UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

# OPEN-SOURCE RISC-V IN-ORDER PROCESSOR MODEL FOR A HARDWARE EVENT-DRIVEN SIMULATOR

PAU MORILLAS MUÑOZ

**Thesis supervisor**
CRISTOBAL ORTEGA CARRASCO (OPENCHIP & SOFTWARE TECHNOLOGIES,S.L.)

**Tutor:** RAMON CANAL CORRETGER (Department of Computer Architecture)

**Degree**
Bachelor's Degree in Informatics Engineering (Computer Engineering)

**Bachelor's thesis**

**Facultat d'Informàtica de Barcelona (FIB)**

**Universitat Politècnica de Catalunya (UPC) - BarcelonaTech**

26/06/2025

# Acknowledgements

I'd like to thank Openchip to bring me the opportunity to work with them

# Abstract

This thesis presents the development and validation of an open-source model of the RISC-V CVA6 processor core using the gem5 simulator. The CVA6 is a six-stage, in-order, single-issue CPU that fully supports the RV64GC instruction set and three privilege levels (M, S, U), making it a viable target for Unix-like operating systems. The purpose of this work is to provide a high-accuracy simulation environment for this processor, which enables performance evaluation and early hardware/software co-design without relying exclusively on hardware implementations like FPGAs.

The study is motivated by the growing interest in digital sovereignty and the need for accessible, open hardware tools for academic and industrial use. Despite the popularity of CVA6 in research environments, no public implementation in gem5 currently exists. This project addresses this gap by modeling CVA6 behaviorally within gem5's MinorCPU infrastructure and correlating its performance against real hardware results from an FPGA board running the same benchmarks.

The methodology followed is iterative and divided into key phases: (1) documentation and understanding of gem5 and CVA6 internals; (2) construction of a preliminary CVA6 model within gem5 with support for RISC-V I, M, A, and C extensions; (3) execution of workloads on both gem5 and an FPGA platform; (4) comparative analysis using metrics such as Instructions per Cycle (IPC) to validate simulation accuracy within a 10% deviation threshold.

The project provides a complete benchmarking and simulation environment, with cross-compiled workloads, performance monitoring, and result logging. Furthermore, an in-depth analysis of performance bottlenecks, such as branch prediction discrepancies and instruction latency modeling, is performed. The project also addresses the challenges of aligning architectural features between gem5's general simulation models and CVA6-specific designs, especially in cache behavior, branch prediction mechanisms, and pipeline depth.

Results demonstrate that while some architectural approximations are necessary in gem5 before final model achieves a close approximation of actual performance. This work contributes to enabling faster prototyping and more accessible experimentation with RISC-V architectures and promotes open-source development for both academic and industrial applications.

Finally, sustainability and cost analysis reveal the project's feasibility within an academic framework and underscore its broader societal relevance by supporting Europe's goal for digital independence.

Aquesta tesi presenta el desenvolupament i la validació d'un model de codi obert del nucli del processador RISC-V CVA6 mitjançant el simulador gem5. El CVA6 és una CPU de sis etapes, d'ordre estricte i emissió única, que dona suport complet al conjunt d'instruccions RV64GC i als tres nivells de privilegi (M, S, U), fet que la converteix en una opció viable per a sistemes operatius tipus Unix. L'objectiu d'aquest treball és oferir un entorn de simulació d'alta precisió per a aquest processador, que permeti avaluar-ne el rendiment i fer co-disseny maquinari/softuare en fases inicials, sense dependre exclusivament d'implementacions físiques com les FPGA.

L'estudi està motivat per l'interès creixent en la sobirania digital i la necessitat d'eines de maquinari obert accessibles per a l'àmbit acadèmic i industrial. Malgrat la popularitat del CVA6 en entorns de recerca, actualment no existeix cap implementació pública en gem5. Aquest projecte resol aquesta mancança modelant el comportament del CVA6 dins de la infraestructura MinorCPU de gem5 i correlacionant-ne el rendiment amb resultats reals obtinguts d'una placa FPGA executant els mateixos bancs de proves.

La metodologia seguida és iterativa i es divideix en diverses fases clau: (1) documentació i comprensió de l'interior del gem5 i del CVA6; (2) construcció d'un model preliminar del CVA6 dins de gem5 amb suport per a les extensions I, M, A i C del RISC-V; (3) execució de càrregues de treball tant en gem5 com en una plataforma FPGA; i (4) anàlisi comparativa mitjançant mètriques com les instruccions per cicle (IPC) per validar la precisió de la simulació dins d'un marge de desviació del 10

El projecte ofereix un entorn complet per a benchmarking i simulació, amb càrregues de treball creuades, monitoratge de rendiment i registre de resultats. A més, s'hi realitza una anàlisi detallada dels colls d'ampolla de rendiment, com les discrepàncies en la predicció de salts i la modelització de latències d'instruccions. El treball també afronta els reptes d'alinear les característiques arquitectòniques entre els models generals de simulació de gem5 i el disseny específic del CVA6, especialment pel que fa al comportament de la memòria cau, els mecanismes de predicció de salts i la profunditat del pipeline.

Els resultats demostren que són necessàries algunes aproximacions arquitectòniques dins de gem5 abans d'assolir un model final afí al rendiment real. Aquest treball contribueix a facilitar un prototipat més ràpid i una experimentació més accessible amb arquitectures RISC-V, i promou el desenvolupament de codi obert tant en l'àmbit acadèmic com en l'industrial.

Finalment, l'anàlisi de sostenibilitat i costos demostra la viabilitat del projecte dins d'un marc acadèmic i posa en relleu la seva rellevància social més àmplia, donant suport a l'objectiu europeu d'assolir la independència digital.

# Contents

# 1. Introduction

## 1.1 Context

When developing a microprocessor, the design goes through different phases before becoming final. The first step in this process is the specification of the architecture. In this step, the system architecture is specified to meet various constraints and provide a set of key performance indicators (KPIs). Subsequently, the architecture specification is handed over to a design team that implements this architecture specification using a hardware description language (HDL). This implementation will be synthesized into a Register-Transfer Level (RTL) design.

The Register-Transfer Level is an abstraction of the physical implementation to describe the hardware's behavior and how the components that form it are interconnected. An RTL design is an exact representation of the final physical implementation. However, the RTL design is a complex process due to its lack of flexibility: all components and their low-level behavior must be described in detail for the system to function correctly.

To gain flexibility and accelerate the architecture design process, academic and industrial environments use so-called high-level simulators. These simulators are fundamental tools in the early design phases, as they allow understanding the effects of new functionalities (e.g., developing a new branch predictor for our system) or initiating early hardware/software performance verification (e.g., reducing cache misses for a given algorithm) before delving into the corresponding RTL design. To use this tool, a system must already be designed, to which we want to introduce improvements in its design, as well as add new functionalities through new components or redesign existing ones.

One such simulator is the so-called gem5: an open-source event-driven simulator with cycle-level accuracy that supports the modeling of multiple CPUs, the memory subsystem, and the system interconnection. On top of this system, workloads for a specific Operating System (OS) can be introduced, and certain metrics (e.g., CPI, FLOPS, cache misses, MIPS, etc.) can be collected.

As previously mentioned, gem5 requires a system to simulate. In this case, Openchip, the company associated with this work, is interested in evaluating the RISC-V CVA6 core cluster. The CVA6 core is an open-source RISC-V core [Zurich and of Bologna], a 6-stage, single-issue, in-order CPU that implements the 64-bit RISC-V instruction set. It fully implements the I, M, A, and C extensions and supports three privilege levels, M, S, and U, to provide full support for a Unix-like operating system. Openchip is interested in studying the capabilities of open-source processors used in academic and industrial environments, such as the CVA6. Therefore, due to the CPU's complexity, evaluating this component is considered critical, as failure to meet the environment's general requirements could inevitably lead to unacceptable system performance.

In this work, we aim to create a gem5 model of the CVA6 cluster [Lowe-Power et al. 2020]. This model will allow us to evaluate the behavior and approximate performance of this architecture with high accuracy to identify potential weaknesses and strengths. Consequently, this work can be used to make changes to components where there may be potential performance improvements.

Furthermore, we can determine with considerable accuracy who the project is aimed at and where it can be applied, as well as who can benefit from it. Initially, Openchip is the entity this work is directed toward, as it is the primary stakeholder interested in studying the performance of this architecture with specific applications. Secondly, as stated in the OpenHW documentation, this cluster is oriented toward ASIC and FPGA implementations. Therefore, industrial sectors dedicated to producing products based on these technologies or using them are prominent candidates for utilizing this work. The academic sector can also use it, as this work opens the possibility of conducting further studies that are related or complementary. Finally, this work benefits the academic, scientific, and industrial environments equally. This is because Openchip is working on a European project to achieve digital sovereignty, i.e., the European Union's ability to manufacture its own digital products. Among these, its own chips. Specifically, Openchip aims to produce chips for the high-performance computing (HPC) sector, technologies used in all the mentioned sectors due to their extraordinary computational capacity. Therefore, this project will drive advancements in scientific, industrial, and academic fields.

## 1.2 Justification

Although the CVA6 is an open-source model widely used in the academic field, there are no available models for the CVA6 in gem5 or any other simulator. However, previous works have made efforts to model the CVA6. In this section, we analyze the state of the art in CVA6 modeling.

Previous works on modeling and calibrating the CVA6 in gem5 have focused on microbenchmarks [Ravenelk et al., Shahid et al. 2024]. Microbenchmarks are a good initial approach to calibrating the core. However, their code lacks the complex interactions that could arise in real workloads. In this work, we plan to execute benchmarks that represent real workloads.

If we evaluate an alternative to early verification, we might wonder whether running the CPU directly on an FPGA would be more productive than simulating it in gem5 beforehand. While it is true that real experimentation provides more reliable results, this can become cumbersome due to the size of the workloads we want to run on the CPU. Therefore, an assessment with a high-level model is useful. Additionally, another team will execute these workloads, which will help determine if the results align, i.e., whether the simulation-FPGA error margin is acceptable.

## 1.3 Scope

This section defines the project's scope, specifying the objectives, requirements (functional and non-functional), and potential risks associated with the development and validation of the CVA6 core using gem5 and FPGA.

### 1.3.1 Project Objectives

The general objective of the project is to develop and validate an accurate model of the CVA6 core using the gem5 simulator, with result correlation compared to an FPGA implementation, to analyze its performance in a Unix-like environment. This objective is broken down into the following specific sub-objectives:

- Configure an initial CVA6 model in gem5 that supports the RISC-V I, M, A, and C extensions.
- Prepare and execute benchmarks (e.g., dhrystone) to evaluate performance.
- Collect and correlate statistics between gem5 and FPGA with a deviation of less than 10% based on the article Walker et al. (2018).
- Iteratively adjust the gem5 model to improve its accuracy compared to reference data.

### 1.3.2 Requirements

The project requirements are divided into functional and non-functional to ensure its success:

#### 1.3.2.1 Functional Requirements

- The gem5 model must correctly simulate the CVA6 core, including the M, S, and U privilege levels.
- Workloads must be compatible with RISC-V and cross-compiled for the simulated and FPGA environments.
- A Linux image must be supported, functioning in both gem5 and FPGA for consistent testing.
- The FPGA environment must provide reference metrics (e.g., CPI, execution time) for validation.

#### 1.3.2.2 Non-Functional Requirements

- Simulations in gem5 must complete in less than 24 hours per iteration.
- The gem5 model's accuracy must have a reasonable deviation (typically less than 10%) compared to FPGA metrics.
- Code and benchmarks must be versioned and compatible across three platforms: native x86, gem5, and FPGA.

### 1.3.3 Risks and Obstacles

The following potential risks have been identified, along with associated mitigation strategies:

1. **Initial Configuration Complexity:** The configuration of gem5 and CVA6 may fail due to dependencies or documentation errors. *Mitigation:* Consult official documentation and use QEMU [QEM] as a preliminary testing environment.

2. **Cross-Compilation Incompatibility:** Workloads may encounter errors due to differences in RISC-V extensions or libraries. *Mitigation:* Use static linking and the same version of the GCC compiler [GCC] (e.g., 12.2.0) across all platforms.

3. **Platform Differences:** Metrics may vary between x86, gem5, and FPGA due to inconsistent environment versions. *Mitigation:* Share a common operating system image and test binaries with QEMU before simulations.

4. **gem5 Modeling Limitations:** CVA6 components (e.g., branch predictor) may not be accurately simulable. *Mitigation:* Adapt algorithms to approximate results or document limitations as future work.

5. **Lack of Resources or Time:** Extensive benchmarks may exceed established deadlines. *Mitigation:* Reduce benchmark set and scale according to available time.

This scope establishes the project's boundaries and serves as the foundation for detailed planning and execution in subsequent chapters.

# 2. Project Planning

## 2.1 Methodology and Rigor

This section defines the methodology used to develop the project, based on an iterative and experimental approach tailored to the specifics of working with the gem5 simulator and the CVA6 core. This methodology is divided into three key phases:

1. **Understanding:** The official documentation of gem5 and CVA6 will be consulted to acquire specific technical knowledge, such as the simulator's configuration parameters and the RISC-V extensions (I, M, A, C). This step is essential to establish a solid foundation before implementing practical solutions.

2. **Preparation and Configuration:** Using the gathered information, the gem5 environment will be configured to accurately simulate the CVA6, including the selection of representative benchmarks to evaluate the core's performance. This phase will ensure the reproducibility of experiments.

3. **Execution, Measurement, and Correlation:** Simulations will be run with the configured models, performance will be measured, and results will be correlated with the project's objectives.

The research supervisor will oversee each phase, reviewing progress and adjusting approaches as necessary to ensure technical consistency. This iterative methodology allows for early detection and correction of deviations. Later, the schedule of meetings with the supervisor and coordination with other teams will be detailed.

## 2.2 Tracking Tools

The following tools will be used to ensure control and validation of progress:

- **Git:** It will be used to manage versions of the gem5 configuration code, benchmarks, and simulation results. Each version commit will be linked to a specific milestone (e.g., initial configuration completed) to track the project's evolution.

- **pgfgantt:** This LaTeX package will be used to create Gantt charts with clear deadlines, including milestones such as the completion of configuration or the first simulation. This will facilitate visual tracking of objectives.

- **Microsoft Teams:** It will be used as a communication channel with the supervisor and other teams, recording doubts and key decisions in case of non-in-person meetings. Conversations will be documented to maintain traceability.

These tools will be complemented by logs generated by gem5, which will provide objective metrics (e.g., execution time) to evaluate the achievement of objectives.

## 2.3 Validation

Project validation will be carried out through:

- Weekly periodic meetings with the company supervisor to review partial results (e.g., gem5 configurations, simulation data). If unforeseen issues arise, additional meetings will be scheduled.

- A specific meeting for each task identified in section 1.3, where progress will be compared with the established objectives (e.g., "CVA6 simulation completed with CPI within the expected range").

- Synchronizations with other teams when necessary (e.g., with the FPGA team to validate the CVA6 integration in hardware), documenting the results in brief reports.

This structured system ensures that objectives are met with rigor and allows for timely adjustments in case of deviations.

## 2.4 Temporal Planning

### 2.4.1 Task Description

Below, the tasks necessary to carry out the project are detailed, with a specific level of description, temporal estimations, logical sequence, dependencies, and associated resources. All tasks have their corresponding associated documentation.

1. **T0: Project Management**
   Description: Plan the project and delivery deadlines, analyze objectives, evaluate the budget and its sustainability.
   Estimation: 32 hours.
   Resources: Student, computer with Internet access.
   Dependencies: None (initial task).

2. **T1: Study and Understanding of gem5**
   Description: Analyze the official gem5 documentation to understand simulation parameters and RISC-V core configuration options.
   Estimation: 40 hours.
   Resources: Student (documentation review), computer with Internet access and gem5 installed.
   Dependencies: None (initial task).

3. **T2: Understanding the CVA6 Core**
   Description: Study the technical documentation of the CVA6 to identify the RISC-V extensions (I, M, A, C) and key components (e.g., branch predictor).
   Estimation: 40 hours.
   Resources: Student, official CVA6 documentation (PDFs).
   Dependencies: None (can be done in parallel with T1, but we prefer to complete T1 first to have a solid foundation for integrating the processor into the simulator).

4. **T3: Building the Initial CVA6 Model in gem5**
   Description: Configure a basic CVA6 model in gem5, adjusting parameters such as memory and RISC-V extensions.
   Estimation: 20 hours.
   Resources: Student, computer with gem5, supervisor support to validate the configuration.
   Dependencies: T1 and T2 (requires understanding of gem5 and CVA6).

5. **T4: Preparation of Microbenchmarks and Benchmarks**
   Description: Select and compile microbenchmarks (e.g., arithmetic calculations) and standard benchmarks (e.g., SPEC CPU) for RISC-V.
   Estimation: 20 hours.
   Resources: Student, GCC cross-compiler for RISC-V, native x86 computer.
   Dependencies: T2 (requires knowledge of the CVA6 ISA).

6. **T5: FPGA Environment Configuration**
   Description: Prepare an FPGA board to run the CVA6, including loading the design and a Linux image.
   Estimation: 20 hours.
   Resources: Student, FPGA board (e.g., Nexys A7), support from the FPGA team.
   Dependencies: T2 (requires understanding of the CVA6).

7. **T6: Initial Data Collection on FPGA**
   Description: Run the prepared workloads on the FPGA and record metrics (e.g., CPI, execution time).
   Estimation: 20 hours.
   Resources: Student, FPGA board, measurement tool (e.g., Vivado).
   Dependencies: T4 and T5 (requires benchmarks and configured FPGA).

8. **T7: Execution of Workloads in gem5**
   Description: Simulate the workloads in the gem5 model and collect statistics (e.g., CPI).
   Estimation: 25 hours (1h per iteration).
   Resources: Student, computer with gem5.
   Dependencies: T3 and T4 (requires model and benchmarks).

9. **T8: Correlation of gem5-FPGA Statistics**
   Description: Compare gem5 metrics with FPGA metrics to identify deviations.
   Estimation: 50 hours (2h per iteration).
   Resources: Student, analysis tool (e.g., Python [Pyt], Excel [Exc]).
   Dependencies: T6 and T7 (requires reference and simulation data).

10. **T9: Adjustment of the gem5 Model**
    Description: Modify gem5 model parameters (e.g., branch prediction) to reduce deviation from FPGA.

Estimation: 100 hours (4h per iteration).
Resources: Student, computer with gem5, supervisor support.
Dependencies: T8 (requires knowledge of deviations).

11. **T10: Formalizing Results**
Description: Properly document the experiments conducted, filtering out irrelevant ones and linking them to the final conclusions. Prepare a presentation for the defense of the work before the Tribunal.
Estimation: 24 hours.
Resources: Student, computer with Internet access, supervisor support.
Dependencies: T9 (requires an established model).

Tasks T7, T8, and T9 form an iterative cycle that will be repeated until achieving a deviation of less than 10% between gem5 and FPGA, with a total estimation of 3 iterations (180 hours).

**Logical Sequence:** The tasks follow an initial linear order (T1 to T6). Once initial data is collected (T6), the iterative cycle (T7-T9) begins, where each iteration depends on its predecessor and previous results.

#### 2.4.1.1 Task Summary Table

| Task | Description | Hours | Dependencies | Resources |
|---|---|---|---|---|
| T0 | Project Management | 32 | None | Student, PC |
| T1 | Study of gem5 | 40 | None | Student, PC |
| T2 | Understanding CVA6 | 40 | None | Student, PDFs |
| T3 | Initial gem5 Model | 20 | T1, T2 | Student, gem5, supervisor |
| T4 | Benchmark Preparation | 20 | T2 | Student, GCC |
| T5 | FPGA Configuration | 20 | T2 | Student, FPGA |
| T6 | FPGA Data Collection | 20 | T4, T5 | Student, FPGA |
| T7 | gem5 Execution | 25 | T3, T4 | Student, gem5 |
| T8 | Statistics Correlation | 50 | T6, T7 | Student, Python |
| T9 | Model Adjustment | 100 | T8 | Student, gem5, supervisor |
| T10 | Formalizing Results | 24 | T9 | Student, PC, supervisor |

## 2.5 Project Gantt Estimation

Below, the 375 hours dedicated to the project are distributed. Holidays and weekends are taken into account. As the charts are presented, the accumulated hours are mentioned. The following should be noted:

1. Holidays are highlighted in red, and no work hours are dedicated to them. No hours are dedicated to weekends either.

2. The default dedication (in white) is 4 hours per day. If a different number of hours is dedicated, another color will be used.

3. Due to the nature of the project, tasks will be executed sequentially. Each task has two main concurrent subtasks not mentioned due to their obviousness: documentation and eventual meetings with the company supervisor.

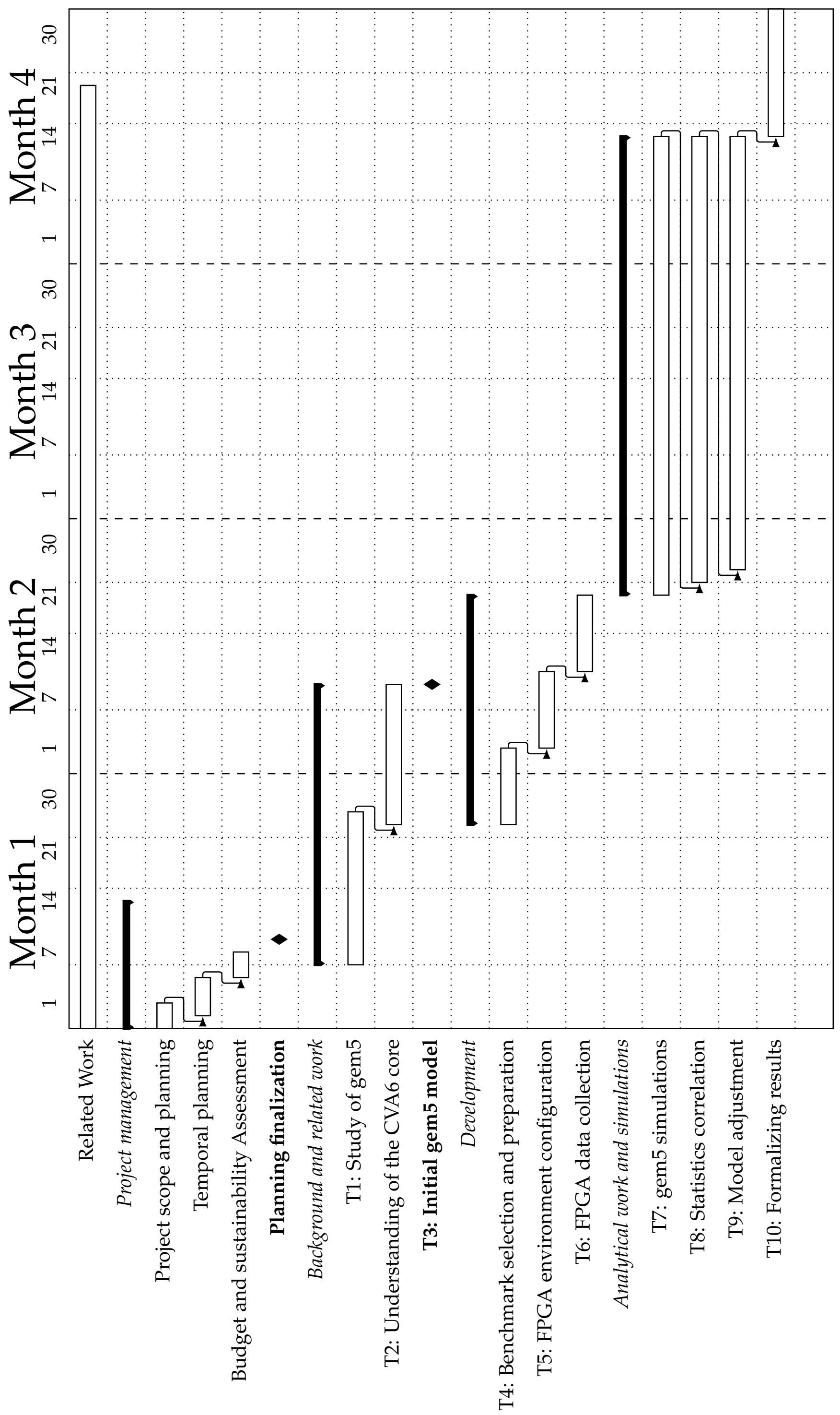


Figure 2.1: Project schedule for this TFG

### Part 1: Project Management

Project management is the shortest part of the work. The following figure presents a Gantt chart representing the first part of the project, corresponding to the Project Management block.

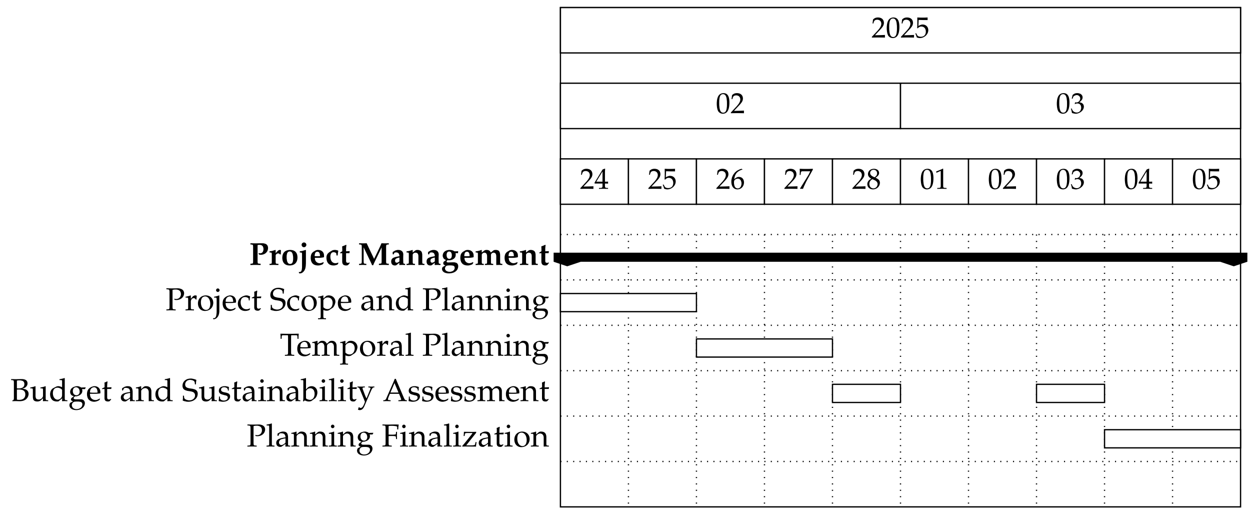


Figure 2.2: Part 1: Project Management. Accumulated Hours: 16 hours

## Part 2a: Initial Work - Study

In this first phase of the initial work, we deepen and consolidate our knowledge of the simulation tool and the CVA6 core. We dedicate two weeks (or 10 working days) to each task, i.e., 20 working days in total.

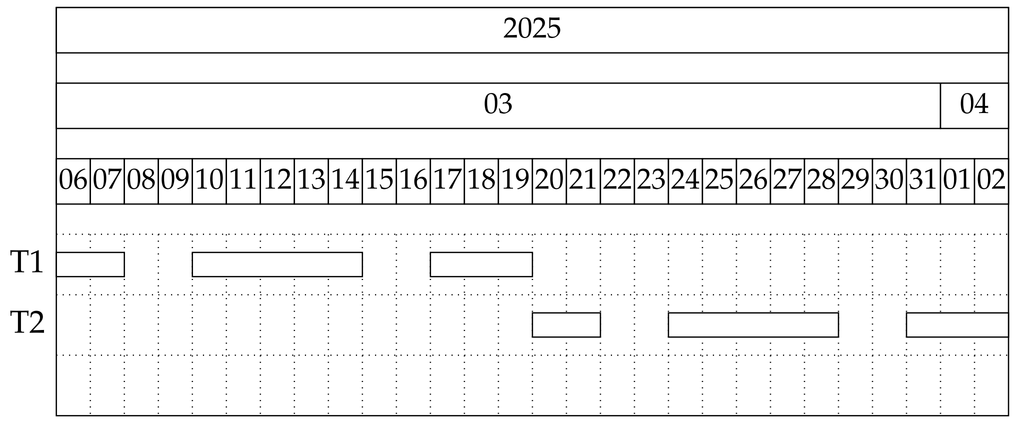


Figure 2.3: Part 2 (Study). Accumulated Hours: 96 hours

## Part 2b: Initial Work - Development

In the second phase of the initial work, we build our working environment, where the experimental work will be carried out. We will use the resources and references gathered during the execution of phase 1. We dedicate 1 week to each task (5 working days), totaling 20 working days.

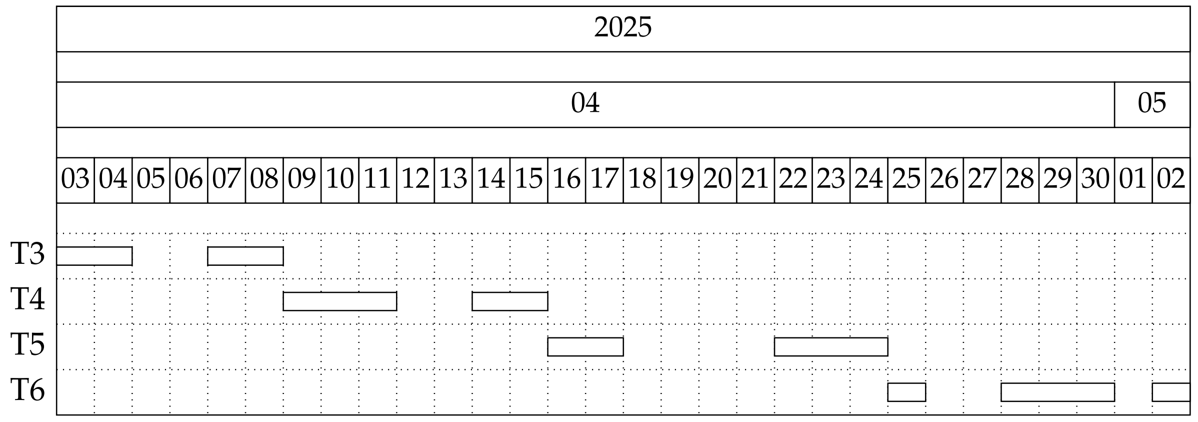


Figure 2.4: Part 2 (Development). Accumulated Hours: 176 hours

## Part 3: Analytical Work

The analytical work will be an iterative process based, in order, on data collection, simulation-FPGA correlation, and high-level model adjustment. On the same day, these tasks will be executed sequentially. Thus, the tasks are concurrent at the day level but sequential at the hour level. This part requires the most hours, as the experimental phase is where most errors and risks associated with the project are encountered. Therefore, we establish a schedule of hours/task and a 7-hour workday. Finally, during the period from June 10 to 13, we establish a 6-hour workday to formalize results and prepare the presentation and defense of the final degree project. Additionally, this part accumulates the 375 hours stipulated in the agreement with the company Openchip.

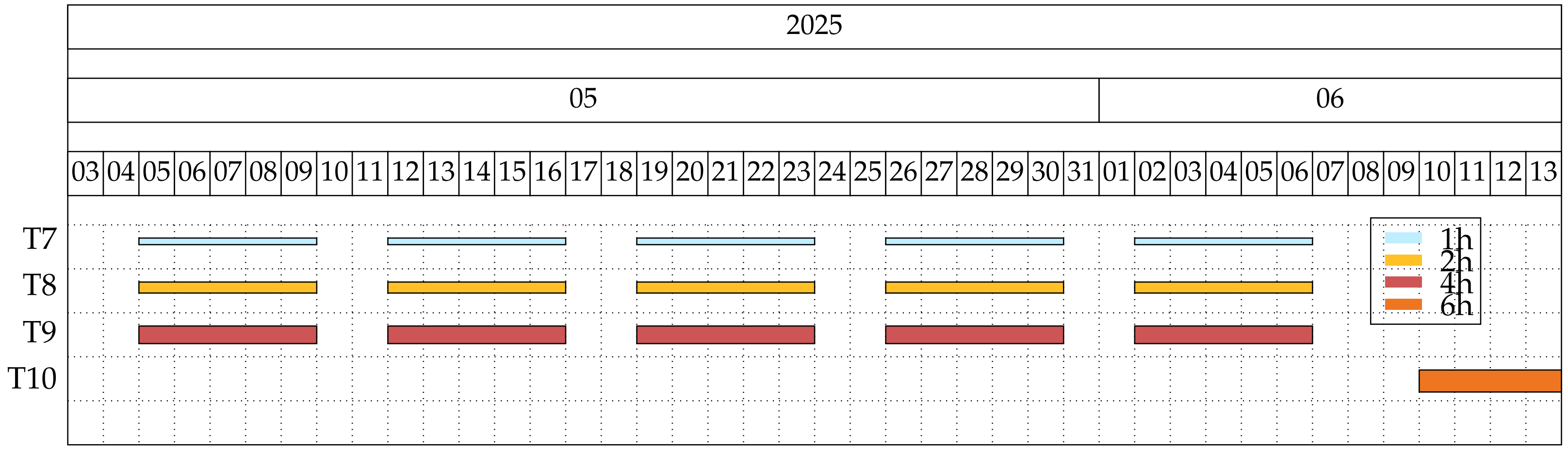


Figure 2.5: Part 3: Iterations. Accumulated Hours: 375 hours

## 2.6 Risk Management: Alternative Plans and Obstacles

This section identifies the main potential risks that could arise during the project's execution and proposes alternative plans to overcome them, ensuring that the established deadlines (June 13, 2025) can be met. Alternative tasks, the impact on the total duration, and the additional resources required for each obstacle are detailed, maintaining consistency with the methodology and project objectives.

### 2.6.1 Risk 1: Errors in the Initial gem5 Configuration

**Risk Description:** The configuration of the gem5 simulator for the CVA6 core (T3) could encounter errors due to misinterpretation of documentation or incompatibilities with selected parameters, delaying subsequent tasks.
**Probability:** Moderate.
**Impact:** High (could affect T7, T8, and T9).
**Alternative Plan:**

a) *Alternative Tasks:* If the initial configuration fails, a preconfigured gem5 model available in the community (e.g., a generic RISC-V model) will be used as a starting point, progressively adapting it to the CVA6. Priority will be given to consulting the supervisor or specialized forums (e.g., gem5 Google groups) to resolve specific errors.

b) *Impact on Duration:* This plan could add 2 working days (8 hours) to T3 to adjust the pre-existing model, but it would be offset by reducing the initial gem5 study time (T1) by 2 days, leveraging the documentation already integrated into the alternative model. The total duration would remain within the planned 81 working days.

c) *Additional Resources:* Access to a gem5 expert (supervisor or external collaborator) to validate the alternative model and Internet connection to download the preconfigured model.

### 2.6.2 Risk 2: Issues with the FPGA Environment

**Risk Description:** The configuration of the FPGA environment (T5) or data collection (T6) could fail due to hardware errors (e.g., defective FPGA board) or issues

with the Linux image, affecting the reference data for correlation.
**Probability:** Low-moderate.
**Impact:** High (critical for T8).
**Alternative Plan:**

a) *Alternative Tasks:* In case of FPGA failure, a RISC-V emulator (e.g., QEMU) will be used to generate provisional reference data. The FPGA team will be contacted to replace or repair the board in parallel, reintegrating real data when available.

b) *Impact on Duration:* Using QEMU would add 3 working days (12 hours) to T5 to configure the emulator, but it could overlap with T6 and T7, keeping the schedule intact. If FPGA repair is delayed, 2 days of T6 would be rescheduled to the end of T9, without exceeding the June 13 deadline.

c) *Additional Resources:* License or access to QEMU (free), technical support from the FPGA team, and a backup FPGA board (if available).

### 2.6.3 Risk 3: Excessive Deviations in gem5-FPGA Correlation

**Risk Description:** The correlation (T8) and adjustment (T9) iterations could show deviations greater than 10% between gem5 and FPGA, requiring more than the 3 planned iterations.
**Probability:** Moderate-high.
**Impact:** Moderate (would delay T10).
**Alternative Plan:**

a) *Alternative Tasks:* If additional iterations are needed, the scope of benchmarks will be reduced to a critical subset (e.g., only arithmetic calculations instead of full SPEC CPU), prioritizing accuracy in key metrics (CPI). Limitations will be documented in T10 to justify this.

b) *Impact on Duration:* An additional iteration would add 7 working days (25 hours), but it would be offset by reducing T10 from 24 to 16 hours, simplifying the formalization of results (e.g., shorter presentation). The project would remain within the deadline.

c) *Additional Resources:* Supervisor support to validate the reduced scope and rapid analysis tools (e.g., predefined Python scripts) to accelerate T8.

### 2.6.4 Risk 4: Delays in Supervisor Availability

**Risk Description:** The company supervisor may have limited availability to review key tasks (e.g., T3, T9, T10), which could slow down validation and necessary adjustments.
**Probability:** Low-moderate.
**Impact:** Moderate (could shift T9 and T10).
**Alternative Plan:**

a) *Alternative Tasks:* Asynchronous reviews will be scheduled via email or Microsoft Teams, sending the supervisor brief reports with partial results (e.g., gem5 configuration or correlation statistics). If necessary, an external collaborator with RISC-V experience will be consulted for provisional support.

b) *Impact on Duration:* This plan would add 1 working day (4 hours) for asynchronous coordination to T3 and T9, but it would be offset by advancing T10 by 1 day, using the response time to draft final documentation. The total duration would not be affected.

c) *Additional Resources:* Access to Microsoft Teams for asynchronous communication and potential support from an external collaborator (e.g., a member of the FPGA team or Openchip).

### 2.6.5 Risk 5: Failures in Benchmark Compilation

**Risk Description:** The preparation of benchmarks (T4) could fail due to errors in the GCC cross-compiler for RISC-V or incompatibilities with selected code (e.g., NAS or SPEC CPU).
**Probability:** Moderate.
**Impact:** Moderate (would affect T6 and T7).
**Alternative Plan:**

a) *Alternative Tasks:* A set of alternative benchmarks already compiled by the RISC-V community (e.g., RISC-V Torture Tests) will be used, adapting them to the CVA6. In parallel, compilation errors will be resolved with technical support or by updating the compiler.

b) *Impact on Duration:* This plan would add 2 working days (8 hours) to T4 to obtain and adjust alternative benchmarks, but it could overlap with T5, keeping the overall schedule intact within the 81 working days.

c) *Additional Resources:* Internet connection to download precompiled benchmarks and technical support from the supervisor or a GCC compiler specialist.

### 2.6.6 Risk 6: Student Overload

**Risk Description:** The student could experience fatigue or personal unforeseen events that reduce productivity, especially during the intensive iteration phases (T7-T9), affecting compliance with deadlines.
**Probability:** Low-moderate.
**Impact:** High (could delay the entire project).
**Alternative Plan:**

a) *Alternative Tasks:* The 7-hour workdays will be rescheduled to 5 hours daily during T7-T9, prioritizing critical tasks (e.g., basic correlation instead of exhaustive adjustments) and delegating the final review to the supervisor. Secondary parts of T10 (e.g., presentation formatting) could be outsourced to a colleague or assistant.

b) *Impact on Duration:* Reducing the workday would add 5 working days (25 hours) to T7-T9, but it would be offset by advancing T10 by 3 days and simplifying documentation (from 24 to 16 hours). The project would remain within the June 13 deadline.

c) *Additional Resources:* Supervisor support to prioritize tasks and potential collaboration from an assistant for administrative or formatting tasks.

# 3. Economic Management and Sustainability

## 3.1 Project Expenses and Costs

This section details the expenses and costs associated with the execution of the project, primarily based on the total project duration. It considers human resources, hardware amortization, software, electricity expenses, and applicable taxes (VAT). The calculations aim to ensure efficient resource management and an accurate estimation of the required budget.

### 3.1.1 Human Resources

Human resources involve two main roles: the Project Manager and the Analyst and Programmer (technical executor). Average salaries are estimated based on Spanish labor market data for 2025, considering a total workload of 375 hours.
**Project Manager:**

- Average annual salary: €40,420 (according to PayScale for Project Manager IT, 2025).
- Standard annual hours: 1,720 hours (40 hours/week $\times$ 43 weeks, excluding vacations).
- Cost per hour: €40,420 / 1,720 h = €23.50/h.
- Hours dedicated: 32 h (T0: Project Management).
- Total cost: €23.50/h $\times$ 32 h = €752.

**Analyst and Programmer:**

- Average annual salary: €28,783 (according to PayScale for Software Developer, 2025).
- Cost per hour: €28,783 / 1,720 h = €16.73/h.

- Hours dedicated: 343 h (T1 to T10, 375 h total - 32 h of management).
- Total cost: €16.73/h × 343 h = €5,738.39.

**Total human resources:** €752 + €5,738.39 = €6,490.39.

### 3.1.2 Hardware Amortization

Two pieces of equipment are used: an HP ProBook 460 16-inch G11 Notebook PC and a Genesys 2 FPGA, with an estimated useful life of 4 years (linear amortization). Prices are estimated based on 2025 market data.
**HP ProBook 460 16-inch G11 Notebook PC:**

- Estimated price: €1,200 (based on prices of similar HP ProBook laptops in 2025).
- Useful life: 4 years = 4 × 1,720 h = 6,880 h total.
- Cost per hour: €1,200 / 6,880 h = €0.174/h.
- Hours used in the project: 335 h (375 h total - 40 h for FPGA).
- Amortized cost: €0.174/h × 335 h = €58.29.

**Genesys 2 FPGA:**

- Estimated price: €600 (based on DigiKey prices for Genesys 2 in 2025).
- Cost per hour: €600 / 6,880 h = €0.087/h.
- Hours used in the project: 40 h (T5 and T6).
- Amortized cost: €0.087/h × 40 h = €3.48.

**Total hardware amortization:** €58.29 + €3.48 = €61.77.

### 3.1.3 Software

The project uses open-source software, with no associated costs:

- **Ubuntu:** Free operating system, used for running gem5 and other tasks.
- **gem5:** Open-source simulator, with no licenses or additional costs.

**Total software:** €0.00.

### 3.1.4 Electricity Expenses

Tasks are performed in an office from 10 AM to 2 PM (4 hours daily). Electricity consumption is calculated for a computer and basic lighting.

- Average electricity cost in Spain in 2025: €0.25/kWh (estimate based on current trends).
- Estimated consumption: 0.2 kW (computer) + 0.05 kW (lighting) = 0.25 kW per hour.
- Cost per hour: 0.25 kW × €0.25/kWh = €0.0625/h.
- Total hours: 375 h / 7 h per day = 53.57 days; with 4 h/day in the office = 214 h.
- Total cost: €0.0625/h × 214 h = €13.38.

**Total electricity expenses:** €13.38.

### 3.1.5 Taxes

The applicable VAT in Spain in 2025 is 21

- Taxable base: €6,490.39 (human resources) + €61.77 (hardware) = €6,552.16.
- VAT (21

**Total VAT:** €1,375.95.

### 3.1.6 Cost Summary

| **Concept** | **Cost (€)** |
|---|---:|
| Human Resources | 6,490.39 |
| Hardware Amortization | 61.77 |
| Software | 0.00 |
| Electricity Expenses | 13.38 |
| Subtotal (excluding VAT) | 6,565.54 |
| VAT (21%) | 1,375.95 |
| **Total** | 7,941.49 |

## 3.2 Contingency Plan

This section estimates the additional costs derived from activating contingency plans for the six risks identified in the "Risk Management: Alternative Plans and Obstacles" section. For each risk, the alternative plan is reviewed, its impact on resources is specified, and the associated cost is calculated, including VAT when applicable. Costs have been calculated while maintaining economic efficiency and ensuring project completion within the established deadline.

### 3.2.1 Risk 1: Errors in the Initial gem5 Configuration

**Contingency Plan:** If the initial gem5 configuration (T3) fails, a preconfigured community model will be used and adapted to the CVA6, with support from the supervisor or specialized forums.
**Associated Cost:**

- Alternative tasks: Model adaptation (8 additional hours) and consultation with an expert.
- Human resources: Supervisor support (Project Manager) for 4 additional hours at €23.50/h = €94.
- Additional resources: Internet connection (already included in the office).
- Subtotal (excluding VAT): €94.
- VAT (21
- **Total:** €113.74.

### 3.2.2 Risk 2: Issues with the FPGA Environment

**Contingency Plan:** In case of FPGA failure (T5 or T6), QEMU will be used to generate provisional data while the FPGA board is repaired or replaced.
**Associated Cost:**

- Alternative tasks: QEMU configuration (12 additional hours) and coordination with the FPGA team.
- Human resources: Analyst and Programmer for 12 hours at €16.73/h = €200.76; FPGA team technical support (assumed as 2 hours of supervisor time at €23.50/h = €47).

- Additional resources: A second FPGA board is not counted as an extra cost since it is optional.
- Subtotal (excluding VAT): €200.76 + €47 = €247.76.
- VAT (21
- **Total:** €299.79.

### 3.2.3 Risk 3: Excessive Deviations in gem5-FPGA Correlation

**Contingency Plan:** If additional iterations (T8 and T9) are required, the benchmark scope will be reduced, and T10 will be simplified, with supervisor support and rapid analysis tools.
**Associated Cost:**

- Alternative tasks: Additional iteration (25 hours) and T10 simplification (-8 hours).
- Human resources: Analyst and Programmer for 25 hours at €16.73/h = €418.25; supervisor for validation for 4 hours at €23.50/h = €94.
- Additional resources: Python scripts (free, already included).
- Subtotal (excluding VAT): €418.25 + €94 = €512.25.
- VAT (21
- **Total:** €619.82.

### 3.2.4 Risk 4: Delays in Supervisor Availability

**Contingency Plan:** Asynchronous reviews via Microsoft Teams and potential support from an external collaborator if the supervisor is unavailable.
**Associated Cost:**

- Alternative tasks: Asynchronous coordination (4 additional hours).
- Human resources: Analyst and Programmer for 4 hours at €16.73/h = €66.92; external collaborator (assumed similar rate to supervisor) for 2 hours at €23.50/h = €47.
- Additional resources: Microsoft Teams (free, license already included).

- Subtotal (excluding VAT): €66.92 + €47 = €113.92.
- VAT (21
- **Total:** €137.84.

### 3.2.5 Risk 5: Failures in Benchmark Compilation

**Contingency Plan:** Use of precompiled RISC-V community benchmarks while resolving compilation errors with technical support.
**Associated Cost:**

- Alternative tasks: Obtaining and adapting benchmarks (8 additional hours).
- Human resources: Analyst and Programmer for 8 hours at €16.73/h = €133.84; supervisor support for 2 hours at €23.50/h = €47.
- Additional resources: Internet connection (already included).
- Subtotal (excluding VAT): €133.84 + €47 = €180.84.
- VAT (21
- **Total:** €218.82.

### 3.2.6 Risk 6: Student Overload

**Contingency Plan:** Reduction of the workday to 5 hours daily during T7-T9, task prioritization, and outsourcing parts of T10.
**Associated Cost:**

- Alternative tasks: Rescheduling (+25 hours for T7-T9) and T10 simplification (-8 hours).
- Human resources: Analyst and Programmer for 25 additional hours at €16.73/h = €418.25; supervisor for prioritization for 4 hours at €23.50/h = €94; assistant for administrative tasks (assumed reduced rate) for 8 hours at €10/h = €80.
- Subtotal (excluding VAT): €418.25 + €94 + €80 = €592.25.
- VAT (21
- **Total:** €716.62.

### 3.2.7 Contingency Cost Summary

| **Risk** | **Cost (€)** |
|---|---|
| Errors in gem5 configuration | 113.74 |
| Issues with FPGA environment | 299.79 |
| Excessive gem5-FPGA deviations | 619.82 |
| Delays in supervisor availability | 137.84 |
| Failures in benchmark compilation | 218.82 |
| Student overload | 716.62 |
| **Maximum total (all risks activated)** | 2,106.63 |

### 3.2.8 Conclusion

The costs associated with contingency plans range from €113.74 to €716.62 per risk, with a maximum cumulative total of €2,106.63 if all plans were activated simultaneously. In a realistic scenario, only some risks would materialize, keeping the budget increase manageable (e.g., €500-1,000 if 2-3 risks are combined). This additional cost, added to the base budget of €7,941.49, raises the potential total to approximately €9,000-10,000, a reasonable range for a technical project of this nature. Flexible planning and the use of already available resources ensure that the project can be completed within the deadline and with efficient economic management. Below, a table summarizes the project costs and their corresponding contingency plan.

| **Concept** | **Cost (€)** |
|---|---|
| Human Resources (base) | 6,490.39 |
| Hardware Amortization | 61.77 |
| Software | 0.00 |
| Electricity Expenses | 13.38 |
| Subtotal (excluding VAT) | 6,565.54 |
| VAT (21%) of base project | 1,375.95 |
| **Total base project** | 7,941.49 |
| Risk 1: Errors in gem5 configuration | 113.74 |
| Risk 2: Issues with FPGA environment | 299.79 |
| Risk 3: Excessive gem5-FPGA deviations | 619.82 |
| Risk 4: Delays in supervisor availability | 137.84 |
| Risk 5: Failures in benchmark compilation | 218.82 |
| Risk 6: Student overload | 716.62 |
| **Maximum total of contingency plans** | 2,106.63 |
| **Overall total (base + maximum contingency)** | 10,048.12 |

## 3.3 Sustainability Report

This report analyzes the sustainability of the project from three integrated perspectives: environmental, economic, and social. Through a detailed assessment of the final degree project (TFG) production process, its useful life, and potential risks, the goal is to understand the global impact of developing the gem5-CVA6 model while reflecting on how these aspects are currently managed in the state of the art for similar projects. The objective is to ensure responsible planning that minimizes negative effects and maximizes long-term viability.

### 3.3.1 Environmental Perspective

The environmental impact of the TFG begins with its production process, where energy consumption and waste generation are key factors. During the 375 hours of work, of which 214 are carried out in an office from 10 AM to 2 PM, the estimated consumption is 0.25 kW/h, resulting in 53.5 kWh total. This, combined with an emission factor of 0.22 kg CO2/kWh in Spain in 2025, generates 11.77 kg of CO2, a footprint equivalent to the domestic use of an appliance for a month. Additionally, the absence of direct waste during this phase, thanks to the exclusive use of an already-acquired computer and FPGA, keeps the immediate impact low. However, considering the project's useful life, assuming the gem5-CVA6 model is used for 4 years with 100 additional simulations of 10 hours each (1,000 hours total), energy consumption rises to 250 kWh and emissions to 55 kg of CO2. This period also includes hardware obsolescence, with 2 kg of electronic waste (1.5 kg from the HP ProBook and 0.5 kg from the FPGA) if not recycled, a moderate impact that can be mitigated with sustainable practices. Nevertheless, risks could increase this footprint: for example, issues with the FPGA (Risk 2) could add 3 kWh (0.66 kg CO2) with QEMU use, while excessive deviations in correlation (Risk 3) or student overload (Risk 6) could add 12.5 kWh (2.76 kg CO2). In total, the maximum additional impact would be 15.5 kWh and 3.42 kg CO2, manageable if simulations are optimized and low-consumption tools are prioritized. Currently, the state of the art in simulation projects like gem5 shows a trend toward reducing the energy footprint through efficient servers and renewable energy in data centers; however, in individual TFGs, environmental management is often limited to using existing equipment without specific recycling or CO2 offset protocols, an aspect this project could improve with more conscious planning.

### 3.3.2 Economic Perspective

From an economic perspective, the TFG production process involves detailed consumption of human and material resources, with a base cost of €7,941.49. This figure includes €6,490.39 for human resources (€752 for the Project Manager and €5,738.39 for the Analyst and Programmer), €61.77 for hardware amortization (HP ProBook and FPGA), €13.38 for electricity expenses, and €1,375.95 for 21% VAT. This represents a reasonable invoice for a potential client, supported by a 375-hour temporal planning that ensures efficiency. When projecting the useful life, assuming the gem5-CVA6 model could be licensed for research or companies (e.g., €5,000/year for 4 years = €20,000), potential revenues exceed total costs, including maintenance (€2,000) and additional energy (€62.50), with an estimated net profit of €9,996.01. However, this viability may be threatened by economic risks: for example, excessive deviations in correlation (Risk 3) could add €619.82 and delay formalization, while student overload (Risk 6) could increase costs by €716.62 with a 5-day extension. In the worst-case scenario, activating all contingency plans would add €2,106.63, raising the total to €10,048.12, but the profit would remain positive (€7,893.38). To mitigate this, tasks could be simplified or external support sought, keeping the schedule intact. In the state of the art, similar computer architecture projects are often funded by academic grants or industrial collaborations, with similar human costs but less emphasis on detailed contingency plans; this TFG stands out for its comprehensive economic foresight, ensuring resilience against unforeseen events.

### 3.3.3 Social Perspective

The social impact of the TFG begins with the production process, where my personal involvement and immediate environment have been transformed. This project has taught me to manage time and complex technical resources, increasing my confidence in simulation and RISC-V, although it has meant less social time with family and peers. These, in turn, have benefited from shared knowledge about technical sustainability, a positive collateral effect. When considering the useful life, the gem5-CVA6 model could have a broader impact: it will benefit the academic community by facilitating teaching and research in computer architecture and could optimize RISC-V designs for companies like Openchip, with an indirect societal impact through more efficient technologies. Quantifying this is challenging, but I imagine this work could inspire future students or contribute to sustainable innovations. However, social risks must be considered: delays in supervisor availability (Risk 4) could frustrate the team and affect morale, while personal overload (Risk 6) could reduce work quality and strain personal relationships. To avoid this, I would establish regular breaks and fluid communication with the supervisor, minimizing negative impacts. In the state of the art, similar academic projects often prioritize technical impact over social effects, with little attention to personal or collective consequences; this TFG, in contrast, actively reflects on these aspects, promoting a balance between technical success and human well-being.

# 4. Theoretical Analysis

## 4.1 CVA6 Design Concepts

### 4.1.1 Introduction

The Ariane core (the name given to the CVA6 by its creators) is a 64-bit application-class processor based on the RISC-V instruction set architecture, designed to support full operating systems like Linux. Implemented in GLOBALFOUNDRIES 22 FDX technology, it offers a maximum frequency of 1.7 GHz and energy efficiency of up to 40 Gop/sW. Below, the most relevant design concepts are detailed, drawn from the work of Zaruba and Benini (IEEE Transactions on VLSI Systems, 2019). Different configurations followed the design of Ariane, and each one of them is adapted to be applied to a certain field. To fix a reference from now onward, we will use the `cv64a6_imafdc_sv39_config_pkg.sv` Verilog configuration file, from the GitHub repository, which we will explain below:

The CVA6 is single-issue, six-stage, in-order application-class core implementation of the 64-bit ISA variant (RV64GC). Figure 4.1 shows the blocks that compose the six-stage pipeline of the CVA6. The pipeline is divided into frontend (reading current instruction and decoding it) and backend (executing and commiting an instruction).

The frontend is divided in 3 pipestages: PC generation, Instruction fetch, and Instruction decode. While the backend is divided in 3 pipestages: Issue, Execute, and commit stages.

In the PC Generation stage, the address of the next instruction is generated. This can come from a control and status registers (CSRs) when returning from an exception, debug interface, mispredicted branch, or consecutive fetch. This address is checked against the instruction TLB (ITLB) and the instruction cache is requested to read the address.

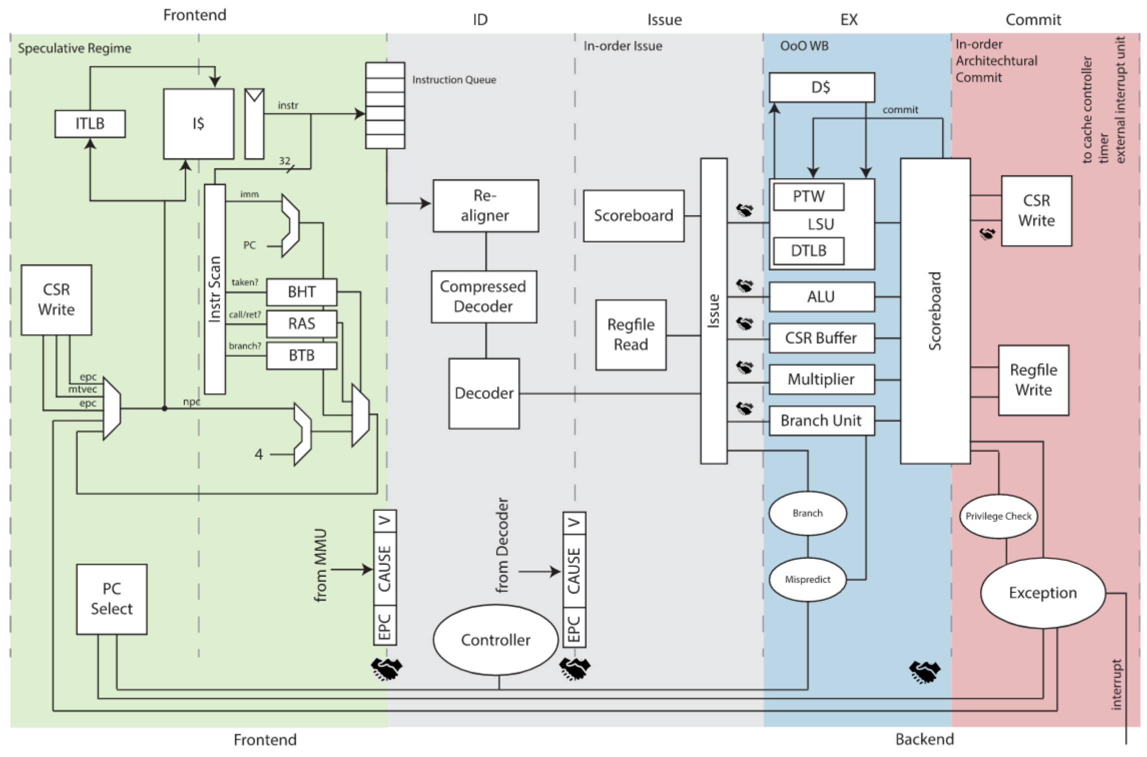


Figure 4.1: Block diagram of the six-stage pipeline of CVA6 Zaruba and Benini (2019b).

In the instruction fetch stage, the instruction cache is read with the corresponding PC. The fetched instruction is sent to the instruction queue (IQ) and is checked against the branch history table (BHT), return address stack (RAS), and branch target buffer (BTB) to predict possible branches.

The instruction decode stage is responsible of realigning potentially unaligned instrucctions, decompress them (in case of compressed instructions), and decodes them. Decoded instructions are written into the issue queue of the issue stage.

The issue stage schedules instructions into the multiplie execution units of the execute stage. For that, it has the issue queue with the decoded instructions, a scoreboard to keep track of the dependencies of the previous issued instructions, and a small reorder buffer (ROB) to remove write-after-write (WAW) dependencies. An instruction is issued once all their source operands are ready.

In the execution stage, instructions are executed by different execution units in this stage. The execution units for the arithmetic logic unit (ALU), multiplier/divider, and CSR handling have a fixed latency (e.g. all ALU instructions take the same number of cycles to be executed). The floating point unit (FPU) and load/store unit (LSU) have a variable latency. Therefore, instructions can be executed out-of-order for the functional units.

Finally, the commit stage reads the up to 2 instructions from the head of the ROB and commits instructions. Once an instruction is committed, the register file is updated for the destination register of the instruction.

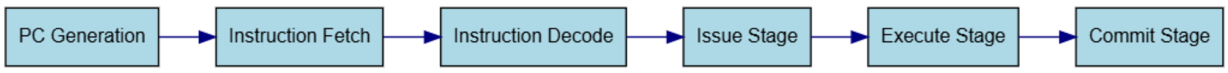


Figure 4.2: CVA6 Pipeline Stages

### 4.1.2 Branch Prediction

To mitigate the impact of mispredicted branches, Ariane uses:

- **Branch History Table (BHT)**: A two-bit saturation counter to predict branch outcomes.
- **Branch Target Buffer (BTB)**: Stores target addresses for jumps.
- **Return Address Stack (RAS)**: Predicts return addresses for function calls.

Static prediction (in the absence of a valid entry) is based on backward branches (taken) and forward branches (not taken), leveraging the PC-relative nature of RISC-V branches.

Ariane performs predictions in the Instruction Fetch stage to feed back to the PC Generation stage. Branch mispredictions are detected in the Execute stage. In case of a misprediction, the subsequent instructions in the pipeline must be flushed.

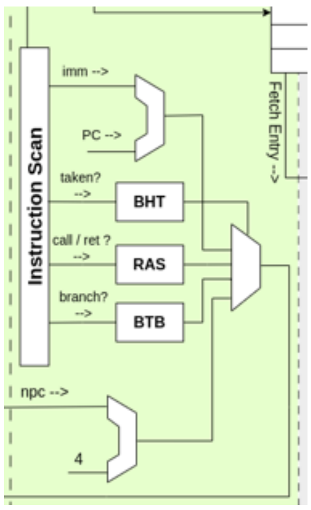


Figure 4.3: Branch Prediction Components.

### 4.1.3 Virtual Memory

Ariane supports address translation through a Memory Management Unit (MMU) with:

- Separate TLBs (Translation Lookaside Buffers) for instructions and data, fully associative and based on flip-flops.
- A Page Table Walker (PTW) to resolve TLB misses by consulting main memory.

## 4.2 CVA6 Energy Efficiency and Performance Analysis

### 4.2.1 Energy Analysis

Analysis based on silicon measurements and post-layout simulations reveals:

- **Energy per Operation**: 51.8 pJ for a mixed workload, with a maximum of 40 Gop/sW.
- **Cache Contribution**: The L1 caches (16 kB instructions, 32 kB data) are the largest consumers of energy and area (470 kGE data, 210 kGE instructions).
- **Virtual Memory**: TLBs and PTW add significant energy overhead (up to 3.8 pJ per instruction).

### 4.2.2 Performance

- **IPC (Instructions per Cycle)**: 0.87 measured with Dhrystone[1], limited by branch predictions and load latency (3 cycles).
- **Frequency**: Up to 1.7 GHz at 1.15 V, with a minimum of 220 MHz at 0.5 V.

### 4.2.3 Physical Design

Implemented in 22-nm FDSOI, Ariane uses:

- **Optimizations**: Useful skew, clock shielding, and a dense power grid to reduce IR drop.
- **Critical Path**: Located around the data caches (30 equivalent NAND gates), due to 8-way associativity.
- **Forward Body Bias (FBB)**: Increases speed by up to 30% at 0.5 V but exponentially increases leakage power at high voltages.

### 4.2.4 Comparison with Other Cores

Table 4.1 summarizes Ariane compared to other RISC-V cores:

Table 4.1: Comparison of RISC-V Cores

| | Ariane | Rocket | BOOM | SHAKTI |
|---|---|---|---|---|
| Bits | 64 | 32/64 | 64 | 64 |
| Technology | 22 nm | 45 nm | 45 nm | 22 nm |
| Frequency | 1.7 GHz | 1.6 GHz | 1.5 GHz | 800 MHz |
| Energy/Op | 52 pJ | 100 pJ | 133 pJ | 122 pJ |
| IPC | 0.87 | 0.95 | 1.45 | 0.9 |

## 4.3 Modeling Ariane in gem5

This chapter analyzes the differences between the **MinorCPU** model of the gem5 simulator and the **Ariane** core. The MinorCPU is a generic in-order execution model provided by gem5, which we will use to create our performance model. Ariane, on the other hand, is a specific open-source design, implemented in silicon

[1]Measured with 128-entry BHT, 64-entry BTB and 2-entry RAS.

and optimized for energy efficiency and performance, addressing aspects that we do not cover in this project. Below, their characteristics are compared in terms of microarchitecture, configuration, and intended use, based on the gem5 documentation Lowe-Power et al. (2020) and the Ariane design described by Zaruba and Benini Zurich and of Bologna.

### 4.3.1 Microarchitecture

#### 4.3.1.1 Pipeline

The **MinorCPU** in gem5 is an in-order model with a fixed **four-stage pipeline**:

- **Fetch1**: Fetches memory lines from the instruction cache (I-cache).
- **Fetch2**: Breaks down lines into macro-op instructions and predicts branches.
- **Decode**: Converts macro-ops into micro-ops (if any, not the case in our case).
- **Execute**: Executes micro-ops using configurable functional units.

This simplified structure is designed to be flexible and configurable via Python scripts, allowing adjustments to pipeline depth and functional units (e.g., ALU, multiplier, memory unit).

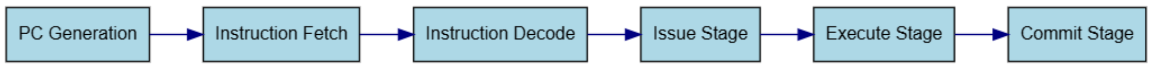


Figure 4.4: Pipeline stages of the CVA6.

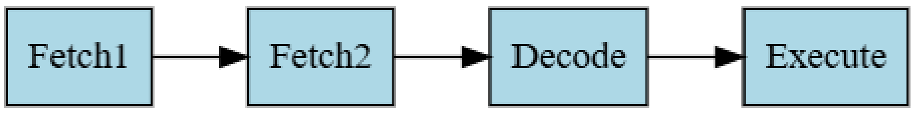


Figure 4.5: Pipeline stages of the minor CPU gem5 model. Issue and Commit stages are integrated into the Execute stage.

To mitigate the different number of stages of each entities, we plan on incrementing each instruction latency by 2. For example, during the creation of the ALU (one of the five Functional Units), we will instantiate a operation latency of 3 instead of the operation latency of 1. The functional units can be also pipelined, so execution flow will be the same in terms of number of stages.

#### 4.3.1.2 Execution and Dependency Management

The MinorCPU uses a **scoreboard** to track dependencies between instructions and an in-flight instruction queue (`inFlightInsts`) to maintain the order of issue and retirement. This components simulate with accuracy the Issue, Execute and Commit stages from the CVA6.

Its default configuration assumes fixed latencies for instructions, defined in the `MinorCPU.py` file. Fixed latencies are a problem when facing division operation latency, which is variable in the CVA6 (they go from 2 to 64).

#### 4.3.1.3 Branch and Jump Prediction

Additionally, Ariane includes a **branch prediction** mechanism with a Branch History Table (BHT) and a Return Address Stack (RAS), features absent in the MinorCPU, which does not natively model branch prediction. To instantiate a branch predictor, the `BranchPredictor.py` object in Python must be used, which internally uses the `2bit_local` predictor, the one most easily adaptable to the CVA6. However, the behavior of this component in the two designs is fundamentally different. This is because the CVA6 predictor leverages the nature of the RISC-V ISA, while the gem5 predictor does not exploit any such characteristics, exhibiting generic behavior.

Conceptually, the predictor used in gem5 follows these steps for a branch instruction:

**gem5 BPU Behavior for a Branch Instruction**

1. Consult the BHT to obtain a prediction.
2. If the prediction is *Taken*, consult the BTB to obtain the branch target.
3. If there is a BTB or RAS miss, predict *Not Taken*. Otherwise, update the PC with the target from the entry.

**CVA6 BPU Behavior for a Branch Instruction**

1. Consult the BHT to obtain a prediction.
2. Check the *valid flag*. If negative, apply BTFN as static prediction.
3. If the prediction is *Taken*, update the new PC with the sum of the instruction's immediate and the current PC.

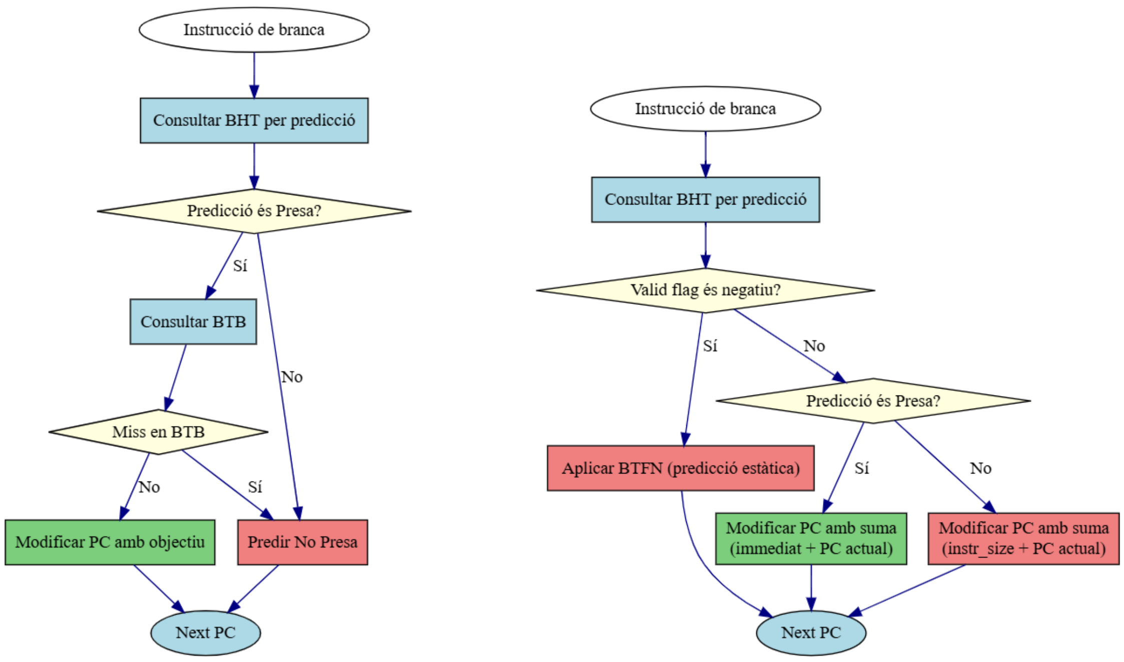

Instrucció de branca
Consultar BHT per predicció
Predicció és Presa?
Sí
Consultar BTB
No
Miss en BTB
No
Sí
Modificar PC amb objectiu
Predir No Presa
Next PC
Instrucció de branca
Consultar BHT per predicció
Valid flag és negatiu?
Sí
No
Predicció és Presa?
Aplicar BTFN (predicció estàtica)
Sí
No
Modificar PC amb suma (immediat + PC actual)
Modificar PC amb suma (instr_size + PC actual)
Next PC

Figure 4.6: Comparison of Branch Prediction Unit (BPU) Behavior of CVA6 and gem5 for a Branch Instruction

This difference can lead to significant errors (likely resulting in a loss of IPC) in modeling the CVA6, as a branch instruction in gem5 depends on a BTB access (which has a certain probability of failure), whereas the CVA6 relies solely on the BHT for prediction (the branch target is provided by instruction scanning in the Instruction Fetch phase, see Figure 4.3). If there is a BTB miss in gem5, the static prediction is *Not Taken*, whereas the static prediction in RISC-V (and Ariane) is *Backward Taken, Forward Not Taken* (BTFN). Additionally, the CVA6 BHT includes a validity bit, while the gem5 BHT is initialized to 0. This results in an initial *Not Taken* prediction when, in reality, BTFN is applied for a false validity bit.

# 5. Testing environment

## 5.1 Overview of the Benchmarking Directory

The FPGA benchmark environment is located within the `gem5` directory, specifically in the `util/fpga_bench` subdirectory. This directory contains tools and scripts designed to automate the execution of benchmarks for performance evaluation on FPGA-based systems. Below is a detailed description of the contents of the `util/fpga_bench` directory, as revealed by the `ls` command:

- `binaries`: A directory containing all the executable binaries to be tested.
- `exec_gem5.sh`: A shell script for automating the execution of gem5 simulations.
- `exec_perf_genesys.sh`: A shell script for automated execution of benchmarks on an FPGA, collecting performance metrics.
- `out`: The output directory where benchmark results are organized based on the scripts' execution.

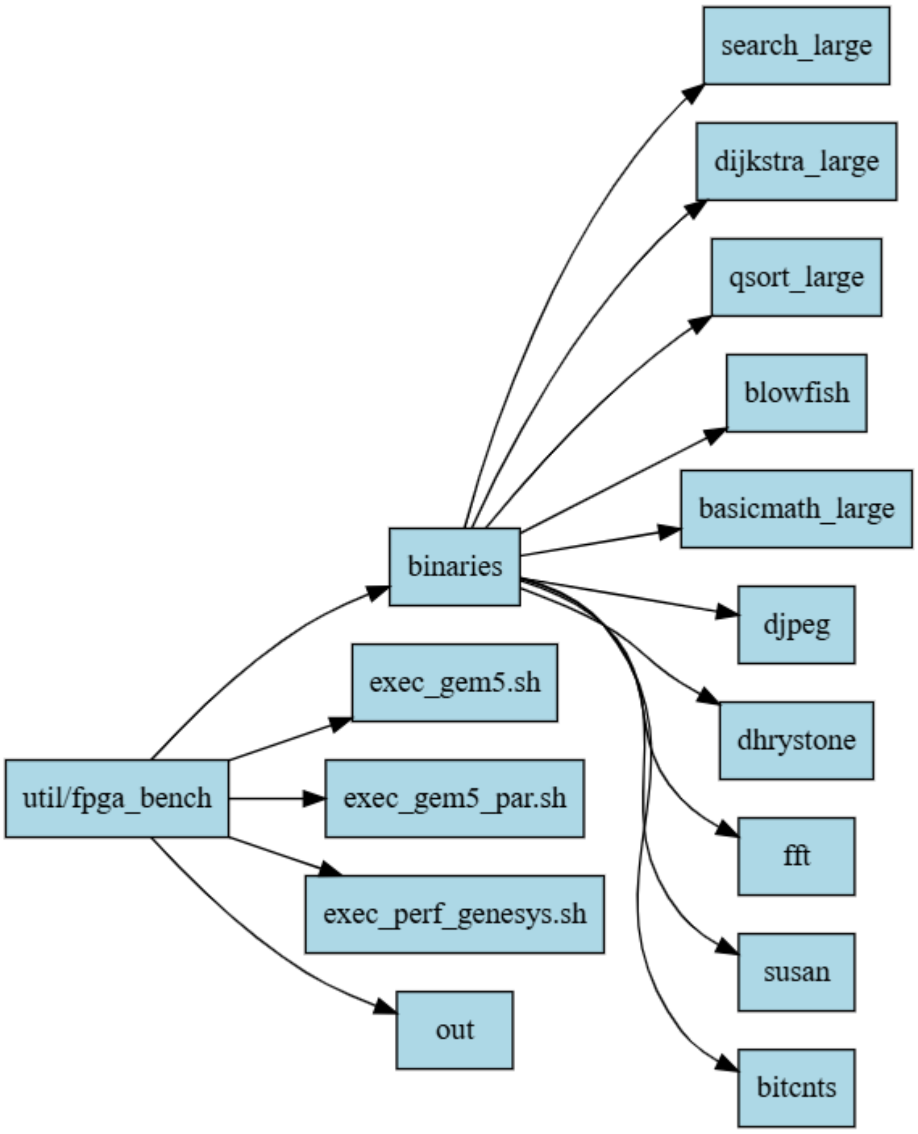


Figure 5.1: Directory structure of `util/fpga_bench`

## 5.2 Detailed Description of Directory Contents

### 5.2.1 binaries

The `binaries` directory contains all the executable files for the benchmarks to be tested. These binaries correspond to various workloads from the MiBench benchmark suite and to the dhrystone benchmark:

- `search_large`
- `dijkstra_large`
- `qsort_large`
- `bf`
- `basicmath_large`
- `djpeg`
- `dhrystone`
- `fft`
- `susan`
- `bitcnts`

Each binary is executed with specific arguments as defined in the automation scripts, and the outputs are stored in the `out` directory.

### 5.2.2 Gem5 execution script

The `exec_gem5.sh` script automates the execution of gem5 simulations for a subset of benchmarks. It is written in a POSIX-compliant shell script. The script performs the following tasks:

- Defines paths to the root directory, gem5 build directory, simulation configuration file, and benchmark directories.
- Specifies a list of benchmarks to execute.
- Creates an output directory structure under `out` for each benchmark.

- Modifies the gem5 simulation configuration file (`simulation.py`) dynamically to set the binary path and arguments.
- Executes the gem5 simulator (`gem5.fast`) with the modified configuration.
- Handles special cases for benchmarks like `dijkstra_large`, `susan`, `qsort_large`, `blowfish`, and `jpeg`, including custom output directory naming and file movement (e.g., `*.pgm`, `*.enc`, `*.ppm`).

### 5.2.3 FPGA execution script

The `exec_perf_genesys.sh` script automates benchmark execution on an FPGA, collecting performance metrics using the `perf` tool. It is also written in a POSIX-compliant shell script. The functionalities are the same as the gem5 execution script, except that `perf stat` is used for executing the benchmarks. Initially, the aim was to collect program instructions and cycles, cache references and mispredictions. Unluckily, the only metrics that were gathered were instructions and cycles.

### 5.2.4 Output directory structure

Once both execution scripts are launched, the structure of the output directory once the program finishes is the following: Each benchmark contains a directory.

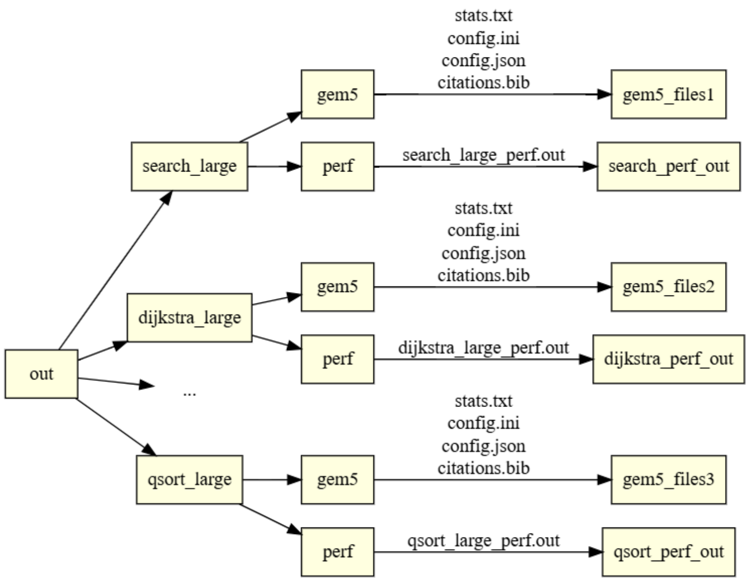


Figure 5.2: Directory structure of `util/fpga_bench/out`

Within it, there are two different directories dedicated to FPGA and gem5 outputs.

## 5.3 Metrics Parser Script

A Python script is used to parse performance metrics from the `stats.txt` files generated by gem5 simulations, located in the `gem5` subdirectories of each benchmark's output directory. The script extracts specific performance metrics, performs calculations to derive additional metrics, and outputs the results to a CSV file named `simulation_metrics.csv`. The extracted metrics are the following:

- **Branch prediction metrics** (`mispredictions`, `btb_hit_ratio`, `cond_incorrect`, `cond_predicted`).
- **Cache performance metrics** (`l1d_cache_misses`, `l1i_cache_misses`, `l1d_miss_latency`, `l1i_miss_latency`).
- **Instruction type counts** (`float_add`, `int_alu`, `mem_read`, etc.).
- **Performance metrics** (`ipc`, `num_cycles`, `num_instructions`).

The derived metrics are the following:

- **Total Floating-Point Operations**: Sums counts of floating-point instruction types (`float_add`, `float_cmp`, etc.).
- **Total Memory Accesses**: Sums counts of memory-related instructions (`mem_read`, `mem_write`, `float_mem_read`, `float_mem_write`).
- **Total Cache Misses**: Sums L1 data and instruction cache misses.
- **Total Miss Latency**: Sums L1 data and instruction cache miss latencies.
- **Misprediction Rate**: Calculates the percentage of conditional branch mispredictions (`cond_incorrect / cond_predicted` * 100).
- **Cache Miss Cycle Percentage**: Calculates the percentage of cycles lost due to cache misses (`total_miss_latency / num_cycles` * 100).
- **Instruction Type Percentages**: Calculates the percentage of each instruction type relative to the total number of instructions.

## 5.4 FPGA Hardware: Digilent Genesys 2

The FPGA team utilized the Digilent Genesys 2 board to execute the benchmarks described in the testing environment. It is used to run the benchmarks with the CVA6 bitstream provided by the OpenHW repository. The choice of a POSIX-compliant shell script (rather than Bash) was driven by compatibility constraints of the FPGA's software environment. Among all of its features, the most important for this project is that it uses a 1 GB DDR3 SODIMM. The memory that resembles the most in gem5 is the `SingleChannelDDR3_1600` which has 200 MT/s less than than the FPGA.

## 5.5 CVA6 bitstream

The CVA6 as we know it today has different configurations. This makes the CVA6 highly adaptable to different applications. In this project, the `cv64a6_imafdc_sv39_config_pkg.sv` configuration from the OpenHW repository will be used for generating the bitstream for FPGA testing and also as a reference to fix certain parameters of our model.

# 6. Procedures and final model

## 6.1 Performance Focus and Metric Limitations

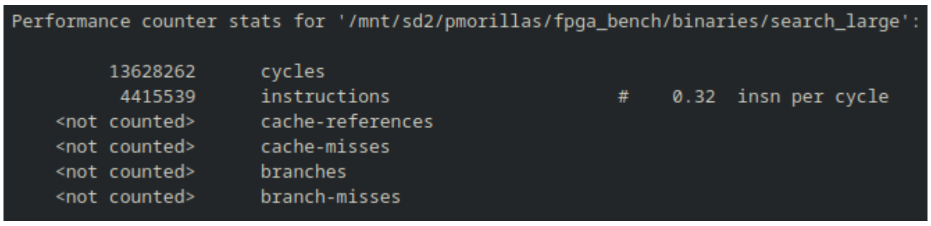
In the process of refining the gem5 model for CVA6, we relied on key insights provided by the original authors of the core. According to their publications, the primary contributors to IPC (Instructions Per Cycle) degradation in CVA6 were *mispredicted branches* and *cache misses*, with a particular emphasis on *data cache loads*. These loads often introduced stalls in the pipeline due to dependencies with subsequent instructions, thus significantly impacting the core's effective throughput.

Based on this, our hypothesis was that accurately modelling the branch prediction unit and the memory hierarchy—especially the data cache behavior—would be critical to reducing the performance modelling error. These components were therefore prioritized in our analysis and tuning efforts.

However, the performance monitoring infrastructure available in our FPGA environment proved to be limited. The only performance counters exposed were the total number of *instructions executed* and the total number of *cycles spent*. This constrained our ability to perform a granular performance correlation between the RTL and the gem5 model.

If we had all the metrics, we could easily tune the BHT in order to obtain a similar prediction rate compared to the CVA6. The procedure would be similar to the cache, since gem5 has a fixed block size of 64B for the `minorCPU` caches and the CVA6 has a block size of 128B for both instruction and data caches, and having cache misses metrics would show if this difference was tolerable or a measure to mitigate it was needed.

As a result, we were compelled to rely exclusively on the *Instructions Per Cycle* (IPC) metric as our primary (and only) point of reference for model calibration.



```
Performance counter stats for '/mnt/sd2/pmorillas/fpga_bench/binaries/search_large':

        13628262      cycles
         4415539      instructions              #    0.32  insn per cycle
   <not counted>      cache-references
   <not counted>      cache-misses
   <not counted>      branches
   <not counted>      branch-misses
```

Figure 6.1: FPGA output from `search_large` benchmark

## 6.2 Component Classification in gem5 Modelling of CVA6

The initial phase of modelling the CVA6 core within the gem5 simulator involved a detailed analysis of the architectural and behavioural correspondence between the CVA6 RTL implementation [Zaruba and Benini 2019a], the specifications from [Zurich and of Bologna] and gem5's internal components. To accurately reflect the microarchitectural structure of the CVA6 core, it was crucial to categorize the simulator's modules into two main classes: *cornerstone components* and *variable components*. Cornerstone components are defined as those fixed architectural elements whose behaviour in gem5 closely matched the CVA6 specifications. In contrast, variable components were those for which the behaviour or structure in gem5 diverged significantly from that of CVA6. These differences could be due to architectural abstractions, simplifications in gem5, or intrinsic differences in implementation logic.

The cornerstone components are the following:

- **Caches**. Minimal error is assumed due to different block size. The size of the cache and the associativity is configurable to equalize the CVA6 caches. On one hand, [Zurich and of Bologna] states that the instruction cache has a size of 16kB and is 4-way associative. Also, it has a latency of 2 cycles. On the other hand, the data cache has a size of 32kB and is 8-way associative. The latency is the same as the instruction cache.

- **Input buffers**. As long as instruction traffic is ensured, increasing the size of the stage input buffers has negligible change. According to the RTL, the input buffer of the decode stage has 2 entries. Since the issue stage is contained into the execute stage, we must set the input buffer size of the execute stage to the number of entries of the scoreboard. That is, the execute stage buffer has 8 entries.

- **Stage latencies**. We have fixed latencies for each stage. That is, 1 cycle to go from one stage to the next one. The execute stage latency is determined by the latency of their functional units (FU), which comprises a set of instructions. We add 2 cycles of latency for each FU latency since issue and commit stages from the CVA6 are comprised into a single stage from gem5 minorCPU model. The FU latencies has been taken from the documentation and from the RTL code.

- **Prediction delay**. It takes one cycle to send the prediction to the previous stage. In other words and taking CVA6 stage naming as reference, the PC generation stage does not receive the prediction in the same cycle as the Instruction Fetch stage from the previous instruction.

- **Posterior head queue analysis**. The only stage that can process other entries apart from the head is the Instruction Fetch queue, since they process the other fetched instruction speculatively (provided that two 16-bit instructions are fetched) when the first instruction is a branch instruction and takes one cycle to predict the outcome.

- **Input width**. We can fetch up to two instructions, so we can send up to two instructions from the instruction fetch stage to the decode stage. However, the remaining stages processes instructions one by one.

- **RAS**. The RAS has two entries.

- **Issue limit**. The processor is single issue. However, gem5 differentiate memory instructions from the ones that are not. It is unclear if memory instructions are a subset of the instructions that you can issue or not. No testing has been done on this component since it had low priority compared to other components. However, we set both limits to 1, since no limit can be set to 0.

- **Store and load buffers**. According to the RTL, the store buffer has 8 entries and the load buffer has 2.

The variable components are the following:

- **BHT**. The BHT had significant differences. We plan on increasing BHT entries in order to mitigate the absence of static prediction. [Zurich and of Bologna] reports a 5% misprediction rate for the Dhrystone benchmark. Besides, we will reduce mispredictions rate because, even on a big-sized BTB, the miss ratio is high on some benchmarks. Hence, the BHT has 512 entries, even if the CVA6 has 128 entries.

- **BTB**. Even if the CVA6 has a BTB for the unconditional indirect branches, we assumed that, given the fact that a taken predict requires a BTB check, a much better approach is to increase the size of the CVA6 BTB. Creating an ideal BTB is discarded because we would ignore unconditional indirect branches, which also takes part in performance statistics. Hence, we plan on doubling the BTB size to reduce misses, while taking 20 bits from the PC after shifting them 2 bits to avoid aliasing.

- **Instruction commit**. The CVA6 commits up to two instructions in a single cycle to avoid artificial starvation. However, as for issue limit, we commit memory instructions and the rest of instructions. There are different possibilities. For instance, setting the commit limit for no memory instructions to 1 and for memory instructions to 1 could have as a consequence that, even if up to two instructions could be committed, just one instruction would be if most of the instructions do not access to memory, when the CVA6 could commit, in that cycle, 2 instructions that do not access to memory. On the other hand, for a commit limit of 2 for both types of instructions, we could be committing up to 4 instructions in a single cycle, which is far more than the instructions that the CVA6 can commit. We choose a middle-point between these two possibilities, setting the commit limit of memory instructions to 1 and the commit limit of the rest of instructions to 2.

The next section gathers all the parameters that configures each component and explains in detail the choosen value.

## 6.3 gem5 minorCPU configuration for the CVA6

### 6.3.1 Core configuration

Table 6.1: CVA6 Core Configuration Parameters

| **Parameter** | **Value** | **Description** |
|---|---|---|
| `fetch1LineSnapWidth` | 4 | Fetch granularity in bytes per cycle |
| `fetch1LineWidth` | 4 | Number of instructions fetched per line |
| `fetch1FetchLimit` | 1 | Max fetch operations per cycle |
| `fetch1ToFetch2ForwardDelay` | 1 | Forward latency to next pipeline stage |
| `fetch1ToFetch2BackwardDelay` | 1 | Backward feedback delay from fetch2 to fetch1 |
| `fetch2InputBufferSize` | 2 | Size of the input buffer at fetch2 stage |
| `fetch2ToDecodeForwardDelay` | 1 | Delay from fetch2 stage to decode stage |
| `fetch2CycleInput` | TRUE | Whether fetch2 can take more than one entry in its input |
| `decodeInputBufferSize` | 2 | Number of instructions buffered before decode |
| `decodeToExecuteForwardDelay` | 1 | Delay from decode to execute |
| `decodeInputWidth` | 2 | Instructions decoded per cycle |
| `decodeCycleInput` | FALSE | Whether decode can take more than one entry in its input |
| `executeInputWidth` | 8 | Instructions accepted by execute stage per cycle |
| `executeCycleInput` | FALSE | Whether execute can take more than one entry in its input |
| `executeInputBufferSize` | 8 | Buffer size before the execute stage |
| `executeIssueLimit` | 1 | Maximum number of instructions issued per cycle |
| `executeMemoryIssueLimit` | 1 | Max memory instructions issued per cycle |
| `executeCommitLimit` | 2 | Instructions committed per cycle |
| `executeMemoryCommitLimit` | 1 | Memory commits per cycle |
| `executeMaxAccessesInMemory` | 1 | Maximum parallel memory accesses |
| `executeLSQMaxStoreBufferStoresPerCycle` | 1 | Store buffer stores allowed per cycle |
| `executeLSQRequestsQueueSize` | 2 | Capacity of the LSQ request queue |
| `executeLSQTransfersQueueSize` | 2 | Size of the queue used for memory transfers |
| `executeLSQStoreBufferSize` | 8 | Store buffer size in the LSQ |
| `executeBranchDelay` | 1 | Cycles to resolve a branch |

### 6.3.2 Functional unit configuration

As stated before, we add a cycle latency of 2 to all the FU to simulate a separate stage for the issue and commit phases. Functional units for the minorCPU model work with fixed latencies. However, integer division has variable latency that ranges from 2 to 64 cycles. The solution provided is to apply an average latency of 35 (average of 2+2 and 64+2) for this instructions. Besides, the integer divider is not pipelined. This has de side-effect that we can't pipeline the issue and commit stages, since both are integrated in the execute stage. However, this has no real effect on IPC, since integer division instruction rate is minimal, probably due to the optimization procedures of the compiler.

Table 6.2: CVA6 Functional Units, Supported Instructions and Latencies

| **Functional Unit** | **Instruction Class** | **Latency (cycles)** | **Pipelined** |
|---|---|---|---|
| Simple Integer ALU | `IntAlu` | 3 | Yes |
| Complex Integer Multiplier | `IntMult` | 4 | Yes |
| Complex Integer Divider | `IntDiv` | 35 | No |
| Floating-Point | `FloatAdd` | 3 | Yes |
| | `FloatCmp` | 5 | Yes |
| | `FloatCvt` | 4 | Yes |
| | `FloatDiv` | 4 | No |
| | `FloatSqrt` | 4 | Yes |
| | `FloatMult` | 3 | Yes |
| Load/Store Unit | `MemRead` | 3 | Yes |
| | `MemWrite` | 3 | Yes |

### 6.3.3 Branch predictor unit configuration

Increasing BTB associativity does not prevent BTB miss rate in the benchmarks observed, however, it is known that high associativity prevents check misses. We also retain 20 bits to avoid aliasing (i.e. more than one instruction is indexed in the same line). Local predictor size granularity is in bits and then divided by the size of the counter (`BHTlocalControlBits`). Hence, we multiply the size by 2.

Table 6.3: CVA6 Branch Predictor Parameters

| **Parameter** | **Value** |
|---|---|
| Predictor type | Local BP |
| `BTBnumEntries` | 64 |
| `BTBassociativity` | 16 |
| `BTBtagBits` | 20 |
| `ReturnAddrStack(numEntries)` | 2 |
| `localPredictorSize` | 512 * 2 |
| `BHTlocalCtrBits` | 2 |
| `BHTinstShiftAmt` | 2 |
| `BTBinstShiftAmt` | 2 |
| `BTBReplPolicy` | LRU |

### 6.3.4 Cache configuration

Caches are configured as stated before. MSHR has a slight effect on IPC when increased in data cache. Targets per MSHR (`tgts_per_mshr`) has no effect on IPC, so we set it to 1, since no information has been found in this parameter.

Table 6.4: CVA6 L1 Cache Parameters

| **Parameter** | **L1 Instruction (L1I)** | **L1 Data (L1D)** |
|---|---|---|
| `size` | 16 kB | 32 kB |
| `associativity` | 4-way | 8-way |
| `tag_latency` | 1 cyc | 1 cyc |
| `data_latency` | 2 cyc | 2 cyc |
| `response_latency` | 2 cyc | 2 cyc |
| `mshrs` | 4 | 2 |
| `tgts_per_mshr` | 1 | 1 |
| `write_buffers` | — | 8 |
| `is_read_only` | TRUE | FALSE |
| `writeback_clean` | TRUE | |

# 7. Results and Model Validation

This chapter evaluates the accuracy of our gem5-based CVA6 model by comparing simulated IPC values with those obtained from an FPGA implementation of the CVA6 core. As discussed previously, the only performance metrics exposed by the FPGA were the number of instructions executed and total execution cycles, from which the IPC (Instructions Per Cycle) was derived. Therefore, IPC is our sole quantitative metric for validating the simulation fidelity.

## 7.1 IPC Comparison

Table 7.1 compares the IPC values from the gem5 simulation and the CVA6 FPGA prototype for each benchmark. The absolute difference and relative error are also shown.

Table 7.1: Simulated vs. Measured IPC per Benchmark

| **Benchmark** | **IPC (gem5)** | **IPC (FPGA)** | **Absolute Error** | **Relative Error (%)** |
|---|---|---|---|---|
| `basicmath_large` | 0.33778 | 0.30526 | 0.03252 | 10.65% |
| `search_large` | 0.34617 | 0.32400 | 0.02217 | 6.84% |
| `dijkstra_large` | 0.37593 | 0.40037 | 0.02444 | 6.10% |
| `fft` | 0.36149 | 0.52000 | 0.15851 | 30.48% |
| `bitcnts` | 0.43060 | 0.77480 | 0.34420 | 44.42% |
| `dhrystone` | 0.43375 | 0.63546 | 0.20171 | 31.73% |
| `susan` | 0.33035 | 0.52589 | 0.19554 | 37.18% |
| `blowfish` | 0.39455 | 0.42606 | 0.03151 | 7.39% |
| `qsort_large` | 0.34048 | 0.38920 | 0.04872 | 12.52% |
| `djpeg` | 0.33047 | 0.30733 | 0.02314 | 7.53% |

The simulated IPC tracks the FPGA-measured IPC reasonably well for most workloads, staying within a 10% error margin for 6 out of 10 benchmarks. However, some significant discrepancies arise, which we analyze next.

## 7.2 Benchmark Analysis

**`basicmath_large`.** This benchmark shows a moderate overestimation of IPC by 10%. The high branch misprediction rate (45%) and low BTB hit ratio (25%). Besides, it has a low overall cache miss latency.

**`search_large`.** The gem5 model is within 7% of the FPGA result. Despite modest cache misses, mispredicted branches (25%) still penalize IPC. Like before, verall cache miss latency is low (2%).

**`dijkstra_large`.** Gem5 underestimates IPC by about 6%. Dijkstra's behavior is memory-heavy. Even though cache miss counts are not extremely high, and misprediction rate is 2%, while overall cache miss latency is 3%.

**`fft`.** This workload shows a significant underestimation ( 30%). FFT combines a high floating-point operation rate with a moderate BTB miss rate (42%) and mispredicted branches (33%). Overall cache miss latency is almost 3%.

**`bitcnts`.** A large gap ( 44%) exists here. The bast majority of instructions uses the alu (91%), while memory accesses is low. Thus, cache pressure is also very low. Misprediction ratio is 3%, but BTB hit ratio is 51%.

**`dhrystone`.** This is another classic benchmark where the gem5 model underperforms by 31%. The simulation shows low misprediction and miss rates. However, we note a high BTB miss ratio (60%).

**`susan`.** IPC is 37% too low. Like `dhrystone`, this is a compute-heavy benchmark with low memory intensity. We also assign the underestimation to the architectural structure of the model's core and to the BTB miss ratio, which is 60%. The percentage of lost cycles due to cache misses is high (27%).

**`blowfish`.** This workload is modeled accurately within 7%. BTB miss ratio is 60%. The percentage of lost cycles due to cache misses is moderate (11%).

**`qsort_large`.** The simulation underestimates IPC by about 12.5%. The report shows a 31% of misprediction rate and a very low hit ratio of 27%.

**`djpeg`.** IPC is overestimated by 7.5%. It has a 60% of BTB miss ratio, The percentage of lost cycles due to cache misses is high (24%).

## 7.3 Summary and Outlook

Most of IPC losses of the model are attributed to BTB misses that trigger a default and incorrect *Not taken* prediction. However, an ideal BTB would inevitably increase the IPC of all the benchmarks, increasing some of their already low relative error. It is yet to be determined the percentage of branch instructions to know the real weigh of IPC loses due to mispredicted or BTB-missed branches. If improving the BPU with a better BTB reduces the error of the high relative error benchmarks while maintaining the other low relative errors, then we confirm that a better BPU is needed to have a more accurate model.

An experiment on some of the benchmarks with higher error has shown that, in a subset of them, a better BPU does not decrease IPC loses. Furthermore, benchmarks that have shown a very low number of lost cycles due to cache misses are providing the highest relative error. For our rather simple system (which comprises a CVA6 core, instructions and data L1 caches, and memory) we can conclude that the relative error is coming from the architectural configuration of the core. Some aspects are discussed in the next chapter.

Table 7.2: BPU impact on `fft 4 4096`

| **BPU Config** | BTBHit | Mispred | IPC$_{sp}$ |
|---|---|---|---|
| 2bit-128, BTB128, RAS2 | 41% | 29% | — |
| 2bit-512, BTB2K, RAS2 | 76% | 13% | 14% |
| 2bit-1K, BTB8K, RAS2 | 76% | 12% | 17% |

Table 7.3: BPU impact on `bitcnts 10000`

| **BPU Config** | BTBHit | Mispred | IPC$_{sp}$ |
|---|---|---|---|
| 2bit-128, BTB128, RAS2 | 59% | 3% | — |
| 2bit-512, BTB2K, RAS2 | 59% | 3% | 0% |
| 2bit-1K, BTB8K, RAS2 | 59% | 3% | 0% |

Table 7.4: BPU impact on `dhrystone 2000`

| **BPU Config** | BTBHit | Mispred | IPC$_{sp}$ |
|---|---|---|---|
| 2bit-128, BTB128, RAS2 | 41% | 7% | — |
| 2bit-512, BTB2K, RAS2 | 42% | 7% | 0% |
| 2bit-1K, BTB8K, RAS2 | 42% | 6% | 0% |

# 8. Future work

This project has yielded a gem5 configuration that approximates the CVA6 microarchitecture with reasonable fidelity, particularly when IPC is the sole available metric. However, several directions remain open to further refine the accuracy, expand the insight, and explore modelling techniques that better reflect CVA6's behaviour. These opportunities fall into architectural analysis, simulator internals, and hardware instrumentation.

## 8.1 Deeper Analysis of CVA6 and gem5 Source Code

While the present modelling used both documentation and RTL-level observations, a more exhaustive reading of the CVA6 RTL could expose more nuanced behaviours not yet captured in gem5. Similarly, a deeper inspection of gem5's `minorCPU` source code (e.g. `memoryIssueLimit` and `memoryCommitLimit`) could resolve ambiguity and guide a more faithful configuration. For example, in the CVA6 RTL code, the store buffer queue makes a separation between store instruction that are being committed and the ones that have been speculatively executed. This may lead to new modifications to mitigate the differences between both implementations.

## 8.2 Handling Fixed Cache Block Sizes

The current gem5 cache model enforces a fixed block size of 64 bytes, while CVA6 uses 128-byte blocks. This mismatch may understate cache efficiency. Modifying or extending the gem5 cache subsystem to support configurable block sizes would provide more accurate simulation of CVA6's cache hierarchy.

## 8.3 Cope with different Memory Models

Our gem5 model supports a memory that is 200MT/s slower than our FPGA memory, leading to unavoidable IPC loses due to a slower response latency under a cache miss.

## 8.4 Instruction Compression and Dual Fetch

An important yet unresolved modelling challenge lies in instruction fetch behaviour. CVA6 supports dual fetching of compressed 16-bit instructions, effectively increasing fetch throughput. It is currently unclear if `minorCPU` exploits this feature. The `fetchLineWidth` parameter offers one way to emulate this behavior. For instance, increasing the `fetchLineWidth` from 4 to 8 showed an IPC improvement of 20% in `bitcnts`, reducing the error from 44% to 24%.

This suggests a promising strategy of *split modeling*: using one configuration (e.g., high `fetchLineWidth`) for benchmarks where the CVA6 is likely exploiting dual fetching, and a different, more conservative configuration for others.

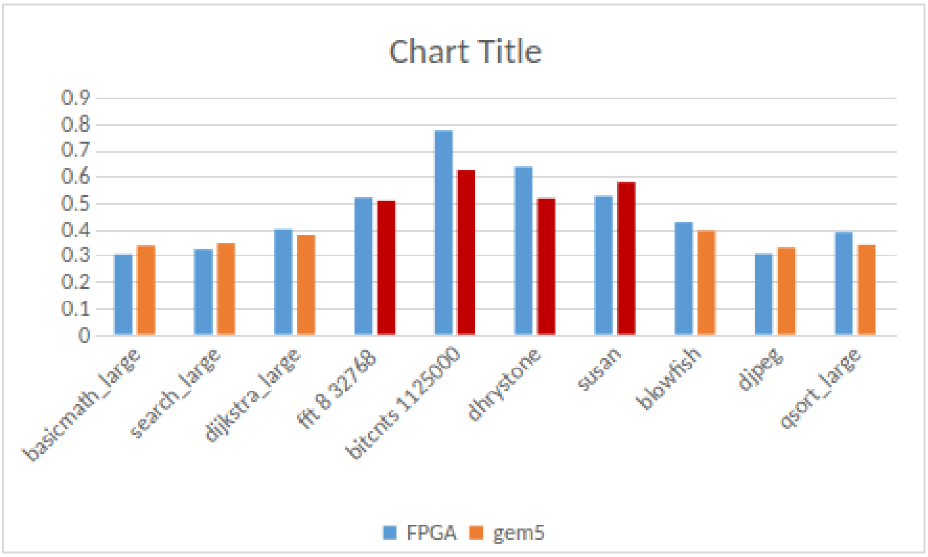


Figure 8.1: Bar plot comparing FPGA and gem5 IPCs. The red bar indicates the dual fetching model IPC, while the orange ones indicates the single fetching model. The average IPC relative error is 10.4%

## 8.5 Source-Level Modifications in gem5

The previous suggestions inevitably lead to the possibility to modify the source code of gem5. Another specific code modification example is disabling the BTB check when a prediction is already taken, which diverges from how CVA6 handles branches. This would require modifying the predictor logic in the gem5 source but could significantly improve branch predictor behaviour for every type of branch instruction.

## 8.6 Re-evaluating the Cornerstone/Variable Classification

Initial modelling distinguished between cornerstone and variable components. However, some components might benefit from being treated as variable, particularly when applying the split modelling approach. This could allow more accuracy even if those units were initially assumed to be fixed.

## 8.7 Compiler-Based Exploration

Another line of investigation involves controlling the compiler to test how ISA-level changes impact the modeled system. For instance, disabling instruction compression could simplify the fetch model and clarify its behavior, but at the cost of modeling fidelity—since compressed instructions are a key CVA6 feature. Nonetheless, such experiments could serve as a validation tool: if gem5 accuracy improves without compression, it points to fetch decoding as a modelling gap.

## 8.8 Performance Monitoring on FPGA

Perhaps the most impactful improvement lies in enhancing the performance monitoring infrastructure on the FPGA. The current limitation to cycle and instruction counters restricts validation scope. Implementing a proper Performance Monitoring Unit (PMU) capable of capturing mispredictions, cache hits/misses, and memory access latency would dramatically increase the resolution and confidence in simulator calibration. This would also allow tuning of components like the BTB, LSQ, or FUs based on real metrics, rather than guesswork.

# 9. Conclusions

Modelling a Linux-capable RISC-V core like CVA6 in an architectural simulator such as gem5 is a non-trivial endeavour. Despite the maturity of both projects, the inherent abstraction levels and design assumptions diverge significantly. Particularly in the case of gem5's `minorCPU`, which most of its functionalities do not match those of the CVA6 core.

The model documented in this memory is the final version of a long chain of different models, which were made based on previous, and wrong, assumptions of the CVA6 architecture. During the CVA6 tuning, we encountered risks that were weighed during the project management phase. For instance, risk 1 (2.6.1), risk 2 (2.6.2) and risk 3 (2.6.3) were problems that happened, and we had to mitigate them as planned initially. However, this inevitably hindered the progress to make further advancements in the project. Given the contracted time, we observed that it was limited.

In addition, we faced other risks that were not predicted. For instance, no FPGA was available during the execution of the project, so we had to rely on the FPGA team from Openchips' Italy headquarters. Since their workload is extensive, we had to make things easier for them and create an environment (see 5) that automates FPGA execution and sorts the outputs. The only tasks that they needed to do was to execute the script, compress the environment folder and send it back. This part added a significant amount of time to the project, making it worth to mention this part of the project.

Another unplanned obstacle was the extensive execution time of the benchmarks. For the large versions of the programs, it took more than, at least, 4h to execute the script completely in the company laptop. Working with the smaller versions of the benchmarks incurred in IPC deviations that ranged from 5% to more than 10% in gem5 comparing them to the larger ones, and even worse deviations on the

FPGA. Since we wanted to work with the most detailed, already limited metrics, we were forced to wait the completion times. Openchip has a server with 48 cores, but discrepancies between UPC and Openchip regarding Intellectual Property and Confidentiality terms impeded the benchmarks to be executed on it. However, the smaller versions were used to do experimental procedures rather than formal comparisons.

All in all, and despite the hindrances of the project, we consider this work a satisfiable final bachelor thesis. In the following sections, we discuss what are the most notable achievements of this project.

## 9.1 Complexity of gem5 Modelling

A key finding of this project is that **gem5 modelling is inherently complex**, not only due to the volume of tunable parameters, but because many components do not directly correspond to their RTL counterparts. The CVA6 core features architectural patterns which are difficult to reproduce faithfully using gem5's `minorCPU` without source-level intervention.

This complexity forced a strategic categorization of components into "cornerstone" and "variable" modules, allowing us to prioritize and iterate more efficiently. However, even components initially considered fixed—such as the fetch unit or input buffering—may need to be treated as variable under more nuanced performance investigations.

## 9.2 A Solid Skeleton for CVA6 Modelling

Despite these difficulties, the resulting configuration establishes a **solid skeleton for CVA6 modelling in gem5**. Through careful calibration against RTL IPC metrics and documentation from the core's authors, we constructed a configuration that reflects key aspects of CVA6's pipeline behaviour, memory hierarchy, and control flow management.

Functional unit latencies, cache sizes, prediction delays, and queue capacities were chosen based on direct RTL analysis and their documented performance impact.

IPC discrepancies were reduced to below 10% for most benchmarks, with further improvements possible through the future work paths outlined in Chapter 8.

## 9.3 A Reliable Benchmarking Environment

Additionally, this work establishes a **robust benchmarking environment for CVA6**, integrating:

- a tuned gem5 configuration targeting CVA6 behavior,
- a consistent set of representative benchmarks (e.g., `dhrystone`, `fft`, `bitcnts`), and a set of scripts to configure, execute and sort the desired experimentation results given the model configuration, while also extracting their corresponding metrics.
- an IPC-based validation framework using FPGA-collected results,
- and a documented methodology for extending, validating, and tuning new microarchitectural features.

While current limitations—such as lack of detailed performance metrics on the FPGA—constrain the granularity of validation, the infrastructure in place is flexible and extensible. It provides a meaningful basis for exploring further architectural changes, evaluating compiler optimizations, or prototyping next-generation cores based on CVA6.

## 9.4 Final Thoughts

This project demonstrates that, even with partial visibility into RTL internals and constrained metrics, gem5 can still be an effective tool for approximating and analysing real-world RISC-V cores like CVA6. With a careful balance of architecture-level insight, source exploration, and benchmark-driven tuning, a faithful and extensible microarchitectural model is achievable—and now exists as a foundation for future development and experimentation.

# Bibliography

Excel W3S tutorial. `https://www.w3schools.com/excel/index.php`. Accessed: 04-03-2025.

GCC documentation. `https://gcc.gnu.org/onlinedocs/gcc-12.4.0/gcc/`. Accessed: 04-03-2025.

Python documenation. `https://docs.python.org/3/`. Accessed: 04-03-2025.

QEMU documentation. `https://www.qemu.org/docs/master/`. Accessed: 04-03-2025.

J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+, 2020. URL `https://arxiv.org/abs/2007.03152`.

P. Ravenelk, A. Perais, B. de Dinechin, and F. Petrot. A gem5-based CVA6 framework for microarchitectural pathfinding. URL `https://api.semanticscholar.org/CorpusID:270493869`.

U. Shahid, A. Ahmad, and S. Wasim. Gem5-based evaluation of CVA6 SoC: Insights into the architectural design. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 298–300, 2024. doi: 10.1109/ISPASS61541.2024.00037.

M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi. Hardware-Validated CPU Performance and Energy Modelling. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 44–53, 2018. doi: 10.1109/ISPASS.2018.00013.

F. Zaruba and L. Benini. CVA6 Git repository. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019a. doi: 10.1109/TVLSI.2019.2926114. URL `https://github.com/openhwgroup/cva6`. Version 2.0.4, released on 2019-07-26, Accessed: 10-06-2025.

F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11): 2629–2640, 2019b. doi: 10.1109/TVLSI.2019.2926114.

E. Zurich and U. of Bologna. Reference management with BibTeX: A short guide. `https://cva6.readthedocs.io/en/latest/#`. Accessed: 26-02-2025.