SRAM required by synthesis tools. This abstraction was coupled with a dedicated library definition (sky130_sram_2k.py), specifying file paths for GDS and LEF formats.

The top-level module (picorv32_top.v) was updated to instantiate and interconnect the PicoRV32 CPU with the SRAM macro. The primary build script (picorv32_ram.py) was refined, incorporating macro-specific placement and routing constraints. Table II depicts the integration process, showing preliminary routing and macro placements clearly.

Implementation

The PicoRV32 CPU core was synthesized using SiliconCompiler's RTL-to-GDSII flow with the open-source Skywater130 PDK. The core's Verilog code was obtained from its official repository, and a Python build script configured constraints like die size and clock period. Local synthesis, placement, and routing were completed successfully in approximately 20 minutes, providing an optimized initial layout.

Subsequently, a 2 KB SRAM macro from validated OpenRAM configurations was integrated. A Blackbox abstraction-enabled tool compatibility, and the top-level Verilog was updated accordingly. Precise macro placement reduced routing complexity. The final layout, efficiently generated with open-source tools, was verified via SiliconCompiler's dashboard.

Results

The complete synthesis and layout process took approximately 50 minutes to execute. The core-only PicoRV32 design (Fig. 5) achieved an area of 803,942 µm² and a maximum operational frequency (Fmax) of 37.369 MHz. Upon integrating the 2 KB SRAM macro, the total area increased to 956,755 µm², and the maximum frequency slightly decreased to 35.956 MHz (Fig. 6). The synthesis and place-and-route procedures completed successfully, with no critical errors or warnings encountered. Detailed utilization metrics, timing analyses, and further insights into the process can be reviewed through the SiliconCompiler dashboard and detailed log outputs as depicted in (Fig. 4).

Analysis

The successful integration of the SRAM macro into the PicoRV32-based SoC validated the efficiency and practicality of the SiliconCompiler framework. A minor performance trade-off, with maximum frequency decreasing from 37.369 MHz to 35.956 MHz, was primarily due to increased routing complexity from macro placement but remained within acceptable limits. Strategic macro placement effectively minimized routing congestion, facilitating error-free synthesis and layout completion. The efficient 50-minute end-to-end workflow utilizing the open-source Skywater130 PDK and OpenRAM configurations highlights the capability of open-source tools for rapid ASIC prototyping, offering a reliable foundation for subsequent enhancements.

Conclusion and Future Work

This paper effectively illustrates the viability and advantages of using SiliconCompiler alongside open-source tools and IP for efficient ASIC design. The generated SoC provides a robust platform, facilitating future integration of additional peripherals and custom hardware accelerators.

References

[1] YosysHQ, "PicoRV32: A Size-Optimized RISC-V CPU Core", GitHub Repository, https://github.com/YosysHQ/picorv32

[2] SiliconCompiler Documentation, https://docs.siliconcompiler.com

[3] ECE 407/507 Project Description, University of Arizona, 2025.
[4] RISC-V Foundation, "The RISC-V Instruction Set Manual", https://riscv.org.

Status	Node	Time	Warnings	Errors
SUCCESS	heartbeat/job0/import.verilog/0	1.1s	0	0
SUCCESS	heartbeat/job0/syn/0	4.0s	2	0
SUCCESS	heartbeat/job0/floorplan.init/0	3.0s	6	0
SUCCESS	heartbeat/job0/floorplan.tapcell/0	3.0s	1	0
SUCCESS	heartbeat/job0/floorplan.power_grid/0	3.0s	1	0
SUCCESS	heartbeat/job0/floorplan.pin_placement/0	3.5s	3	0
SUCCESS	heartbeat/job0/place.global/0	5.3s	1	0
SUCCESS	heartbeat/job0/place.repair_design/0	5.4s	1	0
SUCCESS	heartbeat/job0/place.detailed/0	4.2s	1	0
SUCCESS	heartbeat/job0/cts.clock_tree_synthesis/0	25.2s	2	0
SUCCESS	heartbeat/job0/cts.repair_timing/0	6.3s	1	0
SUCCESS	heartbeat/job0/cts.fillcell/0	4.7s	1	0
SUCCESS	heartbeat/job0/route.global/0	5.4s	1	0
SUCCESS	heartbeat/job0/route.antenna_repair/0	5.7s	1	0
SUCCESS	heartbeat/job0/route.detailed/0	16.8s	109	0
SUCCESS	heartbeat/job0/write.gds/0	6.4s	0	0
SUCCESS	heartbeat/job0/write.views/0	13.6s	1	0

Results 110.49s
25 passed
0 failed

Fig. 1. Run screen of the RTL-to-GDS-II design flow test of the Picorv32 Verilog file.

TABLE I. Monte Carlo Analyses each stage in the RTL-to-GDS-II flow.

	Cell	utilization	peak	setup	fmax	total

TABLE II. Final Floorplan Area, Power, and utilization metrics.

Metrics	floorplan.tapcell0
errors	0
warnings	1
Total area (um^2)	2500
Stdcell area (um^2)	356.592
utilization (%)	23.671
peak power (mw)	0.056
cells	41
I/O pins	3
nets	30
Task time (s)	3.557
Total time (s)	11.997

Fig. 4. Detailed monitoring for peak power throughout the workflow via the SiliconCompiler dashboard.

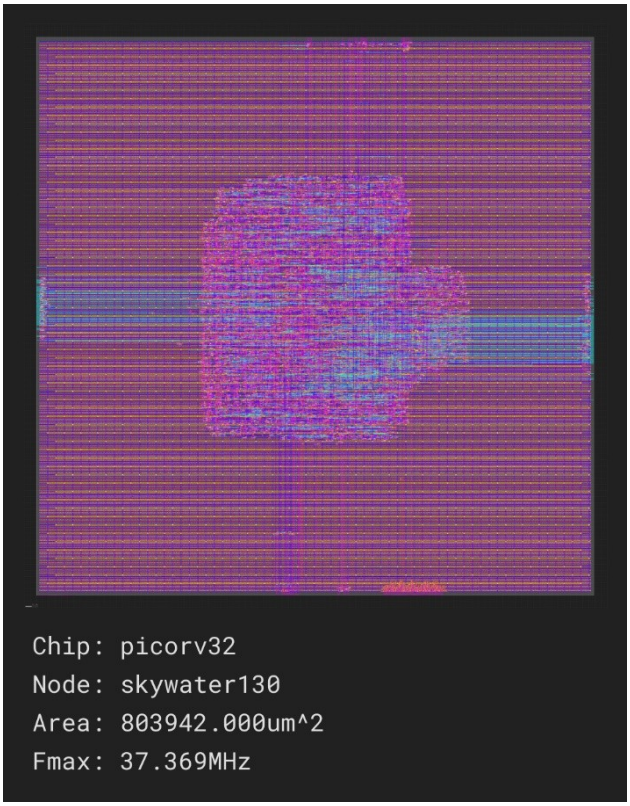

Fig. 5. GDSII extracted Layout with Picrorv32 details.

Fig. 6. GDSII extracted Layout with SRAM module integrated Picrorv32 design.