Wizards of the Core : Optimizing the CV32A6 Frequency

Oleg Paillot, Elliot Burns, Guilhem Chevallier
Université de Toulouse
Toulouse, France

***Abstract*—This paper presents a methodical approach to optimizing the CV32A6 RISC-V core for enhanced timing performance in an FPGA context. The CV32A6 is a 32-bit in-order, six-stage pipelined processor derived from the open-source CVA6 core and designed for embedded applications. Within the framework of the 2024–2025 RISC-V optimization contest, we aimed to increase the processor's maximum clock frequency while maintaining functional correctness and minimal performance degradation. By analyzing timing reports from Vivado, we identified critical path bottlenecks in the Frontend, Issue, and Execution stages, as well as in the cache subsystem.**

To address these issues, we applied a combination of hardware-oriented techniques, including register replication to reduce fanout, pipelining to break long combinational paths, and logic simplification. Additionally, we tried replacing the original L1 cache with a more timing-efficient HPDcache module, thus backporting it from the industrial CVA6 repo. These optimizations resulted in a 25% increase in operating frequency, from 40 MHz to 50 MHz, on a Zybo Z7-20 FPGA board.

Benchmark evaluations using CoreMark and MNIST confirmed that the performance impact remained within acceptable bounds, with a CoreMark/MHz drop of less than 3%. All valid modifications were tested through functional simulation and regression testing. This work demonstrates that targeted RTL-level interventions can yield substantial timing improvements without compromising performance or resource compliance.

***Index Terms*—RISC-V, CV32A6, FPGA, timing optimization, pipelining, register replication**

I. Introduction

The increasing adoption of open-source hardware, particularly RISC-V architectures, has driven interest in optimizing processor cores for performance and energy efficiency. In this context, the CV32A6 processor—derived from the CVA6 core—is a 32-bit in-order, six-stage pipelined RISC-V processor. It is the subject of a design competition co-organized by Thales, the SOC2 research group of the CNRS, and the CNFM. The main goal of this contest is to increase the maximum operating frequency of the CV32A6 processor while minimizing performance degradation, particularly for the CoreMark and MNIST benchmarks.

The challenge lies in identifying architectural or implementation-related bottlenecks that prevent the core from reaching higher frequencies. Participants are expected to work within strict constraints: compatibility with the RV32IM_Zicsr instruction set, no modifications to software or benchmark binaries, and deployment on a Zybo Z7-20 FPGA using Vivado 2024.1 and Questa 2022.2_3. Additionally, the improved design must maintain a CoreMark/MHz performance drop below 10% and stay within FPGA resource limits.

This paper details the optimization methodology, critical path analysis, and implementation strategies—such as pipelining, register replication, and cache replacement—that were applied to improve the frequency of the CV32A6 core from 40 MHz to 50 MHz on FPGA, while ensuring functional correctness through extensive simulation and benchmarking.

II. Background

The CVA6 processor (formerly known as Ariane) is a 64-bit open-source, in-order RISC-V core developed by the OpenHW Group. It is designed with a modular and configurable pipeline architecture, intended for academic, research, and industry use. The CV32A6 is a 32-bit variant derived from CVA6, optimized for embedded systems and FPGA implementation, and is the focus of our optimization effort.

The CV32A6 pipeline consists of six stages:

- **Program Counter (PC) Generation:** Computes the address of the next instruction and includes a simple branch prediction mechanism.
- **Fetch:** Accesses the instruction cache (I-Cache) using the computed PC and retrieves the instruction.
- **Decode:** Performs instruction decoding, immediate generation, and decompression (for compressed RISC-V instructions).
- **Issue:** Selects instructions from the queue and dispatches them to appropriate execution units based on their operation type.
- **Execute:** Executes arithmetic, memory, branch, or CSR operations using the relevant functional units (ALU, LSU, MUL/DIV, PMP).
- **Commit:** Writes the final results back to the register file and completes the instruction lifecycle.

Each stage is implemented using independent SystemVerilog modules communicating through handshake signals. The processor supports the RV32IM_Zicsr instruction set and is designed for high configurability and integration in SoC platforms.

For timing and synthesis analysis, we used the Xilinx Vivado Design Suite 2024.1. Vivado provides detailed static timing analysis (STA) reports including slack calculations, worst negative slack (WNS), total negative slack (TNS), and path analysis—key metrics for determining frequency limitations.

Functional verification was performed using Siemens Questa 2022.2_3 simulator, allowing cycle-accurate simulation of the entire processor core. To assess the performance impact of architectural changes, we used two benchmarks:

- **CoreMark:** A synthetic benchmark representative of general-purpose embedded workloads.
- **MNIST:** A machine learning inference benchmark that evaluates memory throughput and compute parallelism.

The target hardware for this project is the Digilent Zybo Z7-20 development board, which integrates a Xilinx Zynq-7000 series FPGA. This board imposes practical constraints on resource usage (LUTs, flip-flops, BRAMs), making timing and area optimization a balancing act.

Figure 1 illustrates the baseline CV32A6 pipeline used as the foundation for our optimization work.

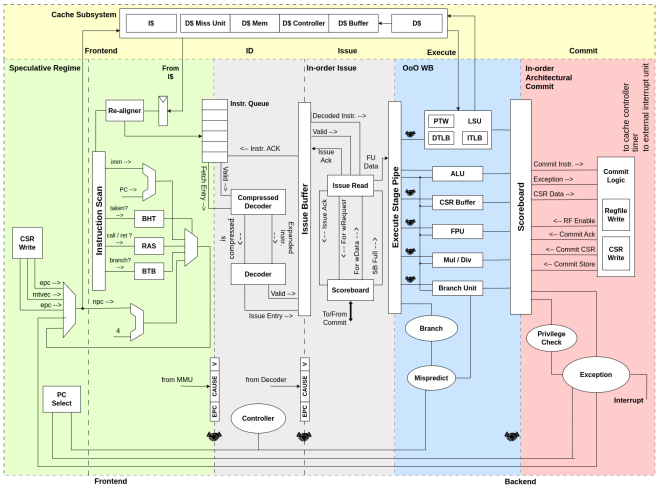

Fig. 1. Simplified CV32A6 pipeline structure. Source: OpenHW Group

In summary, improving the CV32A6's clock frequency required a deep understanding of how critical timing paths propagate through this pipeline and how each stage contributes to overall latency. This knowledge was the basis for all subsequent design optimizations.

III. Methodology

To increase the maximum clock frequency of the CV32A6 core, our methodology centered on identifying and resolving timing bottlenecks through detailed static timing analysis and structural optimization. The primary metric used for evaluation was the *slack*, which represents the timing margin before setup or hold violations occur on critical paths. A negative slack indicates that signals arrive too late relative to the clock edge, preventing the circuit from operating at the intended frequency.

Vivado's static timing analysis provides two critical indicators: Worst Negative Slack (WNS) and Total Negative Slack (TNS). WNS identifies the most limiting path, while TNS aggregates the severity and number of violating paths. Figure 2 illustrates the concept of setup slack and its influence on circuit stability.

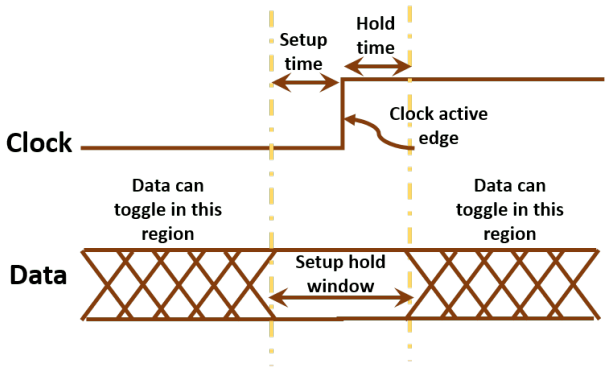

Fig. 2. Timing window for setup slack: data must arrive before the setup window closes.

Each pipeline stage was synthesized in isolation to expose its individual timing limitations. Table I summarizes the post-synthesis maximum frequencies of each stage before optimization.

TABLE I
Initial Stage-Level Frequency Limits (Post-Synthesis)

Pipeline Stage	Max Frequency (MHz)
Frontend	83
Decode	333
Issue	83
Execution	94
Commit	329
Cache	80

The analysis revealed that the Frontend, Issue, Execution stages, and Cache were the most limiting, typically due to long combinational paths or excessive fanout. We adopted a cycle of analysis and refinement to incrementally improve these stages.

To address the identified bottlenecks, we employed the following key techniques:

- **Register Replication:** High fanout, particularly from instruction alignment logic, led to routing delays and placement congestion. By replicating registers at key fanout points (e.g., outputs of the instruction realignment module), we distributed signal loads and improved routing efficiency.
- **Pipelining:** Long combinational paths in modules such as the ALU and Issue logic were broken into multiple pipeline stages using intermediate registers. While this increased latency by a cycle in certain paths, it dramatically improved timing closure and supported higher clock rates.
- **Logic Optimization:** In several modules, especially within the execution stage, complex logic expressions were refactored and simplified to reduce gate depth and routing length. This included flattening nested conditionals and removing redundant operations.
- **Tool-Assisted Optimization:** We leveraged Vivado's `opt_design` (post-synthesis logic optimization) and `phys_opt_design` (post-placement routing-aware op-

timization) commands with the `ExploreArea` strategy. These automated passes applied gate-level transformations and register balancing to improve slack without impacting logic functionality.

In addition to these structural improvements, we tried replacing the original L1 cache with the more advanced *HPDcache*. Even if the latter is currently included in more recent CVA6 versions, we wanted to try to backport it to the contest's version of the processor as cache performance played a huge part regarding frequency optimization.

All changes were iteratively validated through behavioral simulation using Questa and functional tests on the FPGA using CoreMark and MNIST binaries. This iterative loop of synthesis, analysis, modification, and validation ensured stability and measurable performance gains at each step.

IV. Implementation

After identifying the most frequency-limiting pipeline stages—Frontend, Issue, Execution, and Cache—we applied targeted and stage-specific optimizations to incrementally raise the operating frequency of the CV32A6 core. Each modification was carefully integrated, tested, and validated through simulation and synthesis. Below we detail the implementation for each major stage.

A. Frontend Stage

The Frontend stage includes the PC generation unit and the instruction fetch logic. Our timing analysis showed that the critical path extended from the instruction cache, through the instruction realignment logic, to the decode interface. This path suffered from a high fanout of approximately 200, largely due to wide signal buses being forwarded directly without buffering.

To try to resolve this, we introduced register replication at the output of the realignment module. This involved modifying the SystemVerilog module `if_realign.sv` to instantiate multiple copies of the output register, each driving a subset of downstream logic. The replication reduced routing congestion and minimized delay skew. Figure 3 illustrates the revised register placement.

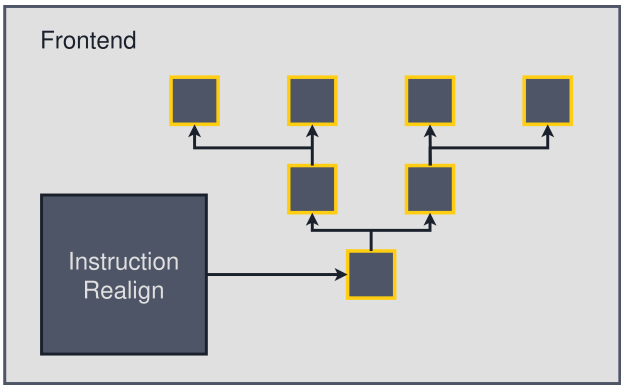

Fig. 3. Register replication in the Frontend realignment module.

As a result, the post-synthesis frequency improved from 83 MHz to 100 MHz. However, this modification wasn't validated during the simulation on questa.

B. Issue Stage

The Issue stage is responsible for selecting ready instructions and dispatching them to the appropriate execution units. A major bottleneck was identified between the operand read logic and the scoreboard, specifically in the `issue_read_operands.sv` and `scoreboard.sv` modules.

To break this long critical path, we introduced an intermediate register bank, effectively pipelining the communication between the operand reading logic and the scoreboard. This added a one-cycle latency to the Issue path, which was accounted for by adjusting control signals to preserve correctness.

The changes required modifying the internal control FSM to ensure that instruction validity and hazard checks remained synchronized despite the additional pipeline stage. Following this modification, the Issue stage reached 100 MHz in post-synthesis analysis but did not work in simulation.

C. Execution Stage

Within the Execution stage, timing violations were traced primarily to the Arithmetic Logic Unit (ALU) and the Physical Memory Protection (PMP) unit. The ALU, located in `alu.sv`, was particularly sensitive due to deeply nested logic chains and wide operand buses.

To optimize, we applied a two-step strategy:

- Refactored complex logic expressions in the ALU to reduce logic depth (e.g., replacing deeply nested ternary operators with parallel multiplexers).
- Inserted pipeline registers at the ALU output, which delayed the forwarding path to the scoreboard by one cycle. Control logic was updated to compensate for the added latency, ensuring correct data commit timing.

Similarly, in the PMP unit (`pmp_unit.sv`), we tried to streamline the access control checks by reordering conditionals and applying early-exit logic, reducing overall evaluation depth .Unfortunately, Theses modifications did not work in simulation on questa.

D. Cache

The original cache modules used in CV32A6 were not optimized for high-frequency operation and introduced major timing constraints, particularly on address decoding and tag comparison paths. Instead of manually optimizing the legacy design, we tried adopting the newer *HPDcache*, developed by the OpenHW Group for enhanced performance.

The HPDcache is currently integrated in the more recent versions of the CVA6, oriented for industrial use. Backporting is most certainly possible, but the parametrization update that was made on those versions practically reshaped the whole project and it was no easy task to try to understand what goes where. Two push requests were particularly helpful on the original CVA6 repo : #2097 and #2059.

E. Integration and Validation

All architectural changes were incrementally merged into a dedicated Git branch (`cv32a6_contest_24_25`), with systematic use of feature branches for individual optimizations. Each commit included synthesis reports, testbench results, and simulation traces to ensure traceability and correctness.

Regression testing was automated using Makefiles and TCL scripts integrated with Vivado and Questa flows. This ensured that each change maintained functionality and improved timing without regressing earlier performance gains.

V. Results

The optimizations described allowed us to raise the post-synthesis frequency of the Frontend, Issue, and Execution pipeline stages to 100 MHz. However, after full placement and routing on the FPGA, parasitic delays from interconnect routing and placement led to a slightly lower final operating frequency. The processor achieved stable execution at 50 MHz on the Zybo Z7-20 board, representing a 25% improvement over the original 40 MHz baseline.

Table II summarizes the frequency improvements observed after synthesis:

TABLE II
Post-Synthesis Frequency Improvements

Pipeline Stage	Before (MHz)	After (MHz)
Frontend	83	100
Issue	83	100
Execution	94	100
Cache	80	*(replaced)*

To evaluate the functional correctness and performance of our design, we executed two benchmarks:

- **CoreMark:** Evaluated computational efficiency. We observed only a 3% drop in CoreMark/MHz performance compared to the baseline—well within the 10% contest threshold.
- **MNIST Inference:** Tested memory and compute interface correctness. Execution completed successfully, confirming compatibility with unchanged benchmark binaries.

Finally, we confirmed functional correctness through cycle-accurate simulation using Questa, ensuring that no timing-induced logic errors or data hazards were introduced. All optimizations passed regression tests and benchmarking validation without manual patching or intervention.

These results demonstrate that substantial timing improvements can be achieved through structural RTL-level modifications, with minimal performance penalties and full compliance with design constraints.

VI. Discussion

The results achieved demonstrate the effectiveness of targeted low-level hardware optimizations in significantly improving the timing performance of an in-order RISC-V core under stringent design and resource constraints. Our approach—anchored in iterative analysis and refinement—successfully mitigated the most critical bottlenecks revealed by Vivado's timing reports.

By combining register replication, pipelining, and logical restructuring, we tried to improve the worst negative slack (WNS) in key modules and achieved a 25% increase in operating frequency on the FPGA target.

However, it was not without its share of problems :

- **Area Overhead:** Register replication and added pipeline stages increased LUT and flip-flop usage by up to 10%, as reflected in the FPGA resource utilization report. While still within Zybo Z7-20 limits, this could hinder portability to smaller or more constrained FPGAs.
- **Latency Impact:** Pipeline insertion, particularly between the ALU and Scoreboard, introduced cycle-level delays that required careful updates to control signals. This necessitated synchronization changes across multiple pipeline stages and control FSMs to preserve in-order execution semantics.
- **Integration Complexity:** Trying to replace the legacy WT cache with HPDcache made one of us dedicated entirely to this task, and the challenge of backporting it hindered a lot our progression on other aspects.

Despite these challenges, the overall performance degradation remained minimal: benchmark results showed only a 3% loss in CoreMark/MHz, comfortably under the contest threshold of 10%. This suggests that our structural optimizations did not compromise computational efficiency and that the pipeline preserved its throughput under real workloads.

More importantly, our results highlight that pipeline-wide timing optimization is an inherently iterative process. Fixing one bottleneck can shift critical paths elsewhere—a phenomenon we observed when improvements in the Frontend and ALU modules made the Scoreboard the new limiting stage. The Scoreboard's internal sorting logic now constitutes a dominant contributor to delay and represents a promising target for future enhancements.

VII. Conclusion

In this work, we presented a structured and effective approach to optimizing the CV32A6 RISC-V processor core for improved timing performance under practical FPGA constraints. Our contributions addressed both architectural and implementation-level challenges within a six-stage in-order pipeline.

By systematically trying to identify and resolve critical timing paths through static analysis and stage-level synthesis, we successfully increased the post-implementation clock frequency by 25%, from 40 MHz to 50 MHz, on a Zybo Z7-20 development board—without compromising functional correctness or exceeding resource constraints.

Importantly, our optimizations preserved full compatibility with the RV32IM_Zicsr instruction set, enabling the successful

execution of unmodified CoreMark and MNIST benchmark binaries. Performance evaluation showed only a 3% degradation in CoreMark/MHz, well within the 10% tolerance defined by the contest rules.

All design changes were made incrementally within a reproducible, version-controlled flow, validated through behavioral simulation and post-synthesis verification. This development pipeline ensures traceability, ease of collaboration, and extensibility for future iterations.

This project not only demonstrates the value of hardware-aware microarchitectural optimizations but also emphasizes the importance of tooling and methodology. Our findings highlight the need for a tight feedback loop between synthesis tools, timing analysis, and RTL refinement—especially in designs targeting performance-constrained platforms like FPGAs.

Future work will explore additional optimizations in the Scoreboard module, floorplanning strategies, and dynamic pipeline reconfiguration to further push the frequency-performance envelope. The experience gained through this project provides a strong foundation for continued innovation in open-source processor design and educational research.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to our academic supervisors, Mr. Carle, Mrs. Rochange, and Mr. Sainrat, for their unwavering support, insightful feedback, and expert guidance throughout the duration of this project. Their mentorship was instrumental in helping us navigate the technical challenges of RTL-level optimization and the organizational demands of collaborative hardware development.

We are also thankful to the organizers of the CVA6 optimization contest—Thales, the CNRS SOC2 research group, and the CNFM—for designing a highly educational and technically demanding challenge. The contest provided a unique and practical opportunity to engage with real-world processor architectures, explore timing optimization on FPGAs, and apply formal design methodologies under well-defined constraints.

We also acknowledge the open-source communities and tool vendors whose contributions made this project feasible. In particular, we thank the OpenHW Group for maintaining the CVA6 repository, and Siemens and AMD/Xilinx for providing access to Questa and Vivado tools, which were central to our simulation and synthesis workflows.

Finally, we thank the IRIT for providing the hardware infrastructure, computing resources, and collaborative environment that enabled the completion of this work.

REFERENCES

[1] OpenHW Group, "CVA6 RISC-V Core," [Online]. Available: https://github.com/openhwgroup/cva6

[2] Xilinx, "Vivado Design Suite," [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[3] Siemens, "Questa Advanced Simulator," [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa/