Verilog Implementation of a Low-cost Vector AI Accelerator and Integration in a RISC-V Processor

---

Author:
Christos Zarkos

Director:
Dr. Leonidas Kosmidis

A Thesis submitted in fulfilment
of the requirements of a Bachelor's Degree
with specialization in Computer Engineering

Facultat d' Informàtica de Barcelona
Universitat Politècnica de Catalunya

Barcelona Supercomputing Center

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya
Barcelona, Spain

January 2023

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

El acelerador SPARROW AI portátil y de bajo costo se propuso y demostró recientemente en VHDL en dos procesadores espaciales, el LEON3 y el NOEL-V. En este trabajo de fin de grado se implementa SPARROW en SystemVerilog y se integra con un procesador RISC-V escrito en SystemVerilog, el SweRV Core EH1. Esta implementación proporciona tres resultados importantes. Demuestra la portabilidad de SPARROW, proporciona una extensión útil a un procesador de grado industrial existente y nos brinda la capacidad de comparar las implementaciones de SPARC y RISC-V. Los resultados obtenidos demuestran que SPARROW puede proporcionar aceleraciones significativas también en el núcleo EH1 de doble emisión.

ABSTRACT

The low-cost and portable SPARROW AI accelerator has been recently proposed and demonstrated in VHDL in two space processors, the LEON3 and NOEL-V. In this Bachelor's thesis, SPARROW is implemented in SystemVerilog and is integrated with a RISC-V processor written in SystemVerilog, the SweRV Core EH1. This implementation provides three important results. It proves the portability of SPARROW, provides a useful extension to an existing industrial grade processor, and give us the ability to compare the SPARC and RISC-V implementations. The obtained results demonstrate that SPARROW can provide significant speedups also in the dual issue EH1 core.

ACKNOWLEDGMENTS

Firstly, I would like to thank my supervisor **Dr. Leonidas Kosmidis** for all he has done for me. First of all, for giving me the opportunity to be part of this very interesting project. Apart from that I want to thank him for giving the direction and guidance I needed throughout the course of this project in order to complete it and also for always been there to help and/or consult me whenever I needed him to. Even regarding matters outside the scope of this thesis. Also, I would like to thank him for giving me the opportunity to participate in the GPU4S project and work at the Barcelona Supercomputing Center in the CAOS group.

I also want to thank **Marc Solé Bonet** for allowing to be a part of this very interesting project, which is in some way his "baby". Furthermore, I want to thank him also for always been eager to help me and explain his work to me.

Finally, I want to thank my **family**. They are the reason I am the person I am today. They have given me everything, my education, and the priviledge to be able to focus on my studies without neeeding to work. Bust most importantly, they all have offered me constant love and support throughout the years. There are my biggest motivation.

1 CONTEXT AND SCOPE

1.1 MOTIVATION

Artificial Intelligence (AI) applications are becoming more and more popular in our times. This is completely justified since the use of AI and Machine Learning has already completely revolutionized many scientific fields. The problem is that AI workloads need a lot of processing power and memory. In terms, of processing power the most popular solutions are accelerators like (GP)GPUs, or other forms of specialized ASICs.

Space processing wants to also take advantage of AI and there already many efforts made by agencies like the European Space Agency (ESA) and the National Aeronautics and Space Adminstration (NASA). The problem is that space processors have many special requirements for high reliability and a long and expensive process before they are qualified. For this reason, they are based on older, mature and well-established technologies, which are proven-in-use. For these reasons and to avoid the long and expensive process of creating new technologies from scratch, it would be more beneficial to perform minimal modifications on the existing ones.

Commercial off-the-shelf (COTS) accelerators could be of use but they are not designed with radiation hardening, nor with support for real-time operating systems (RTOS).

The needs of space AI workloads could be met using a tightly integrated Single Instruction Multiple Data (SIMD) AI accelerator that would extend an already existing space processor. This is exactly the capability SPARROW offers, although in this thesis the host core is not designed for space. However, it has interesting features which make it a good candidate for space use, such as Error Correction Code (ECC) protection and it is an industrial-grade, fully verified processor used in the commercial sector, which increases the confidence on its correctness.

The main goals for this project are to prove the portability of SPARROW, have the ability to compare the different implementations, draw conclusions and expand on the capabilities of SPARROW either directly from this project or indirectly using its conclusions.

1.1.1 *Context*

This is a Bachelor's Thesis of the Computer Science Degree, in the specialization of Computing Engineering, in the Faculty of Informatics of Barcelona of the Polytechnic University of Catalonia, directed by Dr. Leonidas Kosmidis. The work of this thesis has been per-

formed in the Barcelona Supercomputing Center (BSC) and also contributed in the GPU4S project, in which SPARROW was first implemented [1]. The code and full implementation of this work can be found here [2]. The standalone SystemVerilog implementation of SPARROW can be found here [3]

1.1.2 *Concepts*

In this section I mention the key concepts and background knowledge required in order to fully comprehend the contents of this thesis. In addition, I provide a small summary and/or reminder for each one, along with references to bibliography in case the reader wants more information about a specific topic.

SPARROW[4] is a light, low-cost and portable AI accelerator designed to be integrated with space processors (LEON3, NOEL-V). It is designed by Marc Solé Bonet as a Master's thesis project at FIB, UPC and BSC and it is written in VHDL. The code and full implementation of his work can be found here [5].

RISC-V[6] is an open-source hardware instruction set architecture (ISA), based on established reduced instruction set computer (RISC) principles. Although it is an academic design, the designers planned that the RISC-V ISA be usable for practical computers. Hardware designed based on the RISC-V ISA tends to be less complex to other architectures, like x86, due to the (much) fewer possible instructions. RISC-V supports many different extensions, which can be combined based on the needs of the target domain in which the processor is going to be used. In addition, RISC-V core implementations which rely on standardized extensions, are compatible with each other, so they can leverage the same software ecosystem.

Dual-issue[7] is when processors issue certain pairs of instructions simultaneously in order to increase instruction througput. When this happens, the instruction with the smaller cycle count is assumed to execute in zero cycles. Processors that support dual-issue potentially have the capability of committing 2 instructions per cycle, at least a good percentage of the time. Because dual-issue processors can execute multiple instructions simultaneously at the same clock cycle, they are also considered superscalar.

Single instruction, multiple data (SIMD)[8] is a type of parallel processing in Flynn's taxonomy. SIMD can be internal (part of the hardware design) and it can be directly accessible through an instruction set architecture (ISA), but it should not be confused with an ISA. SIMD describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously.

Vector processing is when a central processing unit can perform complete operations on a vector input in an individual instruction.

An **AI accelerator** is a dedicated hardware module designed to accelerate machine learning computations. Machine learning, and particularly its subset, deep learning is

primarily composed of a large number of linear algebra computations, which can be easily parallelized. AI accelerators are designed to improve performance and reduce the latency of machine learning based applications by accelerating these linear algebra operations.

SweRV Cores [9] EH1, EH2 and EL2, are three 32-bit RISC-V cores developed by Western digital. They are open source in the CHIPS Alliance github and they will be used in a multitud of projects by Western Digital in the future. SweRV Core EH1 will be used in this thesis. Recently they were renamed to Veer Cores, but because when I started the work they were named SweRV, I will refer to the core I am using as SweRV Core EH1 in this thesis, instead of VeeR Core EH1.

Lagarto[10] is the first academic RISC-V core implemented in silicon and also the first chip with an open source ISA developed in Spain. The source code of the processor is not yet open source but it will be in the (near) future. It is the result of a project led by the Barcelona Supercomputing Center (BSC).

Hardware emulation is when one piece of hardware, most commonly an FPGA imitates the behavior of another piece of hardware. Logically, this is very useful when developing a project for debugging and functional verification.

RTL (Register Transfer Level) is a design abstraction, in modern digital design, used for defining the digital portions of a design. It is based on synchronous logic and contains three pieces, registers, combinatorial logic and clock. It is usually captured using a hardware description language such as (System) Verilog and VHDL.

1.1.3 *Stakeholders*

There are various stakeholders in this project, both direct and indirect.

Direct

Dr. Leonidas Kosmidis who is the director of my thesis.

Marc Solé Bonet who implemented SPARROW in VHDL and it was his Master's thesis.

Myself, because this project is going to help me get my degree, build my resume but also get the valuable engineering and project management experience for my future.

Indirect

European Space Agency because the processor in which SPARROW will be integrated has the potential to be used in a real space system and is a proven industry-grade processor. Also, the existence of this project will provide an extra level of versatility to the existing options for space processors, offering an extra solution for AI acceleration for RISC-V processors written in SystemVerilog, and more importantly it will

be a step towards "European Independence in space" which is something that ESA strives to achieve.

Western Digital because one of my implementations will be on a western digital processor and the company can afterwards use it freely.

RISC-V CHIPS Alliance because my implementation will involve only RISC-V processors.

Barcelona Supercomputing Center, in the same way as Western Digital, if in the end SPARROW will be also integrated with Lagarto which is a BSC creation.

Since the project is open-source, anyone else who would like to use a low-cost effective RISC-V processor with AI acceleration.

1.2 JUSTIFICATION

1.2.1 *Previous Work*

In recent years, the advancements and applications of AI technology are vast and rapid. Thus, there has been an growing interest and demand in AI acceleration in terms of hardware and more specifically there is the notion that RISC-V chips with AI accelerators will be very popular in the near future. Already, there are convincing examples, such as the work of the company **esperanto**[11] that is focusing on high performance acceleration and also opensource project like **Ztachip**[12] that is a RISC-V AI accelerator for vision and AI edge applications running on low-end FPGA devices. Of course, there is also the VHDL version of **SPARROW** that this thesis is based upon. The differentiating factor of SPARROW compared to other existing projects out there is that it is low-cost, energy-efficient and it is suitable for use in space processors. Space processors have requirements and are very difficult to get verified so an improved version of them is more useful in terms of time and cost than making a new processor.

Another work that is related to SPARROW and my thesis is the RISC-V **P-extension proposal** [13], which is a proposal for introducing packed-SIMD instructions as an instruction set extension. Unlike many other SIMD instruction set extensions, the P-extension makes use of general purpose registers to perform the SIMD operations on, rather than dedicated registers. The utilization of the register file is something very innovative and useful and is something that SPARROW also does in order to achieve such low cost and easy integration within a CPU.

1.2.2 *Justification of my work*

First of all, the project is an AI accelerator, so the first reason of its creation is to include AI acceleration into an existing industry-graded processor such as the SweRV Core EH1

by Western Digital. Also, we may have already an implementation of SPARROW but this one will be different and useful in many ways. First of all, this implementation proves SPARROW's portability. SPARROW is very efficient and low-cost and having such a RISC-V implementation is very important, especially when taking into consideration the fact that in the RISC-V community there is almost nothing to this day that focuses on embedded cores, safety-critical systems and real-time systems. Lastly, we enable the comparison of different implementations and draw useful conclusions.

1.3 SCOPE

1.3.1 *Objectives and sub-objectives*

The main objective of this thesis is implementing the SPARROW AI accelerator and integrate it into a RISC-V processor. But, we can divide the work into smaller sub-objectives, both theoretical and practical, that the completion of each one of them will lead to the successful completion of the whole project.

Theoretical part

a. Understand SPARROW (both the design and the implementation).

b. Master my understanding of writing RTL design .

c. Understand the base processor(s).

Practical part

a. Refresh SystemVerilog knowledge and learn VHDL.

b. Write the code for SPARROW in SystemVerilog.

c. Integrate it to an existing RISC-V processor.

d. Write RISC-V assembly tests for validation.

e. Validate the project using the existing sequential C code tests.

f. Draw conclusions.

1.3.2 *Requirements*

There are some crucial requirements that will guarantee the quality and correctness of the project.

a. Write good, comprehensive and well-structured code with comments. This will help debugging and remembering the logic of code segments in the future.

b. Use a thorough testbench, which is the test program that is run when simulating the design. A good and thorough testbench is a step closer to RTL code that is both simulatable and synthesizable.

c. Use thorough tests for assessing the validity of the project.

1.3.3 *Potential obstacles and risks*

Although I have faith in the completion of this project, there are some potential problems and risks that I may face along the way. Here are the three issues I have recognized as most important:

Time. This thesis must be completed within a semester which is just 4 months, which is a tight time frame and forces the work to be done in a sharp manner because a mistake that costs time can be detrimental to the completion of the project.

Hardware code simulating but not synthesizing. Unfortunately, writing hardware code is not as easy as writing software since when the verification of its correctness is more complex. It is not certain that a hardware program that simulates, will also synthesize. The reasons for this are many, from poor testing to timing and physics related aspects but the bottom line is that it is a very serious parameter of the development of the project that can cause major time-consuming problems.

Inexperience on hardware projects of such scale. Although I am familiar with (System) Verilog and have written programs, which both simulate and synthesize and have emulated them on an FPGA and I have very good theoretical background and knowledge on hardware, I have never implemented a hardware project of such scale before, thus it is going to be a challenge for me.

1.4 METHODOLOGY AND RIGOUR

1.4.1 *Methodology*

The methodology I will use in order to manage the work I will have in this project is the Kanban methodology of the Agile family of methodologies, which is a very famous technique in the industry and focuses in visualizing the work, limiting work in progress, and maximizing efficiency. I will have a Kanban board and the work I will have will be seperated into tasks and each task will be put in one of 4 different columns of the board depending on its state.

The 4 different states are:

- **TODO:** Here are tasks that I have not yet started implementing.
- **DOING:** Here are tasks that I have started implementing but are not completed.

- **TESTING:** Here are tasks that have been completed but not tested in terms of correctness yet.
- **DONE:** Here are task that have been completed and tested and should not be bothered with again.

1.4.2 *Validation*

There will be multiple levels of validating the correctness of the project. First of all, Verilator will be used for simulating the code. This will make the design process faster and easier but when a checkpoint is reached synthesis will be done on a Xilinx FPGA using Xilinx Vivado as the tool for it. The synthesis is very important because as we discussed earlier in the potential risks segment, it is possible that hardware code will simulate successfully and seem correct but not synthesize and thus not actually work! For verifying the correctness of the entire project at the end, we will compare its output with the scalar C version of the benchmarks used for evaluating the VHDL version of SPARROW.

For managing the code of the project, BSC's Gitlab will be used, as per usual in such projects. A repository will be created containing the main code and different branches for testing, making changes and developing. This will help a lot at keeping the work well-organized and backed up.

Last but not least, a weekly meeting will be scheduled with my director where we will make sure that everything goes according to plan.

2

TEMPORAL PLANNING

The start date I consider for the project is the 19th of September 2022, and the end date as the 24th of January when it is my thesis' defense. That is a total of 126 days over 18 weeks and the total amount of hours I am going to work based on my tasks is 265 hours. I plan to usually work only on weekdays which are exactly 90 over this time period, so the estimated amount of time I will work each day is 3 hours. Of course those are estimates and changes based on more recent circumstances will be made.

2.1 DESCRIPTION OF TASKS

2.1.1 *Task Definition*

In this section I will further analyze and define the tasks that I will complete, in order to finish my thesis. This will include categorizing the tasks into groups, estimating how much time each one will take and explain in short the reason and contents of each task. I will categorize my work into 3 main groups: The planning and management of the project, the theoretical studying for the project and the practical implementation of the project.

Project Management and Planning

This section mostly contains the work I will do in the start of the semester to manage my work, plus some enhancements I will do with my director while time goes by. It is an important part of the project because it is going to help me organize my work and clear things out in my head in terms of the direction I will follow and how I will work on my thesis.

a. **Contextualization and project scope (T2).** Here, I will define the scope of the project in the context of its study. I will indicate the general objective of the TFG, the context, the reason for selecting the subject area, how the project will be developed and using which means. I have assigned 10 hours to this task

b. **Time Planning (T3).** Here I will plan the entire execution of the TFG. Also, I will describe the project phases, the resources and requirements associated with each one. I have assigned 7.5 hours to this task

c. **Budget and sustainability (T4).** Here, I will do an analysis of the sustainability of the project and make a budget for the project. I have assigned 7.5 hours to this task.

d. **Integration in final document (T5).** Here, I will compose a written document summarizing the whole project. I have assigned 10 hours to this task

e. **Director Meetings (T6).** Part of the management of the course will be weekly meetings with my director to make sure everything goes according to plan and make management changes when needed. I have assigned 20 hours to this task, since I calculate about 20 meetings in total

Theoretical Study

The goal of the project is to implement a low-cost AI accelerator and integrate it into an existing CPU. But before doing that there is some theoretical reading that is necessary to be completed. More specifically:

a. **Understand SPARROW (T8).** Of course, one of the most important things I have to do before implementing the code of SPARROW for my project is read the original master thesis of Marc for SPARROW and understand perfectly not only its design, but also its implementation. I have assigned 15 hours to this task

b. **Master my understanding of writing RTL design (T9).** I already have knowledge and have written SystemVerilog, but in order to confidently continue working on my project, I must have a strong understanding of philosophies and design patterns for RTL on a bigger scale. I have assigned 20 hours to this task

c. **Understand the base processor(s) (T10).** To integrate my module into a processor I have to have a good understanding of its pipeline, how are modules integrated/inserted to the cpu and lastly how interactions with the register file work (due to the nature of my project). Generally, I will have to know a lot about the processor but it not necessary to know everything about it as long as I understand the aforementioned key parts. I have assigned 15 hours to this task

Practical implementation

In this section I will describe the practical tasks of my project that will build step by step the implementation of my project.

a. **Refresh System Verilog knowledge and learn VHDL (T12).** This may seem like a theoretical task but anyone who has studied programming knows it is actually practical. I need to know very well both languages since I will read a lot of VHDL and both read and write a lot of System Verilog. I have assigned 20 hours to this task

b. **Write the code for SPARROW in (System) Verilog (T13).** With this task I will implement the module of SPARROW and it should take a significant amount of time. I have assigned 40 hours to this task

c. **Integrate it to an existing RISC-V processor (T14).** After completing the module of SPARROW, I will integrate it to the base CPU. I have assigned 20 hours to this task

d. **Write RISC-V assembly tests for validation (T15).** Those tests are needed for low-level testing and can be made at any time. I have assigned 10 hours to this task

e. **Validate the project using the assembly tests and existing sequential C code tests (T16).** With this task I will verify that my job is actually correct. It is worth noting that validation will be also done intermittently on smaller parts of the project that will work as checkpoints. Those will be smaller tasks and will be decided later based on the understanding I will have of the project after the theoretical studying and some practical work. I have assigned 20 hours to this task

f. **Draw conclusions (T17).** I have assigned 10 hours to this task

Independent Tasks

In this section I will present 2 tasks that do not exactly fit into any of the other 3 main categories of tasks.

- **Keep documentation of my work throughout the semester (T19).** This is a very important task that needs to be carried out all semester long and in parallel of my work and will help me easily compile my final thesis document. I have assigned 20 hours to this task, i.e. every meeting with my director will correspond to some documentation collected for the portion of work between the 2 last meetings.

- **Prepare the oral presentation (T20).** After all my work is finished I still have to present my thesis to the appointed committee, and the quality of the presentation plays a big role in the overall viewing of the project, thus it must be good and well-prepared. I have assigned 10 hours to this task

2.1.2 *Resources*

In this section I will analyze the resources needed in order to successfully complete my thesis in all aspects, i.e. reading, implementation, managing, writing. These resources can be divided into 4 main categories: Human, Hardware, Software and Other Material Resources.

Human Resources

The first and most important human resource is, obviously, myself(**HR1**), since without my individual good effort and productive work the project will not finish. Apart from that, the contribution of my director, Dr. Leonidas Kosmidis(**HR2**), will be of great importance in giving me guidance when needed in all aspects of the project, from what theory to read to understand some concepts better, to a different approach on the practical end of the project and even to how I should write a part of my thesis so that it is better understandable for the reader. In all aspects the experience of my director in such work is far greater than mine and his assistance will be priceless. Last but not least, my GEP tutor, Paola Lorenza Pinto(**HR3**), will help me enhance the writing part of my thesis with her feedback.

Hardware Resources

This is a computer science project specialized in hardware, so its hardware resources are very important albeit few in this case. The tw main resources I will use for the completion of my project are my personal laptop and BSC's server machines.

- My personal laptop, an M1 16GB RAM, 256GB SSD, MacBook Air 2020(R1)
- BSC's servers(R2)

Software Resources

As any other computer science project, there will be a lot software resources needed to assist its completion. Specifically:

- Gitlab will be used for managing the code base of the project (R3)
- Vim is my editor of choice and where I will write most, if not all, of my code (R4)
- Verilator will be used for simulating my project locally, on my personal machine (R5)
- Xilinx Vivado will be used for synthesizing the code(R6)
- Overleaf is used for writing my thesis through LateX (R7)
- Zoom and Google Meets will be used for online meetings with my director (R8)

Other Material Resources

Apart from the obvious hardware and software resources, other material resources will also make themselves useful towards the completion of this project. I am talking about research papers, manuals (e.g., the SweRV Core EH1 manual or Verilator manual) and Marc Solé Bonet's thesis for reading purposes in order to improve both my theoretical

understanding but also my practical skills. In integrating the module in the SweRV Core EH1 the Imagination University Programme will be very helpful[14]. Nevertheless, my practical skills will mostly be improved through tutorials whether they are video or web-coding tutorials. Lastly, the GEP course material will help me improve my writing and manage my project in a better way.

2.1.3 *Task Summary*

In this section I will summarize the tasks, their dependencies, the material resources they use and their estimate time through a table. The hours given to each task and task category are rough estimates and are likely to change throughout the course of the semester. For the clarification, the second to last column refers to the human resources and the last column refers to material resources.

Table 2.1: Table of Task Summary.

Tid	Task Name	Time(h)	Dependencies	H. Resources	M. Resources
T1	**Project Management and Planning**	**55**	-	-	-
T2	Contextualization and project scope	10	-	HR1,HR3	R1, R7
T3	Time planning	7.5	T2	HR1,HR3	R1, R7
T4	Budget and Sustainability	7.5	T3	HR1,HR3	R1, R7
T5	Integration in final document	10	T4	HR1,HR3	R1, R7
T6	Director Meetings	20	-	HR1,HR2	R1, R8
T7	**Theoretical Studying**	**50**	-	-	-
T8	Understand SPARROW	15	-	HR1	R1
T9	Understand RTL design	20	-	HR1	R1
T10	Understand base CPU	15	-	HR1	R1
T11	**Practical implementation**	**130**	-	-	-
T12	Refresh Verilog and learn VHDL	20	-	HR1	R1
T13	Write the code for SPARROW	30	T8, T9, T11	HR1	R1-R6
T14	Integrate SPARROW to a CPU	40	T10, T13	HR1	R1-R6
T15	Write assembly tests	10	-	HR1	R1, R4
T16	Validate the project	20	T14, T15	HR1	R1-R6
T17	Draw conclusions	10	T16	HR1	R1
T18	**Independent tasks**	**30**	-	-	-
T19	Keep documentation	20	-	HR1	R1
T20	Prepare oral presentation	10	T16, T17, T18	HR1	R1

2.2 ESTIMATES AND THE GANTT

In this section I will present a Gantt diagram of my project's timeline based upon the descriptions and time estimates of the previous section.

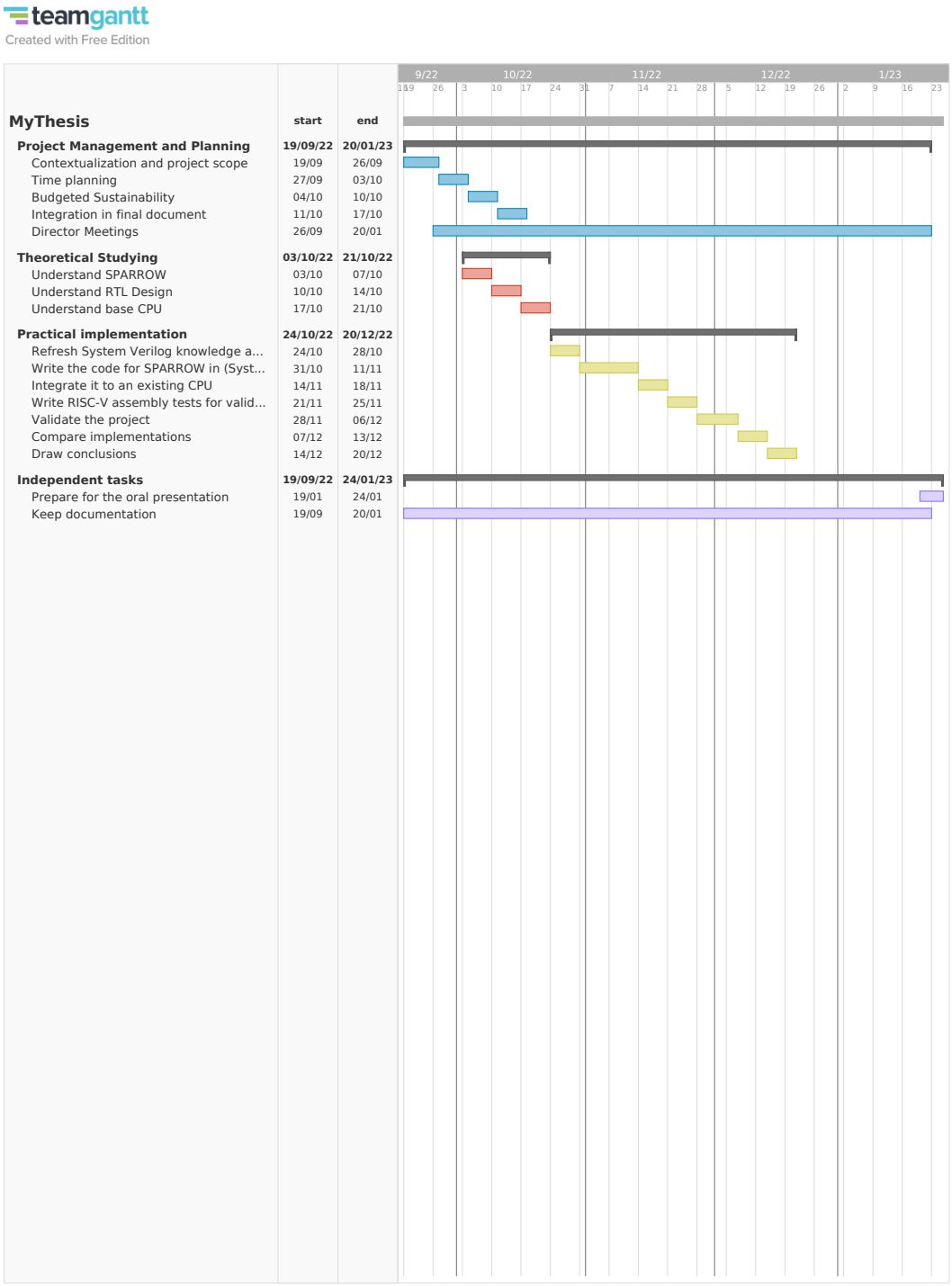

Figure 2.1: Gantt Diagram of my project's timeline

2.3 RISK MANAGEMENT: ALTERNATIVE PLANS AND OBSTACLES

As we previously saw in the *Potential obstacles and risks* section, there are may occur some obstacles along the way that will expose the completion of this project. For every one of these risks/obstacles we will analyze its importance, the possibility of it happening and an alternative plan in case of its occurrence.

Time

This is an obstacle that may cause a problem for many reasons. Whether that could be bad time-planning of the tasks, or delay of the completion of a task for ***whatever*** reason or even my *inexperience on hardware projects of such scale*. A lot of things can cause a setback, whether that is minor or major and the time-planning of the project cannot be perfect from the beginning, no matter how hard we try.

- **Probability of happening**: Very Low
- **Impact**: Very High
- **Alternative Plan**: Although, setbacks will occur for sure and the planning cannot be perfect, this imperfect planning has already predicted some time for setbacks. Also, if a task does not get completed in time that only means extra work for me and since the hours of this project although tight, are reasonable, there will most probably always be time to fix some setbacks. Lastly, a paradoxically good thing that usually occurs with imperfect planning is that some tasks are overestimated, something that will "give time back" for tasks that take for time.

Hardware code simulating but not synthesizing

This a very common obstacle that can occur from bad testing (primarily in simulation), incorrect use of timing and delays in the code and physics and/or tool-related reasons.

- **Probability of happening**: Medium
- **Impact**: Medium
- **Alternative Plan**: From the start of the project, the goal will be to be very careful with the development of the code and use thorough simulation testing before moving to synthesizing. In this way it is possible that I will never see the project simulating and not synthesizing. In the unfortunate event that this happens, I will have to make an extra task, of a reasonable amount of time(8 hours), for debugging the project. The worst case scenario here is not fixing the error. In that case, I will have to make an extra task, which will be even more time-consuming(20 hours), for deleting and redoing from scratch the part of the project that causes the problem.

Inexperience on hardware projects of such scale

- **Probability of happening**: Medium
- **Impact**: Low to Medium
- **Alternative Plan**: It is certain that at some extent, my inexperience will affect my productivity. But on the one hand, that is something common for a bachelor thesis and on the other hand, guidance from my director will certainly help me short things out. Usually, what I think will happen is having to create some extra (smaller) tasks (2-6 hours) to fill some gaps in my work and nothing more serious. This has also being prevented by giving myself generous, but also on schedule, time for completing my tasks.

3

BUDGET

In this chapter I analyze the budget of my project in terms of the total costs and I mention how I managed it throughout the course of the completion of my project.

3.1 IDENTIFICATION OF COSTS

To have a proper idea of the budget I will have to consider all its elements and estimate their value. I have categorized the different costs in 4 groups, staff costs, generally calculated costs, contingencies and incidentals, which I will further analyze in this section.

3.1.1 *Staff Costs*

In order to estimate correctly and accurately the staff costs, I first have to define the staff roles that would be needed for a real life project of such scale to be carried out, and I will also define who will play these roles in the case of my project. First of all, a **Project Manager** would be responsible for most the first part of the project as I have defined it, which is **Project Management and Planning(T1)**. In my case, this part will be carried out by me, with the help of my director and my GEP tutor. Also, a **Technical Writer** would assist in the composing of the final thesis document(T5). This project is of a reasonable scale so most probably in a real life scenario (as in my case), only one person would be responsible for all the rest of the tasks but I will further divide the work into roles for the sake of this analysis. A **Computer Engineer** would do all of the **Theoretical Studying(T7)** and most of the **Practical Implementation(T11)**. Also, a **Tester** would write the tests(T15) and validate the project(T16) and lastly, an **Analyst** would draw conclusions(T17). As I mentioned earlier, all these roles can be carried out by the same person, and that is what will happen here with myself being responsible for all these tasks. The salaries I use later are averages of the respective real life jobs, and my source for these averages is glassdoor[15]. The acronyms for the responsible roles as well as their salaries and analytically the division of tasks along with their cost are displayed in the two following tables. It should be noted that in the tables below I do not account for social security, which will be accounted for later in the calculation of the final budget.

Table 3.1: Summary of the roles and their salary.

Role	Yearly Salary (€)	Hourly Salary (€)
Project Manager (PM)	37.250	17.91
Computer Engineer (CE)	39,184	18.84
Tester (T)	39,184	18.84
Analyst (A)	39,184	18.84
Technical Writer (TW)	36,342	17.47

Table 3.2: Estimated costs per task based on the roles responsible.

Tid	Task Name	Time(h)	Role Responsible	Cost(€)
T1	**Project Management and Planning**	**55**	-	**1357,45**
T2	Contextualization and project scope	10	PM	179,1
T3	Time planning	7.5	PM	134,325
T4	Budget and Sustainability	7.5	PM	134,325
T5	Integration in final document	10	TW	174,7
T6	Director Meetings	20	PM, CE	367,5+367,5=735
T7	**Theoretical Studying**	**50**	-	**942**
T8	Understand SPARROW	15	CE	282,6
T9	Understand RTL design	20	CE	376,8
T10	Understand base CPU	15	CE	282,6
T11	**Practical implementation**	**130**	-	**2449,2**
T12	Refresh System Verilog knowledge and learn VHDL	20	CE	376,8
T13	Write the code for SPARROW in (System) Verilog	30	CE	565,2
T14	Integrate it to an existing RISC-V processor	40	CE	753,6
T15	Write RISC-V assembly tests for validation	10	T	188,4
T16	Validate the project	20	T	376,8
T17	Draw conclusions	10	A	188,4
T18	**Independent tasks**	**30**	-	**739,9**
T19	Keep documentation	20	CE	376,8
T20	Prepare for the oral presentation	10	CE	363,1
-	**Total CPA Cost**	265	-	**5.538,55**

3.1.2 *Generally Calculated Costs*

In this section I will consider costs that are not directly connected to the tasks and are more general. For these costs I will consider amortization, work space and internet costs.

Amortization Costs

- Hardware

  Throughout the course of the 5 months of my project I will use my personal laptop. I will have to take into account the amortization of this resource into the cost of my project. The price of my laptop is 1500€. I will use the laptop for all 5 months and every day. I consider for this resource a *useful lifetime* of 5 years(60 months) and the equation for amortization I will use is:

$$Amortization = ResourcePrice \times TimeUsed \times \frac{1}{lifetime} \tag{3.1}$$

  So we will get the following result:

Table 3.3: Table of amortization costs.

Resource	Price(€)	Lifetime	Time Used	Amortization(€)
M1 16GB RAM, 256GB SSD, MacBook Air 2020	1500	60 months	5 months	125
Total	-	-	-	**125**

- Software

  All software I will be using is free and opensource, so the amortization cost for my software will be 0€.

Work Space Costs

Mainly, I will work from home but I will also do parts of my work at the Tillers Building at BSC, where they are so kind to have me as a visitor. Thus, I will only consider the hours I will be working from home as work space costs. Out of the 90 work days I have planned to work, I consider that 60 of them will be done from home, and I have planned to work 3 hours every day. That is in total 180 hours or 7,5 whole 24-hour days. Considering a month of rent without accounting the internet is 740€ for 30 days, the cost of the work space will in the end be:

$$WorkSpaceCost = 740 \times \frac{7,5}{30} = 185 \tag{3.2}$$

Internet Costs

Internet costs 30€ each month and as calculated in the previous section, it will be used for 7,5 whole 24-hour days in order to complete my work. Thus, the total internet cost will be:

$$InternetCost = 30 \times \frac{7,5}{30} = 7,5 \tag{3.3}$$

Total Generic Costs

Table 3.4: Summary of generic costs.

Case	Cost (€)
Amortization	125
Work Space	185
Internet	7,5
Total	**317,5**

3.1.3 *Contingencies and Incidentals*

Contingencies

Throughout the course of this project there may occur some unpredicted events that will cause the budget to increase or they may even be some bad estimations in the initial budget we have now. Those possible extra costs, have to be taken into account beforehand and thus we have to add a contingency margin which will be, as usually in the current market, 15% of the total cost (CPA + generic costs). So we get:

$$ContingencyMargin = 0.15 \times (5.538,55 + 317,5) = \mathbf{878,4} \tag{3.4}$$

Incidentals

As we have previously already discussed in the Potential obstacles and risks section, there are some unfortunate events that can occur throughout the course of this project, that will threat to derail its course. Those events will of course also cost, at the very least in terms of work hours, which will have to be paid. Thus we have to take into account the potential costs these risks can cause and see if our contingency margin can cover them.

- **Time.** In general I will give a maximum of 20 more hours of work in order to achieve deadlines that will be tight, and I give a 20% chance of all them happening.
- **Hardware code simulating but not synthesizing.** As I explained in the previous chapter, at the worst case scenario I believe no more than 28 more hours will be given to this obstacle, with again a 20% chance of that happening.
- **Inexperience on hardware projects of such scale.** For this task I will give 20 hours of more work with a 50% chance of that happening.

The total incidental costs will be:

Table 3.5: Summary of Incidental costs.

Risk	Extra Hours	Risk(%)	Cost(€)/h	Extra Cost(€)
Time	20	20	18,84	75,36
Hardware Code Problems	28	20	18,84	105,5
Inexperience	20	40	18,84	150,72
Total	-	-	-	**331,58**

3.2 FINAL BUDGET

Based on the previous sections and also accounting for **social security**(35% of CPA) the total final budget will be:

Table 3.6: Table of the Final Budget.

Activity	Cost(€)
CPA	5.538,55
Social Security(35% of CPA)	1.938,5
Generic Costs	359,16
Contingency Margin	878,4
Incidental Costs	331,58
Total	**9.046,19**

3.3 MANAGEMENT CONTROL

As I have mentioned before throughout this chapter the estimations made for this budget are likely to be off, either positively (less cost actually needed) or negatively (more cost actually needed). For this reason, at the end of each task, I will assess it in terms of cost in order to manage accordingly. The process I follow is every time I finish a task, I will measure the **actual cost** needed to complete it. If that cost is higher than the **estimated cost**, I will take some capital off the contingency margin to feel that void. In the fortunate case that that cost is less, I will just add the difference in the contingency margin.

4

SUSTAINABILITY

In this chapter perform a self-assessment of my previous knowledge and capabilities in terms of assessment and I answer some reflective questions referring the sustainability of my project in terms of three different dimensions. Those dimensions are the Environmental, the Economic and the Social Dimension.

4.1 SELF-ASSESSMENT

In completing the questionnaire, I realized a lot about my knowledge on the topic of sustainability. First of all, beforehand, I knew that there are many problems in our modern world involving various dimensions that sustainability covers. On the environmental part, I was aware of the most important issue that is climate change and an issue more connected to computer science is the huge amount of energy that is consumed. On the social aspect, it is no secret that we live in weird times and that many social issues have arose in the last few years, most of them being directly linked to the IT industry like mental health crisis, social media addiction and an increase in consumerism. Lastly, I understood that it would be important for all parts involved in a project to reduce its costs as much as possible. But, as I found out through the questionnaire, all this is just the tip of the iceberg.

Furthermore, I have always thought I have been sensitive to the effects that my personal work and also the work of my field in general will have to society in whatever way that is. Over the years I have contemplated about ethical issues, issues of social justice and many more that it is not necessary to mention one by one. But, this questionnaire made me think about how many issues there actually are to consider and how many different viewpoints for each one.

Finally, I got to admit I was starstruck by the sheer amount of technical knowledge that exists on the assessment of those fields that I really knew very little about. Knowledge involving metrics, tools, strategies and many more. Thus, I have to say that filling that questionnaire really opened my eyes about how much there is out there and how much room for improvement there is for me in this aspect in terms of becoming practically better at it as a professional in the near future, besides abstractly thinking about only part of the topics included.

4.2 ENVIRONMENTAL DIMENSION

Have you estimated the environmental impact of undertaking the project? Have you considered how to minimize impact, for example by reusing resources?

Yes, I have thought about the environmental impact of my project. It is a project of small to reasonable scale that requires very few resources in order to complete, thus I consider its direct environmental impact to be small, no great amount of energy (other than my one) will be consumed nor the production of this project will directly affect the environment in any other way. Indirectly, since the project is a very low-cost and energy-efficient AI accelerator and it will be initially implemented on an also low-cost, small area base CPU, we can say that it is environmentally friendly. It is difficult to calculate how big or small its indirect impact will be at the end of the day.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution environmentally improve existing solutions?

As I have mentioned in the first chapter, there are few solutions in the RISC-V community that are low-cost and energy-efficient and the focus is more towards performance. In this way, my project will environmentally improve the existing solutions at the state-of-the-art since it will offer an environmentally friendly CPU processor with AI acceleration to the RISC-V community. Also, it worth noting that RISC-V processors are by default more energy efficient than their x86 counterparts due to the simplicity of the ISA.

4.3 ECONOMIC DIMENSION

Have you estimated the cost of undertaking the project (human and material resources)?

Yes, I have and very thoroughly and you can see it at the chapter about BUDGET.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution economically improve existing solutions?

SPARROW is targeting integration with existing space processors. Currently, to produce a new space processor is a very difficult, tiring and expensive procedure that companies would like to avoid. SPARROW being integrated into an existing processor, will make it far easier in terms of both time and money to improve the performance of an existing processor, and thus eliminating the need of going through the whole procedure of creating a new one.

4.4 SOCIAL DIMENSION

What do you think undertaking the project has contributed to you personally?

Personally, I seek to improve a lot as a person by undertaking this project. First of all, it is one of the biggest projects I have undertaken so far in my life and I will get valuable experience in a wide range of work activities from project managing and technical writing to coding, analyzing and testing a project of such scale. I am in my Erasmus semester so in order to enjoy it I will have to find a way to balance work and free time, a skill that will be valuable for the rest of my adult life. Lastly, in order to complete such a project I will need to improve my self-discipline and productivity. In general, I will learn a lot on my way towards completing this project.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution socially improve existing solutions?

I don't think my project has much of a social impact. It is a portable and cheap AI accelerator that aims at being a good and easy-going solution for people in the industry and research domains that would like to use a CPU with cheap and efficient AI acceleration. Of course it will be open-source.

Is there a real need for the project?

From a social standpoint I don't think there is one. From a research and industry standpoint this project will definitely be useful to people in the future, so yes.

5

ANALYSIS OF THE IMPLEMENTATION

In this chapter I explain in detail the design and pipeline of SweRV Core EH1 and the design of SPARROW. I also explain design decisions I have made throughout the progress of the project and explain the process of integrating SPARROW in SweRV Core EH1.

5.1 THE SWERV CORE EH1 AND ITS PIPELINE

The SweRV Core EH1 is a commercial open-source processor designed by Western Digital. As it can be seen in Figure 5.1, it has a 9 stage pipeline and it is superscalar, having 2 ways. Specifically, the stages of its pipeline are: Fetch1, Fetch2, Align, Decode, EX1/DC1/M1, EX2/DC2/M2, EX3/DC3/M3, Commit and Writeback stages. SweRV has parallel paths (load/store vs. integer vs. multiply pipes), and splits some stages into multiple stages (Fetch is 2 stages and Execute is 3 stages). In terms of memory, besides Main Memory, the open-source version of EH1 has an Instruction cache, an ICCM (Instruction Closely Coupled Memory) and a DCCM (Data Closely Coupled Memory). The ICCM and the DCCM are scratchpads for the instructions and the data respectively. The use of the Instruction Cache, the ICCM and the DCCM is optional. It is important to note that during the Execution Stages only one of the pipes actually works per way, the Decode stage has decided which one that is based on the instruction that it currently going to run. All the other pipes are disabled. At the end of the 3rd Execution stage the correct result between the load/store, the multiply and the two integer pipes is selected through a multiplexer. Again the value that the multiplexer will select has been determined at the Decode Stage. There is a 34-cycle Out-of-Pipeline Divider. The aforementioned multiplexer does not take into account the result of this divider. This result is selected, if necessary, during the Writeback stage. It is also worth mentioning that EH1 understands where data is placed (Main Memory or DCCM) based on a fixed address map. For this reason, in order to use the DCCM, a different linker script has to be used in order to place the program data in the address space mapped to the DCCM.

5.1.1 *A few words about DCCM and ICCM*

The Tightly Coupled Memories are known also as scratchpads[16]. Scratchpads are software managed "cache" memories, in which the data which are written to them are guaranteed to be present when they are read. Therefore it is the programmer's responsibility to

manage their contents, unlike cache memories, which in case of a cache miss, the required piece of memory is automatically requested from DRAM.

Scratchpad memories are used very frequently in real-time systems and in other systems which are latency sensitive, because they can provide low-latency and deterministic execution time.

DCCM and ICCM in SweRV are two dedicated memories, one for instruction and the other for data, that are tightly coupled to the core. These memories provide low-latency access and SECDED ECC protection. Their respective sizes (4, 8, 16, 32, 48, 64, 128, 256, or 512KB) are set as arguments at build time of the core.

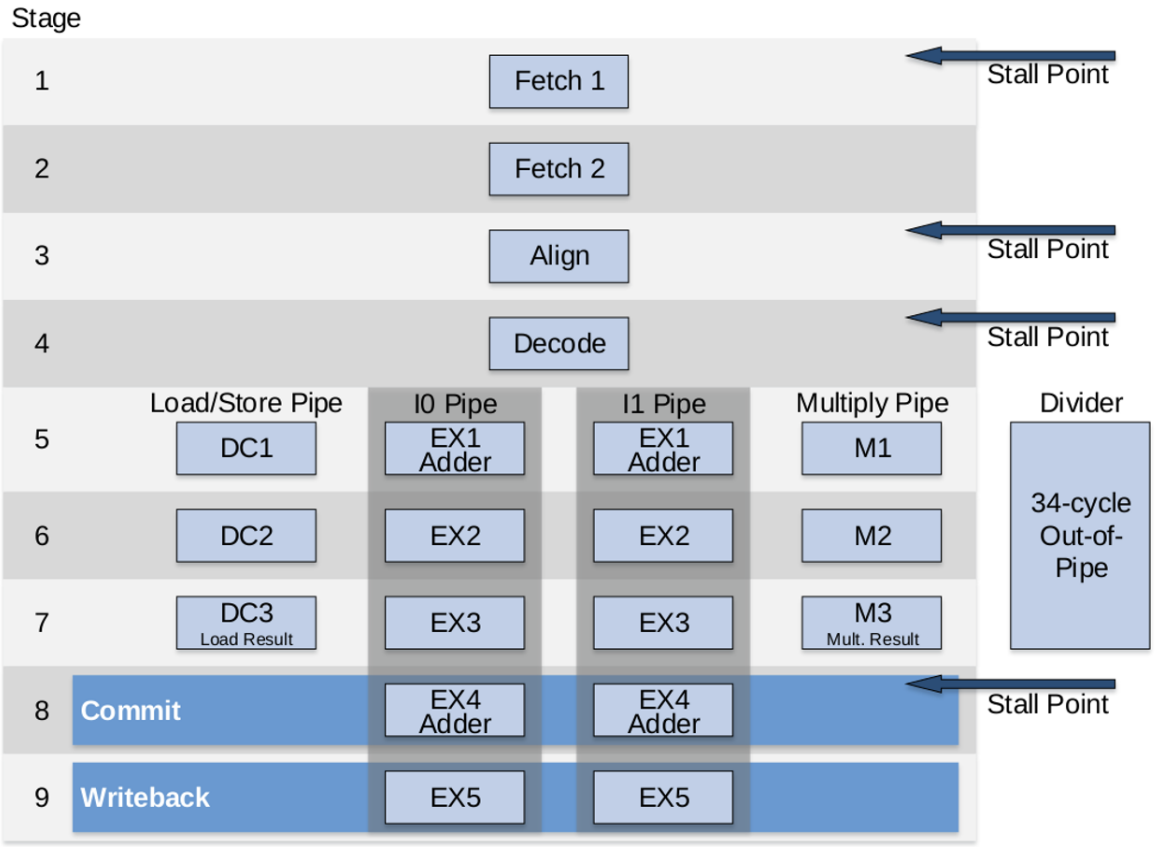

Figure 5.1: Overview of the SweRV Core EH1 Pipeline (image taken from [17])

5.2 THE SPARROW DESIGN

SPARROW [4][1] is a 2-stage SIMD accelerator. It is targeted for AI workloads and to be integrated mainly with space processors. The main idea behind SPARROW is that it is proven that for modern AI workloads, 8-bit values are enough [18]. This way SPARROW just by having two 32-bit words as input can process simultaneously 4 different 8-bit input values.

5.2.1 *First stage*

In the first stage of SPARROW the same operation is executed for all pairs of values. The operations supported for the first stage are arithmetic(add, sub, mul, etc, but not div), shifts and logical. For the arithmetic operations there signed and unsigned versions of them and also versions with built-in saturation. For saturation, the maximum and minimum value depends on the sign, more specifically 0 to 255 for unsigned and -127 to 128 for signed, since the output value must be 8 bits. In Table 5.1 we can analytically see the operations supported for the first stage of the of SPARROW.

Table 5.1: SPARROW first stage operation codes (table taken from [1])

sd1	Name	Operation		
00000	`nop`	$rd' = rs1$	$rd' \in \mathbb{N}$	
00001	`add`	$rd'_i = rs1_i + rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
00010	`sub`	$rd'_i = rs1_i - rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
00011	`mul`	$rd'_i = rs1_i \times rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
00101	`max`	$rd'_i = max(rs1_i, rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
00110	`min`	$rd'_i = min(rs1_i, rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
00111	`and`	$rd'_i = rs1_i \wedge rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
01000	`or`	$rd'_i = rs1_i \vee rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
01001	`xor`	$rd'_i = rs1_i \oplus rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
01010	`nand`	$rd'_i = \neg(rs1_i \wedge rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
01011	`nor`	$rd'_i = \neg(rs1_i \vee rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
01100	`xnor`	$rd'_i = \neg(rs1_i \oplus rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
01101	`sadd`	$rd'_i = max(-128, min(127, rs1_i + rs2_i))$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
01110	`ssub`	$rd'_i = max(-128, min(127, rs1_i - rs2_i))$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
01111	`smul`	$rd'_i = max(-128, min(127, rs1_i \times rs2_i))$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
10000	`merg`	$rd' = rs2$	$rd' \in \mathbb{N}$	
10001	`shft`	$rd'_i = rs1_i \ (\ll \mid \gg)^1 \ rs2_i/2$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{Z}$
10011	`umul`	$rd'_i = rs1_i \times rs2_i$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
10101	`umax`	$rd'_i = max(rs1_i, rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
10110	`umin`	$rd'_i = min(rs1_i, rs2_i)$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
11001	`sshft`	$rd'_i = max(-128, min(127, rs1_i \ (\ll \mid \gg)^1 \ rs2_i/2))$	$\forall i \in \{0,...,3\}$	$rs1 \in \mathbb{Z}^2$
		$rd'_i = max(0, min(255, rs1_i \ (\ll \mid \gg)^1 \ rs2_i/2))$	$\forall i \in \{0,...,3\}$	$rs1 \in \mathbb{N}^2$
11101	`usadd`	$rd'_i = max(0, min(255, rs1_i + rs2_i))$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
11110	`ussub`	$rd'_i = max(0, min(255, rs1_i - rs2_i))$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$
11111	`usmul`	$rd'_i = max(0, min(255, rs1_i \times rs2_i))$	$\forall i \in \{0,...,3\}$	$rs1, rs2 \in \mathbb{N}$

1. The second operand sign determines the direction of the shift. Also its last bit identifies whether the shift is logic or arithmetic.

2. If the shift is logical it treats the data as unsigned whereas if it's arithmetic as signed.

5.2.2 *Second stage*

In the second stage there are performed reduction operations using all 4 results from the first stage. There is also the option of not performing any operation at all. The supported reduction operations for the second stage are: sum, max, min, xor, usum, umax, umin. The reduction operations also have a saturated version, but due to limited bits in the instruction for the opcodes, the same saturation option as for the first stage is used. In Table 5.2 we can analytically see the operations supported for the second stage of the of SPARROW.

Table 5.2: SPARROW second stage operation codes (table taken from [1])

sd2	Name	Operation	
000	`nop`	$rd = rd'$	$rd' \in \mathbb{N}$
001	`sum`	$rd = \sum rd'_i \quad \forall i \in \{0,...,3\}$	$rd' \in \mathbb{Z}$
010	`max`	$rd = max(rd'_0, rd'_1, rd'_2, rd'_3)$	$rd' \in \mathbb{Z}$
011	`min`	$rd = min(rd'_0, rd'_1, rd'_2, rd'_3)$	$rd' \in \mathbb{Z}$
100	`xor`	$rd = rd'_0 \oplus rd'_1 \oplus rd'_2 \oplus rd'_3$	$rd' \in \mathbb{N}$
101	`usum`	$rd = \sum rd'_i \quad \forall i \in \{0,...,3\}$	$rd' \in \mathbb{N}$
110	`umax`	$rd = max(rd'_0, rd'_1, rd'_2, rd'_3)$	$rd' \in \mathbb{N}$
111	`umin`	$rd = min(rd'_0, rd'_1, rd'_2, rd'_3)$	$rd' \in \mathbb{N}$

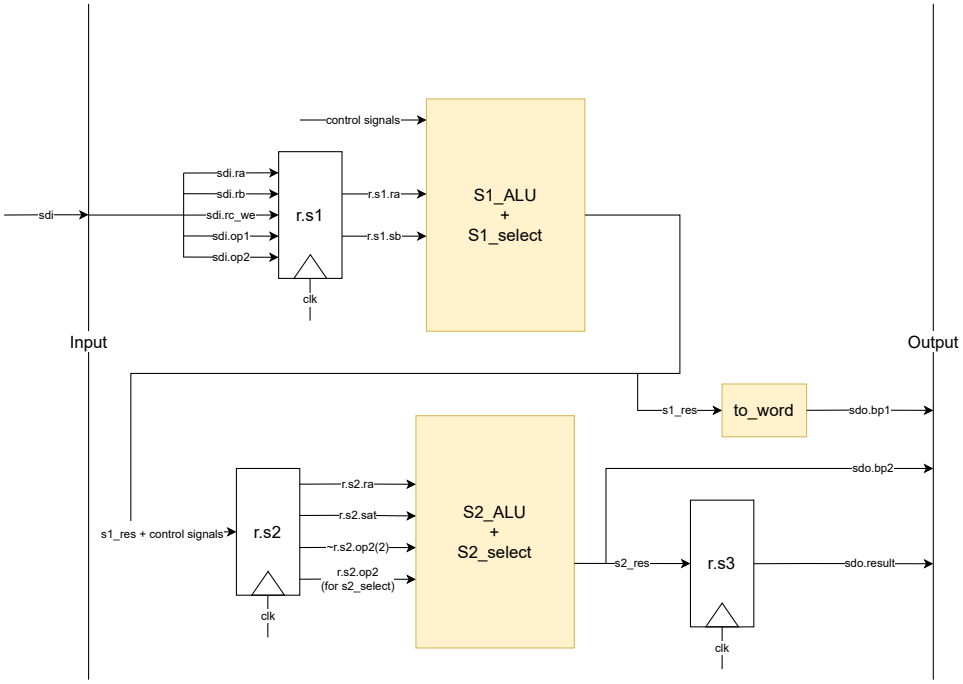

Figure 5.2: Overview of the SPARROW design

5.2.3 *General information*

The most novel feature of SPARROW is that it reuses the register file of the host core. In this way, SPARROW is very low cost in terms of area when integrated into a core and the integration has to be very close. Lastly, there is compiler support[19] for the SPARROW instructions which is very useful for testing of my hardware design, without the need to modify the compiler.

5.3 IMPLEMENTING SPARROW AND INTEGRATING SPARROW IN EH1

5.3.1 *Experience of Porting SPARROW from VHDL to SystemVerilog*

The most challenging part of the implementation was learning VHDL and the correspondences between VHDL and SystemVerilog. Features that SystemVerilog has - unlike Verilog - regarding types were really helpful in achieving a similar design pattern. Apart from that, the design of SPARROW is fairly simple using primitive operations and thus there was no severe problems faced during the implementation. At the end, a simple, but thorough testbench was used to validate the correctness of the design. All features supported by the VHDL version are supported by the SystemVerilog version.

5.3.2 *Experience integrating SPARROW in SweRV Core EH1*

The most difficult part of this project was integrating the SystemVerilog implementation of SPARROW into the SweRV Core EH1. The main reason for this is that the selected core is an industry-ready, power, area and timing optimised processor instead of an academic core written with clarity in mind and in order to be easily modified, like for example the Ariane core from ETHZ [20]. To integrate SPARROW there were some key steps that needed to be followed.

Support in the decode unit

First of all, the core needs to be able to decode the SPARROW assembly instructions. To do that there had to be made various changes in data structures and logic in the decode unit. Changes in the logic were optimized with the help of the espresso logic generator [21] from UC Berkeley.

Support in the Execution Stage

Once, the instructions were recognizable by the decoding unit of the processor and the correct control signals in terms of SPARROW were generated, the next step was to actually integrate the module in the execution stage. For this, I had to understand in the code of the core, which were the correct inputs for SPARROW and which were the correct outputs. The biggest design decision made throughout the integration was the decision to integrate

SPARROW inside the multiplier pipe. There were various reasons for this. First of all, that, and also making the processor recognize SPARROW instructions as a subset of mul instructions, made the integration easier in terms of small things that the processor already did for the existing pipes. The biggest takeaway from this decision was that now sparrow had the same output signal as the multiplier pipe and the processor would do the handling from there. Even in terms of bypassing the output of SPARROW, since for the core it was like bypassing a mul instruction. Also, a very important consequence of this decision was that SPARROW instructions would now take a **fixed 3 cycles to complete**, like the mul instructions.

Bypassing support

As stated before the bypassing of the output is automatically handled by the core, because SPARROW is integrated with the multiplier pipe. That was not the case for the input though. In order for the SPARROW instructions to be able to run with no additional delays than the ones that were absolutely necessary there had to be implemented some handling of late inputs that were not already handled by the core. The way SweRV Core EH1 works here is that when there is going to be a late input, a signal is raised inside the pipe (the multiplier in our case). That late input comes at the first Execution Stage and usually there is going to be an additional stall there. This means that all the signals inside the module have to remain unchanged for the duration of the stall, in order not to lose information that would change the result. When the stall (if there is actually one) is over, the correct inputs for the module are available, and since all the control signal are correct, the correct result can be computed without any problem.

6

EVALUATION

In this chapter I present and analyze the results from evaluating my SPARROW implementation within the EH1 core. To evaluate SPARROW in EH1, 4 benchmarks were used, representative of AI application workloads. The same benchmarks were used to evaluate SPARROW integrated with LEON3 and NOEL-V in [4][1]. One containing a program performing a matrix multiplication algorithm, one containing a program converting an RGB image to greyscale, one applying a 3x3 filter to a square matrix and lastly one performing the calculation of N polynomial equations. For each benchmark many different sizes were used, and the speedup between a version of the program with sequential C code and a version of the program utilizing SPARROW instructions was measured. Also, 2 different measurements were performed for all benchmarks, for 2 different configurations. One using the DCCM, and without it, in which the data are fetched from DRAM. Here is a good place to remind that the open-source version of EH1 does not have a data cache. In order to use the DCCM in the programs, only changes in the linker script needed to be made. The size of the DCCM was also changed to be 512KB, from 64KB that is the default. For everything else, the default configuration was used. In Table 6.1 we can see in detail the configuration parameters used for obtaining the measurements.

Table 6.1: The EH1 hardware configurations used in the evaluation.

Dual-issue	Enabled
Instruction Cache	Enabled, 16KB
ICCM	Enabled, 512KB
DCCM	Enabled, 512KB

Even when enabled in the hardware configuration of the core, in order to actually utilize the ICCM and DCCM from the software, the appropriate linker script must be written, so that the data or instructions are mapped to the corresponding closely coupled memories. In the benchmarks below, the ICCM was not used by the software, although it was enabled in the processor configuration.

Moreover, it is important to note that the benchmark input arrays were not hardcoded with static values, but they were (pseudo)randomly initialized dynamically inside the program. Thus, the cycles corresponding to the initialization were calculated and subtracted from the total program cycles, in order not to obscure the speedup measurements.

The 4 benchmarks used are the same (with some minor changes), that were used to evaluate the initial SPARROW implementation in Marc Sole Bonet's Master's thesis. Of course, I have validated the correctness of the results when using SPARROW, which produces identical results with the scalar version. It is also worth noting that these 4 benchmarks were based on 4 benchmarks kindly provided by Mr. Mathew Johns, and were used in his work [22].

6.1 MATRIX MULTIPLICATION BENCHMARK

In Tables 6.2 and 6.3 we can see the results of the matrix multiplication benchmark. We can see significant speedups both when using and when not using the DCCM. But when using the DCCM the speedups are even greater. It is also worth mentioning that for this benchmark the multiplication is with saturation enabled. Saturation was implemented with optimized (min, max) defines in the C code and for the SPARROW version, the appropriate buillt-in instructions were used. This benchmark is a prime example of were SPARROW can thrive, since we can take advantage of the fact that SPARROW can calculate the dot product of 4 values with just a single instruction. The first stage calculates the multiplication of the values and the second stage adds the results together. Also, the same instruction can calculate the saturation with no additional cost.

Table 6.2: Speedup results for the matrix multiplication benchmark.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	9,168	4,294	2.13
8	61,996	21,366	2.90
16	468,450	145,630	3.22
32	3,764,663	1,092,748	3.44
64	32,622,775	8,503,118	3.83
128	273,189,693	69,192,579	4.03

Table 6.3: Speedup results for the matrix multiplication benchmark using the DCCM.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	4,078	1,581	2.58
8	26,778	7,327	3.65
16	199,875	46,751	4.28
32	1,631,891	340,255	4.80
64	12,614,395	2,603,426	4.85
128	99,734,715	20,369,698	4.90

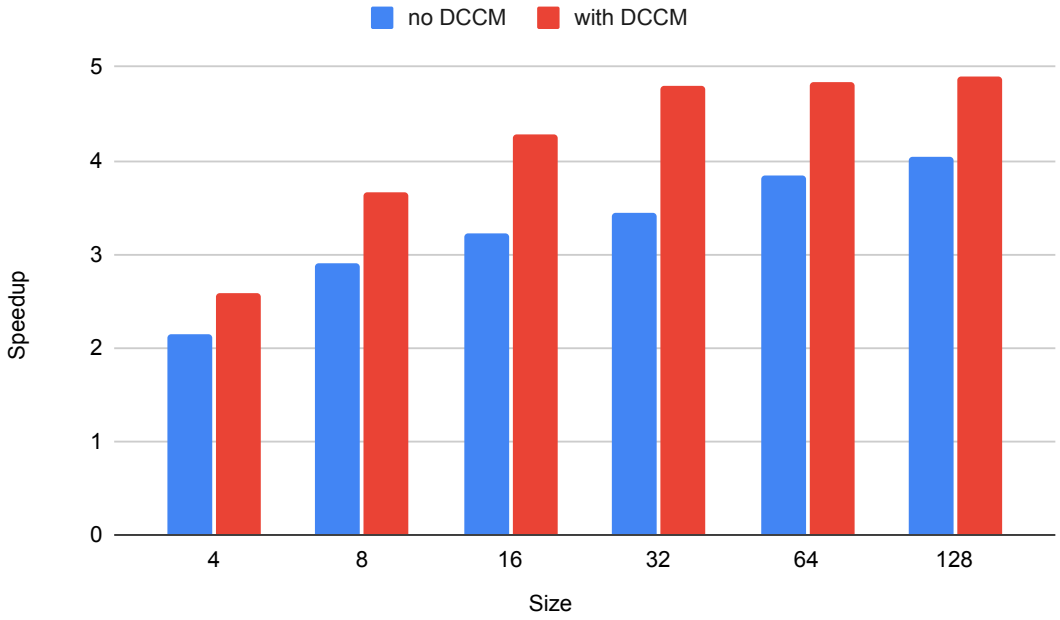

Figure 6.1: Comparison of the speedups with and without the DCCM for the matrix multiplication benchmark

6.2 GREYSCALE BENCHMARK

In Tables 6.4 and 6.5 we can see the results for the greyscale benchmark. This is the benchmark with the lowest speedups, both when using the DCCM and when not using it. There could be many reasons for this, but I have identified 2 main ones. The first reason is that there are dependencies between consecutive SPARROW instructions in the same for-loop of the SPARROW version of the program, something that could not be avoided. The second reason is the fact that this is a very memory intensive program. The source array is a 3D array of dimensions [N][4][N]. Basically, the 4 represents R,G,B,A. This array is enormous and the initialization of this program was constantly 3/4 of the total runtime. I speculate that the memory intensity of the program and the instruction dependencies don't leave much room for improvement. Lastly, it is important to note that although this algorithm could work for RGBA, the input array is in fact RGB and only calculations about the 3 components were made by the C program. For SPARROW nothing changes depending on whether we use 1, 2, 3, or 4 components, but for the C version a fair amount of less instructions per for-loop were made. Had we used all 4 components the speedup would certainly be greater and SPARROW would be fully utilized.

6.3 FILTER (CONVOLUTION) BENCHMARK

In Tables 6.6 and 6.7 we can see the results for the filter benchmark. This benchmark applies a 3x3 convolution kernel over an image. For this benchmark we see similar speedups in both configurations. For this benchmark, there has to be some data manipulation before

Table 6.4: Speedup results for the greyscale benchmark.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	2,363	2,111	1.12
8	7,027	5,227	1.34
16	25,485	17,418	1.46
32	98,860	65,736	1.50
64	395,624	258,251	1.53

Table 6.5: Speedup results for the greyscale benchmark using the DCCM.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	791	789	1.00
8	2,023	1,797	1.13
16	6,802	5,598	1.22
32	25,583	20,513	1.25
64	108,128	75,526	1.43
128	1,382,248	1,251,482	1.36

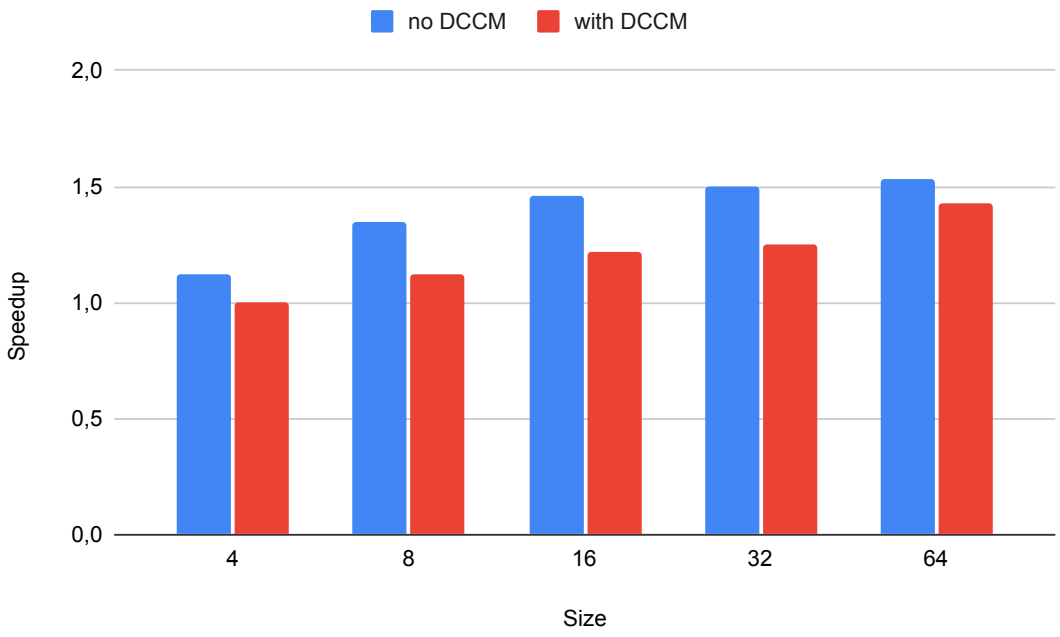

Figure 6.2: Comparison of the speedups with and without the DCCM for the greyscale benchmark

the data are able to be used by SPARROW, so it makes sense that we do not see extreme speedups, although the speedups that we do see are still very satisfactory.

Table 6.6: Speedup results for the filter benchmark.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	28,126	15,851	1.77
8	119,574	62,208	1.92
16	496,873	251,772	1.97
32	2,055,728	1,020,129	2.01
64	8,313,841	4,183,825	1.99
128	33,436,273	16,578,665	2.02

Table 6.7: Speedup results for the filter benchmark using the DCCM.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	8,783	5,733	1.53
8	35,223	21,102	1.67
16	142,912	82,703	1.73
32	720,580	330,025	2.18
64	2,833,171	1,359,283	2.08

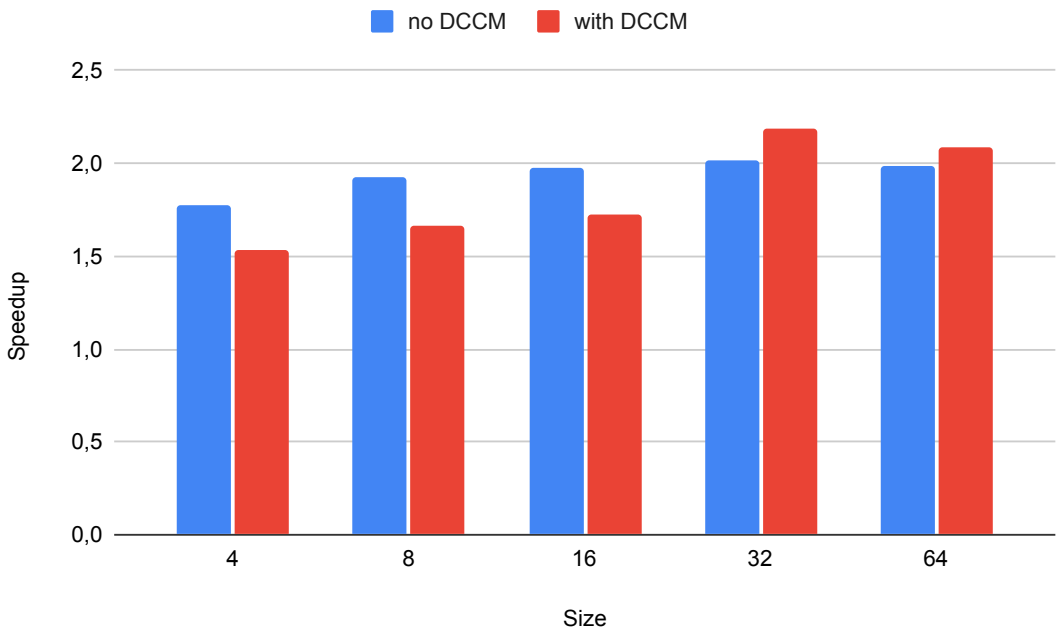

Figure 6.3: Comparison of the speedups with and without the DCCM for the filter benchmark

6.4 POLYNOMIAL BENCHMARK

In Tables 6.8 and 6.9 we can see the results for the polynomial benchmark. For this benchmark we see similar speedups between the 2 configurations, and they are slightly worst when using the DCCM. In this benchmark in the SPARROW version they are 4 continuous SPARROW instructions in the main for-loop and they are dependent from the previous

one (the 2nd from the 1st, the 3rd from the 2nd and the 4rth from the 3rd). Again, it makes sense that for this reason we don't see extreme speedups, although again they are still very satisfactory.

Table 6.8: Speedup results for the polynomial benchmark.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	524	252	2.08
8	932	386	2.41
16	1,748	654	2.67
32	3,390	1,191	2.84
64	6,653	2,263	2.94
128	13,174	4,407	2.99
256	26,230	8,695	3.02
512	52,342	17,271	3.03
1024	104,570	34,437	3.04
2048	209,020	68,733	3.04
4096	422,012	138,365	3.05
8192	843,900	276,605	3.05

Table 6.9: Speedup results for the polynomial benchmark using the DCCM.

Size(N)	SweRV Core EH1 cycles	EH1+SPARROW cycles	Speedup
4	214	141	1.52
8	330	188	1.76
16	562	282	1.99
32	1,021	466	2.19
64	1,950	857	2.28
128	3,805	1,624	2.34
256	7,527	3,160	2.38
512	14,942	6,242	2.39
1024	29,792	12,390	2.40
2048	59,490	24,676	2.41
4096	122,978	49,256	2.50
8192	245,858	98,408	2.50

6.5 SUMMARY OF THE EVALUATION RESULTS

The results are completely satisfactory. There is noticeable speedup for every benchmark, and there is no case where the program slows down. In the matrix multiplication benchmark where we take advantage of the saturation support of SPARROW the results are the best. Theoretically, in a single-issue core, when using SPARROW the maximum spee-

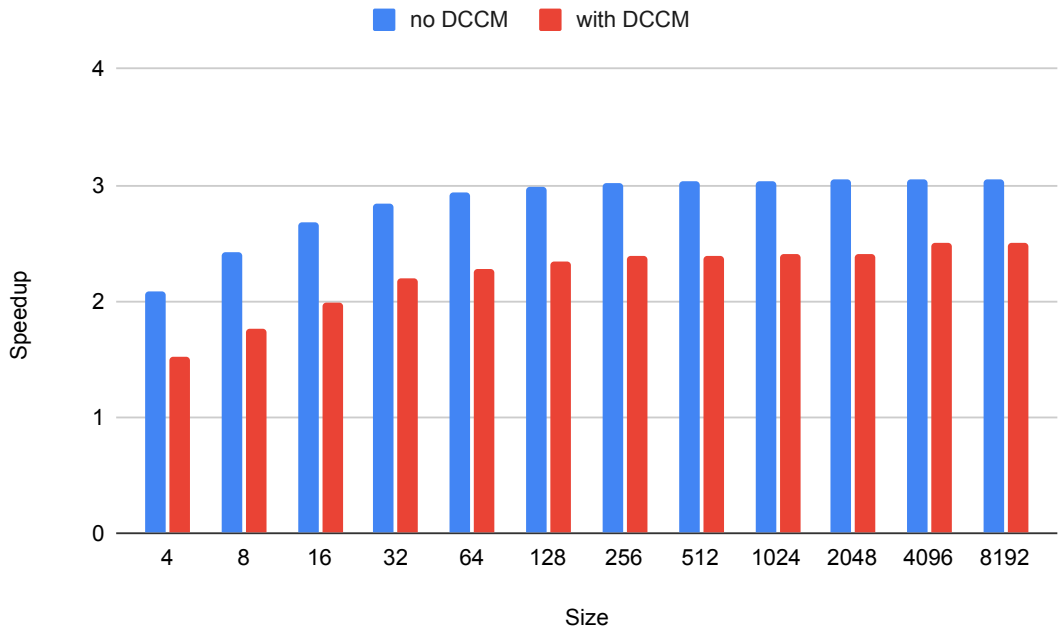

Figure 6.4: Comparison of the speedups with and without the DCCM for the polynomial benchmark

dup we could get would be 4x. That is because one SPARROW instruction does the same operation on 4 different values, something that would take 4 instructions in a simple C program. When utilizing both stages the theoretical maximum speedup gets even higher. But practically, that is not always applicable for various reasons. First of all, there most likely needs to be done some data manipulation in the SPARROW version of the program that will cause some overhead. Also, specifically in this integration of SPARROW in EH1, all SPARROW instructions take a fixed 3 cycles to complete. This especially wastes cycles of when some instructions are ready to go (for example an add that only utilizes the 1st stage of SPARROW).

Furthermore, here we have a dual-issue core, something that reduces the theoretical maximum speedup by a significant amount. It would be naive to say that it goes from 4 to 2, and it needs a very detailed and prolonged analysis to get a good estimation of the actual theoretical maximum speedup, but it is certainly less than 4. Also, it is worth noting that for compiling the benchmarks there was used the llvm version of the compiler support for SPARROW. In Marc Solé Bonet's Master thesis, it was shown that the speedups were worse when using llvm compared to when using the gcc compiler support for SPARROW that exists. It is unfortunate that the gcc compiler support is only for the SPARC ISA version of the SPARROW instructions, and not the RISC-V ones. Considering, all of the above, we can conclude that the results were indeed very good.

6.6 SYNTHESIS

Something very important about the implementation of SPARROW in SystemVerilog and its integration with the SweRV Core EH1 is that it is synthesizable. This reassures the level of quality of the work and helps see the actual physical impact of the integration of SPARROW in EH1. In table In Table 6.10 we can see the results of the synthesis in terms of utilization.

In particular, SPARROW consumes 2,479 LUTs (Look up tables) and 232 flip-flops, which represents an increase of 3.91% and 0.18% respectively over the original EH1 core version. These results confirm that SPARROW's implementation thanks to the reuse of the integer registrer file is very low-cost compared to conventional vector processors.

Table 6.10: SweRV Core EH1 resource utilization comparison for the NEXYS 4 DDR FPGA.

	Available	**SweRV Core EH1**	**SPARROW**
LUT	63,400	32,847(51.81%)	35,326(55.72%)
LUTRAM	19,000	1,129(5,94%)	1,129(5.94%)
FF	126,800	20,425(16.11%)	20,657(16.29%)
BRAM	135	52(38.52%)	52(38.52%)
DSP	240	4(1.67%)	4(1.67%)
IO	210	12(5.71%)	12(5.71%)
MMCM	6	1(16.67%)	1(16.67%)

7

CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

This project was completed successfully. SPARROW has been ported from VHDL to SystemVerilog, and it has been integrated in the SweRV Core EH1 working flawlessly and getting correct results with the 4 benchmarks used for evaluation in the initial SPARROW Master thesis by Marc Solé Bonet. Something that was not necessarily planned at the beginning of the project is that it supports internal bypassing inside the core making work with only the absolutely necessary dependency-caused delays. This way there are achieved noticeable speedups in every benchmark. From this work the portability of SPARROW has been proved and there is now a new alternative in RISC-V embedded systems with an integrated AI accelerator.

7.2 FUTURE WORK

Although this project achieved and even exceeded expectations, there is much more that can be done based on this work. First of all, we will explore the possibility of achieving even greater speedup and improving the integration and/or the implementation if it will be proven helpful. Moreover, there are plans to compare this implementation of SPARROW integrated with EH1, with the VHDL version of SPARROW integrated with LEON3 and NOEL-V, which is going to provide insights about the different benefit achieved by the microarchitecture of each core. Furthermore, there are plans to integrate SPARROW into BSC's Lagarto and SweRV Core variants, EH2 and EL2.

BIBLIOGRAPHY

[1] M. S. Bonet, *Hardware-Software Co-design for Low-cost AI processing in Space Processors*. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/361411/159828.pdf?sequence=1&isAllowed=y.

[2] C. Zarkos, *Cores-Veer-EH1-SPARROW*. [Online]. Available: https://gitlab.bsc.es/czarkos/cores-veer-eh1-sparrow.

[3] C. Zarkos, *sparrow-sv*. [Online]. Available: https://gitlab.bsc.es/czarkos/sparrow-sv.

[4] M. S. Bonet and L. Kosmidis, 'Sparrow: A low-cost hardware/software co-designed simd microarchitecture for ai operations in space processors', in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 1139–1142. DOI: 10.23919/DATE54114.2022.9774730.

[5] M. S. Bonet, *BCC-SPARROW*. [Online]. Available: https://gitlab.bsc.es/msolebon/bcc-sparrow.

[6] A. Waterman, *Design of the RISC-V Instruction Set Architecture*, Jan. 2016. [Online]. Available: https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf.

[7] Zhang Hongsheng et al 2020 J. Phys.: Conf. Ser. 1693 012192, *Design of a dual-issue RISC-V processor*. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1693/1/012192/pdf.

[8] M. Flynn, 'Flynn's taxonomy', in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 689–697, ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_2. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_2.

[9] *VeeR EH1 RISC-V Core*. [Online]. Available: https://github.com/chipsalliance/Cores-VeeR-EH1/tree/main.

[10] J. Abella *et al.*, 'An academic risc-v silicon implementation based on open-source components', in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, 2020, pp. 1–6. DOI: 10.1109/DCIS51330.2020.9268664.

[11] *esperanto.ai*. [Online]. Available: https://www.esperanto.ai.

[12] *GitHub - ztachip/ztachip: Opensource software/hardware platform to build edge AI solutions deployed on FPGA or custom ASIC hardware.* [Online]. Available: https://github.com/ztachip/ztachip.

[13] *riscv-p-spec/P-ext-proposal.pdf at master · riscv/riscv-p-spec*. [Online]. Available: https://github.com/riscv/riscv-p-spec/blob/master/P-ext-proposal.pdf.

[14] *Imagination University Programme*. [Online]. Available: https://university.imgtec.com.

[15] *Glassdoor, 2022*. [Online]. Available: https://www.glassdoor.co.

[16] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan and P. Marwedel, 'Scratchpad memory: Design alternative for cache on-chip memory in embedded systems', in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, ser. CODES '02, Estes Park, Colorado: Association for Computing Machinery, 2002, pp. 73–78, ISBN: 1581135424. DOI: 10.1145/774789.774805. [Online]. Available: https://doi.org/10.1145/774789.774805.

[17] *Programmer's Reference Manual V1.9 for VeeR EH1 core*. [Online]. Available: https://github.com/chipsalliance/Cores-VeeR-EH1/tree/main/docs.

[18] N. P. Jouppi *et al.*, 'In-datacenter performance analysis of a tensor processing unit', *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, Jun. 2017, ISSN: 0163-5964. DOI: 10.1145/3140659.3080246. [Online]. Available: https://doi.org/10.1145/3140659.3080246.

[19] M. S. Bonet, *LLVM-SPARROW*. [Online]. Available: https://gitlab.bsc.es/msolebon/llvm-sparrow.

[20] *CVA6 RISC-V CPU*. [Online]. Available: https://github.com/openhwgroup/cva6.

[21] *Espresso: a Multi-valued PLA minimization*. [Online]. Available: https://ptolemy.berkeley.edu/projects/embedded/pubs/downloads/espresso/index.htm.

[22] M. Johns and T. J. Kazmierski, 'A minimal risc-v vector processor for embedded systems', in *2020 Forum for Specification and Design Languages (FDL)*, 2020, pp. 1–4. DOI: 10.1109/FDL50818.2020.9232940.

Verilog Implementation of a Low-cost Vector AI Accelerator and Integration in a RISC-V Processor,

Author:
Christos ZARKOS

Supervisor:
Dr. Leonidas KOSMIDIS

Institute:
Universitat Politècnica de Catalunya, Barcelona, Spain