Virtualization extension to a RISC-V processor

Author: Gerard van den Berg

Project Director: Juan Jose Costa Prats

Departament d'Arquitectura de Computadors

Grau en Enginyeria Informàtica - Enginyeria de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya –

April 2021

Abstract

This work consist on implementing the hypervisor specification of the RISC-V ISA on an already existing CPU. This includes adding new registers to the CPU, including virtual ones, modifying the interrupt and exception management,implementing new instructions and designing a Two Step Address translation mechanism. The objective of this report is to document the process, and to serve as reference to others wanting to implement it.

Resumen:

Este trabajo consiste en implementar la especificación del hypervisor de la ISA RISC-V en una CPU ya existente. Esto incluye la adición de nuevos registros a la CPU, incluidos los virtuales, la modificación de la gestión de interrupciones y excepciones, la implementación de nuevas instrucciones y el diseño de un mecanismo de traducción de direcciones en dos pasos. El objetivo de este informe es documentar el proceso y servir de referencia a otros que quieran implementarlo.

Resum:

Aquest treball consisteix a implementar l'especificació de l'hypervisor de la ISA RISC-V en una CPU ja existent. Això inclou l'addició de nous registres a la CPU, inclosos els virtuals, la modificació de la gestió d'interrupcions i excepcions, la implementació de noves instruccions i el disseny d'un mecanisme de traducció d'adreces en dos passos. L'objectiu d'aquest informe és documentar el procés i servir de referència a altres que vulguin implementar-lo.

1 Acknowledgements:

First of all, I would like to extend my sincerest thanks to Juan for giving me support and guidance during the full duration of the project.
I would also like to thanks my parents, for their unwavering faith in me, and their confidence that I would be able to pull through.
Also thanks to Max Doblas, from the BSC, without whom I would have been completely unable to undertand the Lagarto Source code and to the support of the Catalan government through the project RIS3CAT DRAC number 001-P-001723.
Finally I would like to thank my friend David, for being by my side for all these years,

Contents

List of Figures

2 Introduction

The objective of this Project is to implement the Hypervisor Extension in an existing RISC-V implementation, in this chapter, we're going to explain the context of the project as well as its motivation and implied parties.

2.1 Context

Virtualization, as a term, appeared in the 1960s and at the time it was used to refer to the concept of dividing a mainframe resources between different applications, however, nowadays it refers to creating a virtual machine, a software that emulates a real computer with its own operative system.
These virtual machines offer numerous advantages over their bare metal counterparts, such as being easy to deploy on different under different hardware configurations, and allowing a single machine to run many virtual machines. Enterprises like Amazon Web Services, use virtualization extensively[1].
However virtualization itself can be quite slow if managed entirely via software, this is why most commercial CPU's offer hardware support to accelerate virtualization, this is done implementing a Hardware level Hypervisor.
An Hypervisor allows a computer system to run multiple guest virtual machines inside one Host system, there exist 2 types of hypervisors:

- Type-1 hypervisors run directly on the host machine hardware, and are often referred as bare metal hypervisors,
- Type-2 hypervisors run like a conventional program inside of the host operative system, for the purposes of this project, we will focus exclusively in the first type.

For this project we will use the RISC-V architecture, because that it is the most well known example of of free and open source hardware. The RISC-V architecture was first developed at the University of Berkeley, California in 2010

[2]. aiming to create a Free and extensible ISA improved by global collaboration, this contrasts with others such as Intel or ARM which have closed sourced ISAs.

2.2 Motivation

The main motivator for this project is the technical challenge and the personal growth opportunity it presents, The skills and knowledge the author will acquire in the fields of specification and implementation of hardware, the RISC-V architecture, as well as virtualization as a whole are of great interest.
Another reason resides in the university context. Being able to understand a specification and build a working implementation based on that makes use of many skills learnt in the Computer Engineering specialization, and the degree as a whole.
Furthermore,this projects documentation could benefit future developers and researchers that aimed to implement a hypervisor in their own RISC-V implementation, as well as a general introduction to RISC-V itself for all kinds of research and development.

2.3 Stakeholders

The development of this project will involve different people, some of those will be detailed:

- **Project author:** I, will be in charge to design, implement and testing the hypervisor, as well as documenting the whole process.

- **Future Hypervisor implementer:** The aim of this project is also to act as a reference document for anyone that wanted to implement the RISC-V hypervisor in the future.

- **Barcelona Super Computing Center:** Given the fact that the *Lagarto* is one of the possible RISC-V implementations to be chosen for this project, their assistance may be needed, furthermore, a successful Hypervisor implementation could be useful in future endeavors.

- **Project Director:** This projects director will be supervising the development of the project, as well as provide aid when necessary.

2.4 Justification:

At the time of writing, an open source Type-1 Hypervisor hasn't been implemented yet, while it is true that many commercial CPUs implement such an hypervisor, I think that there is value in an hypervisor which implementation is completely transparent and auditable by any third party.
Being the spec itself not final, a working implementation would also be suitable to evaluate the specification itself

3 Project Planing

In this chapter, we'll go over the different aspects of the project, such as its scope, The methodology used, the expected risks, its planned budget, and sustainability concerns.

4 Scope and Obstacles

This section will explain the objectives of this project, as well as some possible obstacles and pitfalls to surpass or avoid.

4.1 Reach: objectives and subobjectives

The main objective of this project is the implementation of the RISC-V Hypervisor Specification on an existing RISC-V Implementation, this has yet to be done, as of the start of this project the specification is not final, and the target processor is yet to be chosen, finally it is also my objective to document exactly what is needed to implement the hypervisor for future reference.
All in all the objectives needed to complete the project are the following:

- Study the Hypervisor specification.
- List requirements for the target implementation.
- Choose an initial RISC-V implementation.
- Study the implementation.
- Design the Hypervisor implementation.
- Implement the design and tests.
- Document the process.

4.2 Risks

During the project development, different obstacles may occur that could hinder the process, I will proceed to detail them and possible solutions.

4.2.1 Implementation issues

It is necessary to consider the possibility that the current state of the chosen processor implementation has insufficient support for the Hypervisor, extending the development time excessively.
Solution: Look for additional implementations for backup will allow me to avoid this.

4.2.2 Lack of technical knowledge:

Given that a lot of the project consists on research of both the specification and the chosen implementation, the time estimates are bound to be more speculative.
Solution: The best strategy to try to mitigate this issue is to plan around this lack of information.

4.2.3 Excessive Design Scope:

It is possible that during the investigation phase, the design requirements end up being too large.
Solution: An analysis of the essential features will be made in order to cut non essential requirements.

5 Methodology and Rigor

This section will detail both the work methodology and the tools used to monitor the progress.

5.1 Methodology

Taking into account the breadth and complexity of the Hypervisor specification and related material, the project will be separated in 3 parts.

- Analysis of the specification to identify what needs to be implemented
- Analysis of the chosen processor.
- Design and implementation of the hypervisor.

To archive that, we will apply an agile methodology based on "scrum" strategy, this will consist of doing small "sprints" each week with small doable objectives, then at the start of of the week a reunion will be held with the project director and new objectives will be added to the next sprint based on the previous sprint performance.

5.2 Monitoring tools

For the Design and Implementation of the Hypervisor, I'll use git, an open source version control system, to track and safeguard the progress in the implementation.
I'll also keep a diary-like document detailing the tasks done to facilitate writing of the documentation. The project will be successful if a working Hypervisor is implemented. this will be tested using tests provided by the RISC-V foundation [3] as well as my own.

6 Task Description

In the following section I will define the tasks that need to be performed as well as the expected completion time, due to the investigative nature of the project, the expected time to completion as my vary as well as the number and nature of the tasks.

6.1 Project Management:

This group of tasks involve the management of different aspects of the project.

6.1.1 Scope

this section is important, as it involves defining the scope of the project as well as defining its general structure, this tasks will take 10 hours to ensure the project foundation is solid.

6.1.2 Planning

This task consists of making a timeline of the project tasks as well as designing alternative plans and forecasting obstacles, this task will take 5 hours as part of this task is already done in **Scope**

6.1.3 Cost and Sustainability

This tasks are about documentation and can be done concurrently, each individual task should take the same as the planning, 5 hours.

6.1.4 Documentation

An important part of the project is to document it for future reference, as such an hour will be dedicated each week to document the week progress, as well as any insights, this should take 15 hours total.

6.2 Agile

As Agile is planned in sprints (Short week long intervals) we need a task to plan how to implement Agile **General Planning** and a set of recurrent tasks that plan each sprint **Sprint Planning, QA Reunion and Control Reunion**

6.2.1 General Planning

This task involves scheduling reunions with the Project director as well as planning how to use Agile in the project, this should take 5 hours.

6.2.2 Sprint Planning

In order to make use of the Reunions with the Project Director, half an hour should be set aside to prepare this reunions and the plans for the next sprint.

6.2.3 QA Reunion and Control Reunion

The reunions can be QA Reunions, Control Reunions or both, QA Reunions are about checking the quality of the work being done, while the Control Reunions are about the work to be done, because I expect an even number of both, I expect that I'll need to dedicate 15 hours in total in reunions.

6.3 Spec Analysis

The First part of the project involves Reading and understanding the RISC-V Specification, this will be divided in two tasks:

6.3.1 Reading The Privileged Spec

This task consists of reading the main body of interest, The Privileged Hypervisor specification, reading and synthesizing its contents should take 15 hours given the complexity of the material but it's relative short length

6.3.2 Reading Additional Specs

Given the technical nature of the main document, it will become necessary to cross reference other RiSC-V Specs, like the Supervisor SPEC, the Standard SPEC or the Debugging SPEC, this should take an additional 5 hours and will be done concurrently with **Reading the Privileged Spec**

6.4 Target System Evaluation

After reading the documentation, the next task consists in looking at the already existing implementations and choosing the most fit for the task, I expect 2 o 3 candidates to evaluate and about 7 hours of work.

6.5 Setting up Development Environment

Once the Target has been chosen, I need to set up a development environment, this task may include, setting up a Linux Virtual Machine, installing specific computer software, including tooling and testing software, and other possible necessities, due to the large unknown component of this task I estimate 15 hours to complete it.

6.6 Designing the solution.

This part is both the most opaque one and probably the most time demanding one, it requires to design the necessary hardware modifications that need to be implemented, due to the amount and nature of such changes currently being unknown, I make an estimate of 45 hours, this task will start once the Dev environment is close to being complete.

6.7 Implementing the solution

After the Design is done, it will be time to implement it, given that the design is done properly, the implementation should be done somewhat faster, henceforth

I estimate 30 hours.

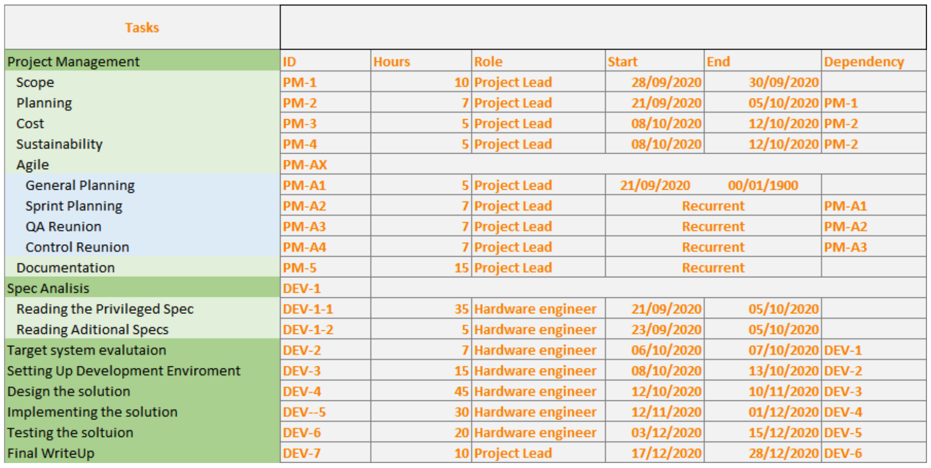
6.8 Testing

This task will both test the correctness of the solution, as well as its performance, this task will take 20 hours and serve as a buffer in case more time is needed in **Designing The solution** or **Implementing the solution**

6.9 Final WriteUp

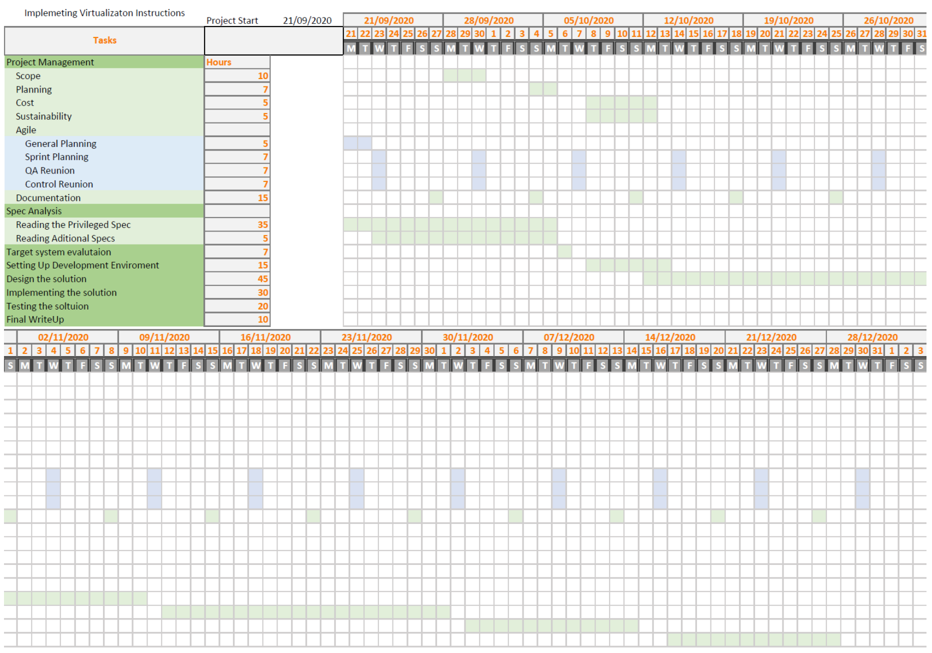
This task is to take all the documentation and the results of the **Testing** task and write the final document, I want to dedicate 20 hours to that task.

7 Gantt graph



Tasks						
Project Management	ID	Hours	Role	Start	End	Dependency
Scope	PM-1	10	Project Lead	28/09/2020	30/09/2020	
Planning	PM-2	7	Project Lead	21/09/2020	05/10/2020	PM-1
Cost	PM-3	5	Project Lead	08/10/2020	12/10/2020	PM-2
Sustainability	PM-4	5	Project Lead	08/10/2020	12/10/2020	PM-2
Agile	PM-AX					
General Planning	PM-A1	5	Project Lead	21/09/2020	00/01/1900	
Sprint Planning	PM-A2	7	Project Lead	Recurrent		PM-A1
QA Reunion	PM-A3	7	Project Lead	Recurrent		PM-A2
Control Reunion	PM-A4	7	Project Lead	Recurrent		PM-A3
Documentation	PM-5	15	Project Lead	Recurrent		
Spec Analisis	DEV-1					
Reading the Privileged Spec	DEV-1-1	35	Hardware engineer	21/09/2020	05/10/2020	
Reading Aditional Specs	DEV-1-2	5	Hardware engineer	23/09/2020	05/10/2020	
Target system evalutaion	DEV-2	7	Hardware engineer	06/10/2020	07/10/2020	DEV-1
Setting Up Development Enviroment	DEV-3	15	Hardware engineer	08/10/2020	13/10/2020	DEV-2
Design the solution	DEV-4	45	Hardware engineer	12/10/2020	10/11/2020	DEV-3
Implementing the solution	DEV--5	30	Hardware engineer	12/11/2020	01/12/2020	DEV-4
Testing the soltuion	DEV-6	20	Hardware engineer	03/12/2020	15/12/2020	DEV-5
Final WriteUp	DEV-7	10	Project Lead	17/12/2020	28/12/2020	DEV-6

Figure 1: Gantt summary


Implemeting Virtualizaton Instructions

Project Start 21/09/2020

Tasks	Hours
Project Management	
Scope	10
Planning	7
Cost	5
Sustainability	5
Agile	
General Planning	5
Sprint Planning	7
QA Reunion	7
Control Reunion	7
Documentation	15
Spec Analysis	
Reading the Privileged Spec	35
Reading Aditional Specs	5
Target system evalutaion	7
Setting Up Development Enviroment	15
Design the solution	45
Implementing the solution	30
Testing the soltuion	20
Final WriteUp	10

Figure 2: Gantt Graph

8 Risk Management:

Due to the nature of this project, it is very possible that certain tasks take longer than initially expected, especially those after **Reading other Specs**, in other to mitigate that, I designed the testing phase to be open ended and thus easily re sizeable also I overestimated all linearly dependant tasks outside of project management to account for unexpected obstacles. in the event that the progress starts to slow down during sprints, the amount of hours will be increased by 50% for the following week.

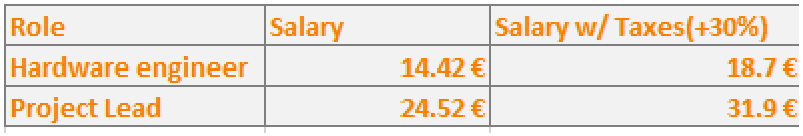
Given a 25% risk of that happening on each of the 15 sprints on average it will lead to a possible increase of up to 62.6 hours to the project.

9 Budget

In this section, we'll examine the economic cost of the project, in order to do that, we'll analyze all the aforementioned task of the project, as well as ”generic costs” that would apply to all steps of the project.

9.1 Human Cost

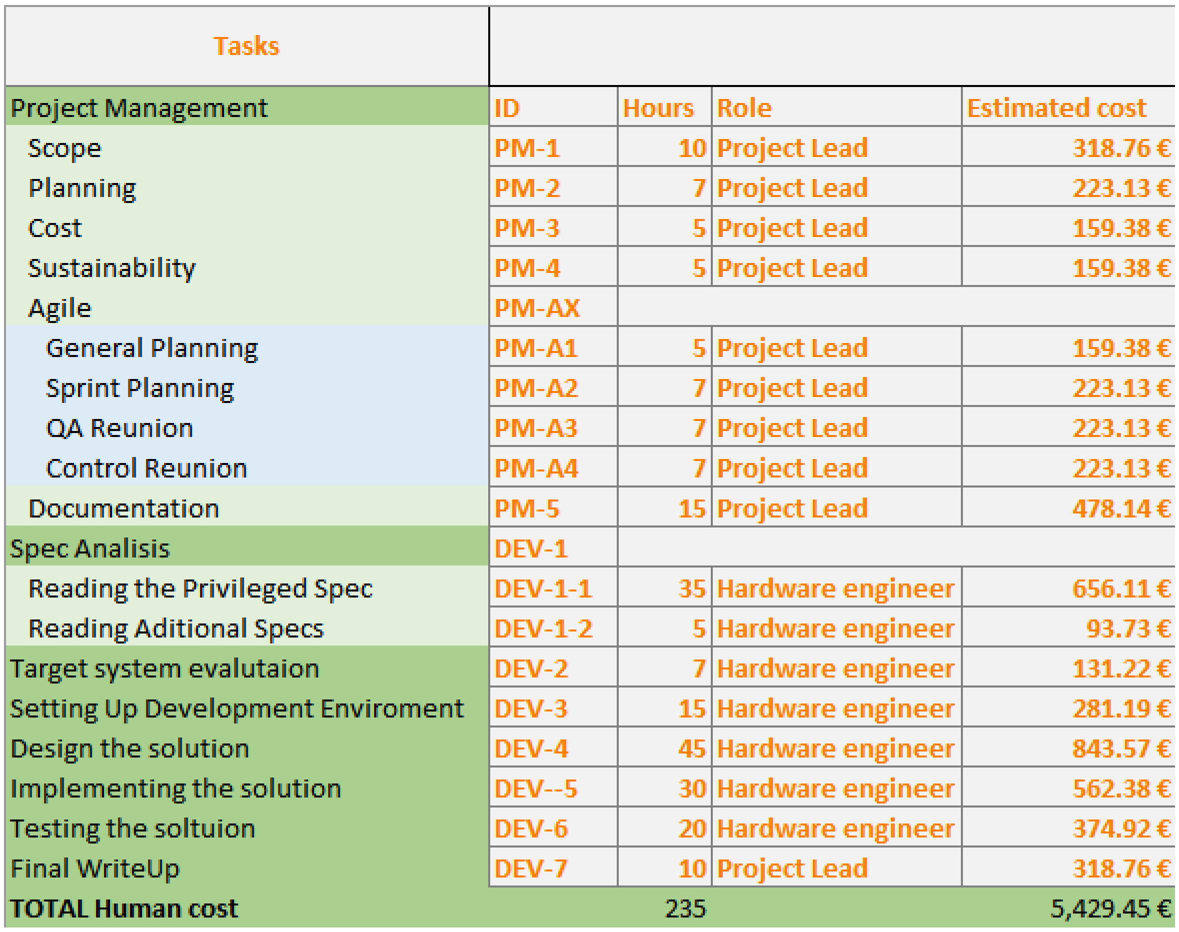
First, we'll employ the social network Linkedin to obtain the average salary of a Project Lead and a Hardware engineer [4], the two essential roles for this project. While I considered a tester role for the testing task, it would not be adequate as most of the task involves work that resembles more that of an engineer that a dedicate testing role.



Role	Salary	Salary w/ Taxes(+30%)
Hardware engineer	14.42 €	18.7 €
Project Lead	24.52 €	31.9 €

Figure 3: Role Salary/h

We use this data to calculate the human cost of each task.



Tasks				
Project Management	ID	Hours	Role	Estimated cost
Scope	PM-1	10	Project Lead	318.76 €
Planning	PM-2	7	Project Lead	223.13 €
Cost	PM-3	5	Project Lead	159.38 €
Sustainability	PM-4	5	Project Lead	159.38 €
Agile	PM-AX			
General Planning	PM-A1	5	Project Lead	159.38 €
Sprint Planning	PM-A2	7	Project Lead	223.13 €
QA Reunion	PM-A3	7	Project Lead	223.13 €
Control Reunion	PM-A4	7	Project Lead	223.13 €
Documentation	PM-5	15	Project Lead	478.14 €
Spec Analisis	DEV-1			
Reading the Privileged Spec	DEV-1-1	35	Hardware engineer	656.11 €
Reading Aditional Specs	DEV-1-2	5	Hardware engineer	93.73 €
Target system evalutaion	DEV-2	7	Hardware engineer	131.22 €
Setting Up Development Enviroment	DEV-3	15	Hardware engineer	281.19 €
Design the solution	DEV-4	45	Hardware engineer	843.57 €
Implementing the solution	DEV--5	30	Hardware engineer	562.38 €
Testing the soltuion	DEV-6	20	Hardware engineer	374.92 €
Final WriteUp	DEV-7	10	Project Lead	318.76 €
TOTAL Human cost		235		5,429.45 €

Figure 4: Task Human cost

The estimate human cost of the project is 5,429.45 euros.

9.2 Generic Cost

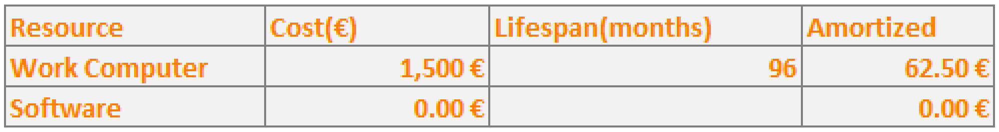
9.2.1 Resources

First of all, the main resource used in this project will be the authors personal computer, valued in 1,400 euros (peripherals included) and an expected life span of 8 years.

All software used on the project is free open source software, with a total cost of 0 euros.

Amortized cost: In order to calculate the amortized cost of the computer we use the following formula

$$Amortized\ cost = resource\ cost * \frac{project\ span}{resource\ lifespan}$$



Resource	Cost(€)	Lifespan(months)	Amortized
Work Computer	1,500 €	96	62.50 €
Software	0.00 €		0.00 €

Figure 5: Resource cost

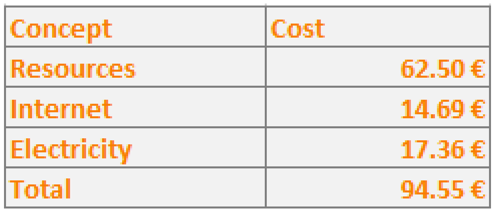
9.2.2 Non Amortizable costs

internet: Using a 45€/month contract during the 4 months of the project for a total of 180€ given that only 235h out of 4 month of internet are used for the project the cost is: 180*(235/(4*30*24))= 14.69 euros

Electricity: Using this resource [5] we establish the kWh cost at 0,147743 euro. assuming I use the computer for every task of the project (235h) and given the fact that the computer uses 500W, henceforth 0,147743*500*235/1000=17.36 euros

9.2.3 Total

The total generic cost is the sum of the previously mentioned costs:



Concept	Cost
Resources	62.50 €
Internet	14.69 €
Electricity	17.36 €
Total	94.55 €

Figure 6: Total generic costs

9.3 Contingency

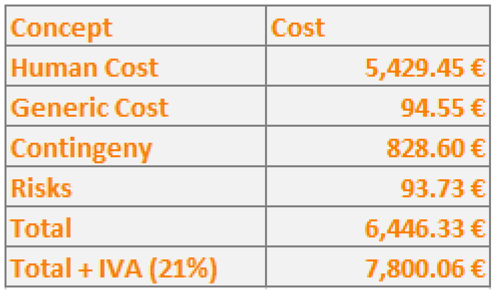
As it is customary in this types of projects we should add a 15% contingency, taking into account the big variability of the project, 828,60 euros.

9.4 Risks

The main risks exposed previously is the risk that the time allocated proved to be insufficient, the main mitigation strategy consists on reallocating testing hours to the overdue task, if that is insufficient, additional time will be allocated, up to 20 additional hours as a hardware engineer (374.92 euros), given the nature of the project I think there is at least a 25% chance of this occurring, given an estimated cost of 93.73 euros.

9.5 Total

Adding up all together we get the following budget:



Concept	Cost
Human Cost	5,429.45 €
Generic Cost	94.55 €
Contingeny	828.60 €
Risks	93.73 €
Total	6,446.33 €
Total + IVA (21%)	7,800.06 €

Figure 7: Total Budget

9.6 Control

Using the Agile methodology allow us to readjust the hours dedicated to each task as well as keep track of the time and resources spent on each sprint.

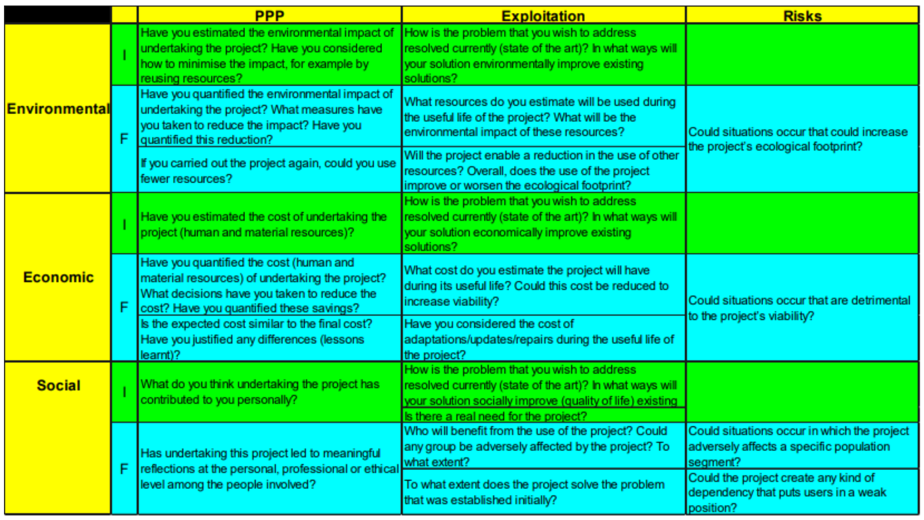
10 Sustainability report

10.1 Survey

Before starting the report, a survey was answered in order to become more aware of subjects related to sustainability in engineering projects.

The survey contained questions about knowledge about sustainability as a concept as well as techniques for identifying and preventing issues of that nature. Mainly due to a lack of experience on the subject, and little to none coverage of this subjects in the curriculum, the survey reflected a lack knowledge in the field, these results weren't surprising.

The next sections will cover sustainability from different perspectives, it is based on the following sustainability matrix.



		PPP	Exploitation	Risks
Environmental	I	Have you estimated the environmental impact of undertaking the project? Have you considered how to minimise the impact, for example by reusing resources?	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?	
	F	Have you quantified the environmental impact of undertaking the project? What measures have you taken to reduce the impact? Have you quantified this reduction?	What resources do you estimate will be used during the useful life of the project? What will be the environmental impact of these resources?	Could situations occur that could increase the project's ecological footprint?
		If you carried out the project again, could you use fewer resources?	Will the project enable a reduction in the use of other resources? Overall, does the use of the project improve or worsen the ecological footprint?	
Economic	I	Have you estimated the cost of undertaking the project (human and material resources)?	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?	
	F	Have you quantified the cost (human and material resources) of undertaking the project? What decisions have you taken to reduce the cost? Have you quantified these savings?	What cost do you estimate the project will have during its useful life? Could this cost be reduced to increase viability?	Could situations occur that are detrimental to the project's viability?
		Is the expected cost similar to the final cost? Have you justified any differences (lessons learnt)?	Have you considered the cost of adaptations/updates/repairs during the useful life of the project?	
Social	I	What do you think undertaking the project has contributed to you personally?	How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution socially improve (quality of life) existing	
			Is there a real need for the project?	
	F	Has undertaking this project led to meaningful reflections at the personal, professional or ethical level among the people involved?	Who will benefit from the use of the project? Could any group be adversely affected by the project? To what extent?	Could situations occur in which the project adversely affects a specific population segment?
			To what extent does the project solve the problem that was established initially?	Could the project create any kind of dependency that puts users in a weak position?

Figure 8: sustainability matrix

10.2 Environmental perspective

What resources have you used during the lifespan of the project? What is the environmental impact of those resources?

The main resource used is a personal computer, The main environmental impact is its electrical usage. **Have you estimated the environmental impact of undertaking the project? Have you considered how to minimise the impact, for example by reusing resources?**
I have not quantified the the environmental impact of the used resources, and neither have I taken measures to minimize them.
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?
Different CPU vendors have their own hypervisor implementations, my solution, while not being directly an improvement could be made more environmentally friendly by any third party given that the project is open source.
What resources could reuse other projects?
Everything related to the project is open source and could be used by other RISC-V implementators either in the BSC or elsewhere.

10.3 Economic perspective

Have you estimated the cost of undertaking the project (human and material resources)
This project includes a budget and the deviations of said project

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?
This solution offers a free Hypervisor implementation, which would be an improvement over the closed source proprietary ones currently on the market.
Is there a collaboration with another project?
While not an explicit collaboration, the project may be integrated the in the Lagarto project of the BSC.

10.4 Social perspective

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

This project has given me experience in the hardware field, as well as a greater understanding on how to manage this kind of projects.

Is there a real need for the project?

I think that having an open source hypervisor is important, as open source hardware is both really important and surprisingly scarce.

11 The RISCV

In this section we are going to discuss the ISA itself as well as some of its more relevant characteristics for our project.
The term RISC itself means Reduced Instruction Set Computer, a type of computer design that prioritizes a simple, small, fixed length instruction set, with explicit load and store instructions in contrast with a CISC (Complex Instruction Set Computer) that tends to have a large number of instructions of various word sizes,the most common desktop PC architecture x86 is a CISC. What distinguishes the RISC-V architecture among other RISCs and most ISAs in general, is that it is a royalty free, open and free ISA, allowing anyone to develop, and sell their own RISC-V implementations. This coupled with a large amount of existing support, makes it an attractive offer for hardware developers and researches alike.
The base design of the ISA is focused on small, low energy, general usage CPUs, however, the ISA has a number of extensions, that allow things such as: Integer multiplication and division, single and double precision floating point operations, atomic operations and more.
One of the extensions, and the one we will be focusing on, is the H extension - Standard Extension for Hypervisor, that allows for the implementation of both Type-1 and Type-2 Hypervisors.

12 RISC-V Privileged Instruction Set - Standard Extension for Hypervisor

As said before, virtualizing without hardware support is really slow and resource intensive. The Hypervisor Extension aims to improve virtualization performance, mainly by reducing amount of interrupts and exceptions that need to be handled by the Host OS.

The approach taken is to virtualize supervisor mode(S mode), changing the existing supervisor mode into an Hypervisor-extended supervisor mode (HS mode) and adding both virtual user mode (US mode) and virtual supervisor mode (VS mode), additionally, a second stage is added to the address translation mechanism, virtualizing the memory and memory mapped I/O devices for the guest OS.

12.1 Trap Handling

RISC-V features a trap delegation mechanism that allows different privilege levels to handle interrupts and exceptions instead of M-mode. this is done by selecting which exceptions are delegated to a lower privilege level, going from M to HS to VS. This allows the Guest OS to handle exceptions by itself.

12.2 Two-Stage Address Translation.

When virtualization is enabled, all memory accesses go through two stages, on the first one, the virtual address is translated into a *guest physical address* this stage is known as VS-stage, then these address is translated again to a supervisor physical address, this is known as the G-stage, when performing this step all accesses are considered U-mode accesses, even those preformed on VS-mode data structures, a guest page-fault must be handled by either M or HS and cannot be delegated further.

12.3 Registers

The specifications details number of registers specific to the Extension, those registers and their functionality, are explained in figure 9.

Register	**Description**
hstatus	Contains information for tracking and controlling exception behavior
hedeleg	Configure exception delegation HS->VS.
hideleg	Configure interrupt delegation HS->VS.
hvip	Trigger virtual interrupt intended for VS.
hip	Shows pending VS or Hypervisor-specific interrupts.
hie	Shows enabled VS or Hypervisor-specific interrupts.
hgeip&hgip	Control external interrupts directed to the guest OS f.e(passthrought)
hcounteren	Controls the availability of specific hardware counters.
htimedelta(l/h)	Controls the Δ between the time read from the time register and the time displayed in VS-VU modes.
htval	additional information for trap handling.
htinst	information about the instruction that trapped.
hgatp	controls the guest stage of the page translation behavior.

Figure 9: Hypervisor specific registers.

Additionally several supervisor specific registers need to have a copy that will be accessed when virtualization is enabled, this is further discussed in Chapter 16.

12.4 New instructions

12.4.1 Virtual-machine load and store

The Virtual machine load and store instructions are HS-mode or M instruction that perform a memory access as if the virtualization was enabled, that means performing the Two-Stage translation, depending on the configuration of *hstatus* register, the protection of the guest pages may be overridden. this allows the host to access the Guest OS address space.

12.4.2 Fences

Coalesces all the memory accesses that precede the instruction, needed when enabling/disabling virtualization to avoid the Guest and Host trying to write to each other memory address spaces.

13 Choosing the right implementation.

While there are many implementations of the Base RISC-V architecture, the amount of them with an implemented S-mode, was sparse, and didn't include most of the reference implementations,which ended up being very problematic for the project, This led to Lagarto, a RISC-V implementation with Multicore support and capable of booting Linux, this CPU is being developed by the DRAC partnership, from which the BSC is partner of, one of their recent versions, in internal development at the time of writing. does implement S-mode, making it the most viable implementation.

14 The Development environment

. The Lagarto uses several programming languages that need their specific tool chain in order to be able to build the processor.

14.1 Chisel

Chisel is a Hardware Description Language used to facilitates circuit generation for ASICs and FPGAs.[6] the language adds primitive to the Scala language, Because it is based on Scala it runs in the Java Virtual machine, we need a java development kit and running environment in order to build it, However, Chisel cannot build the processor if the java version ¿= 8.
Most of the code currently on Chisel is being ported to SystemVerilog and none of the code we need to change is written in Chisel, so no more tools are needed.

14.2 SystemVerilog

SystemVerilog is another Hardware Description Language, considered to be an evolution on Verilog, Adding many features to the later, For the purposes of the project, we'll take advantage of the new types, more specifically, the *reg* or *logic* type for declaring registers, and *enum*s that allow us to give names to constants, useful to avoid having to hardcoded memory addresses or bit masks on the code, greatly improving readability.
While many options exist to compile SystemVerilog, the project already makes use of Verilator, a SystemVerilog simulator/synthesizer that transpiles to C++/C [7].

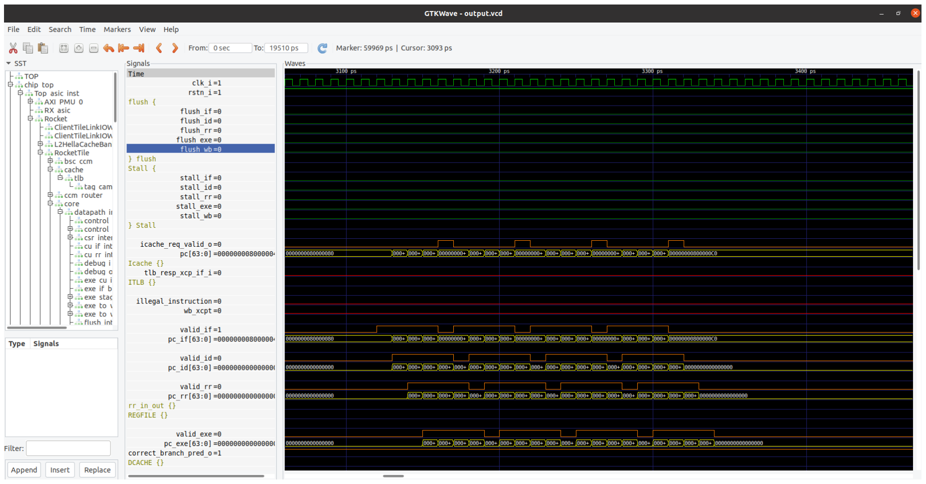
14.3 GNU toolchain

Finally, the tests for this platform are written in RISC-V asm and C. Given that the development machine is not a RISC-V computer, it is necessary to install a cross-compiler, a program that is able to compile for an architecture different from the systems native architecture.

The RISC-V Foundation itself provides with a a cross compiler [8]. The toolchain, however, failed to compile the project if built with gnu-gcc version 8 or above, while the reason is currently unknown, it is suspected to be a bug with the generated verilator code.

14.4 Testing software.

In order test our changes to the codebase. the project provides a Simulator that generates traces that can be load in and analyzed using software like GTKWave.



```
Time
              clk_i=1
             rstn_i=1
flush {
            flush_if=0
            flush_id=0
            flush_rr=0
           flush_exe=0
            flush_wb=0
} flush
Stall {
            stall_if=0
            stall_id=0
            stall_rr=0
           stall_exe=0
            stall_wb=0
} Stall

   icache_req_valid_o=0
            pc[63:0]=000000008000004
Icache {}
    tlb_resp_xcp_if_i=0
ITLB {}

  illegal_instruction=0
             wb_xcpt=0

            valid_if=1
         pc_if[63:0]=000000008000004

            valid_id=0
         pc_id[63:0]=000000000000000

            valid_rr=0
         pc_rr[63:0]=000000000000000
rr_in_out {}
REGFILE {}

           valid_exe=0
        pc_exe[63:0]=000000000000000
correct_branch_pred_o=1
DCACHE {}
```

Figure 10: Gtkwave loading the results of a test.

15 Lagarto Lowrisc

The lagarto is a enormous project, boasting around 260K lines of code Most of which is a mix of C, SystemVerliog, Scala and assembly. However, for this project, we'll only need to touch one specific system, the CSRs, in this project, the CSRs are all implemented in a single file, the csr_bsc.sv with additional definitions in the riscv_pkg.s

16 Implementing Hypervisor mode

in this chapter, we'll design and implement the Hypervisor itself, this task can be narrowed down into the following subtasks:

1. Add the new CSRs to enable support for the rest of the featuers,
2. Expand the Trap Handling Mechanism.
3. Add a second step to the Address Translation
4. Add the new Instructions, defined in the specification.

16.1 New CSRs

CSRs are Control and Status registers, memory units that control the behavior of the CPU. Reading from them allows the developer to know the current status of the CPU, for example, what kind of interruption or exception the CPU is handling at the moment, and writing to them allows them to control it, enabling interruptions, different timers and counters etc. In order to access those registers, a program needs to be running at the appropriate mode, and execute the the CSRR or CSRW instruction, to read or write to the registers.
Adding CSRs requires editing the SystemVerilog file itself to add them. in the language point of view, a register is described as two binary arrays of bits, the _q array and the _d array. The register is read from the _q array, and it is written to the _d array. at each clock cycle. the contents of the _d array are written to the _q array, this is done this way to ensure that the registers read remains coherent for the duration of the cycle. In figure 11, we can see the new CSRs declarations.

```
//Hypervisor
//virtual Registers
logic [63:0] vsscratch_q,      vsscratch_d;
logic [63:0] vsip_q,      vsip_d;
logic[63:0] vsie_q,    vsie_d;
logic[63:0] vsstatus_q,    vsstatus_d;

logic [63:0] vstvec_q,      vstvec_d;
logic [63:0] vstatus_q,      vstatus_d;
logic [63:0] vsepc_q,      vsepc_d;
logic [63:0] vscause_q,      vscause_d;
logic [63:0] vstval_q,      vstval_d;
logic [63:0] vsatp_q,      vsatp_d;

//Hypervisor - Trap Handling
riscv::hstatus_rv64_t hstatus_q,    hstatus_d;
logic[63:0] hedeleg_q,    hedeleg_d;
logic[63:0] hideleg_q,    hideleg_d;
logic[63:0] hie_q,    hie_d;
logic[63:0] hcounteren_q,    hcounteren_d;
logic[63:0] hgeie_q,    hgeie_d;
logic[63:0] htval_q,    htval_d;
logic[63:0] hip_q,    hip_d;
logic[63:0] htinst_q,    htinst_d;
logic[63:0] hvip_q,    hvip_d;
logic[63:0] hgeip_q,    hgeip_d;
logic[63:0] htimedelta_q,    htimedelta_d;
logic[63:0] htimedeltah_q,    htimedeltah_d;
```

Figure 11: CSR declarations

However, sometimes we need to access individual bits of the array, and while it is possible to do so using bitmasks, or accessing a particular bit using its index in the array, SystemVerilog allows use to declare structured arrays with proper names for each of the bits. as seen in figure 12, where declare a type for hstatus, in which we include the different bits, in figure 11, we can see how hstatus is declared as a hstatus_rv64_t type instead of just a logic array like the others.

```
    typedef struct packed {
        logic [63:34] wrpi0;
        logic [1:0]   vsxl;
        logic [8:0]   wrpi1;
        logic         vtsr;
        logic         vtw;
        logic         vtvm;
        logic [1:0]   wrpi2;
        logic [5:0]   vgein;
        logic [1:0]   wrpi3;
        logic         hu;
        logic         spvp;
        logic         spv;
        logic         gva;
        logic         vsbe;
        logic [4:0]   wpri4;

    } hstatus_rv64_t;
```

Figure 12: Hstatus type declaration

We also need to be able able to tell whenever virtualization is enabled, or disabled, the specifics on how to do it are not actually defined in the specification, for this reason, I decided to add an extra CSR named virt or V, that will store this information. In addition to the aforementioned hypervisor specific registers in figure 9, we need to add extra registers to act as the Supervisor mode registers in guest mode, figure 13 contains those registers and their S-mode equivalent.

CSRs	**S-mode Equivalent**
vsstatus	*sstatus*
vsip	*sip*
vsie	*sie*
vstvec	*stvec*
vsscrach	*sscratch*
vsepc	*sepc*
vscause	*scause*
vstval	*stval*
vsatp	*satp*

Figure 13: Virtual CSR

Given that we need to access both sets of registers, depending if virtualization is or not enabled, we need to implement a mechanism that allows us to access them both, Figure 14 shows us a possible design, in which we add a multiplexer to each of the registers in Figure 13 and use our custom CSR as the selector.

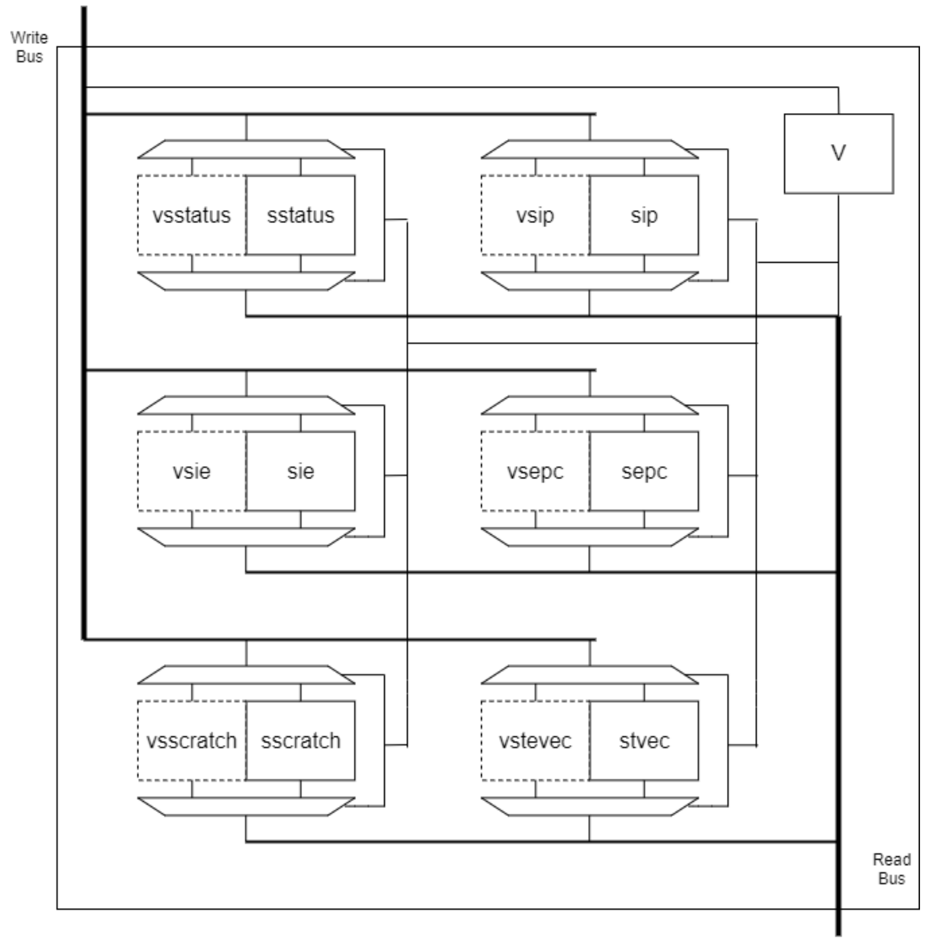


Figure 14: Using Multiplexers to access the virtual registers.

When virtualization is enabled all accesses to the S-mode CSRs will access their virtual counterparts, this will be completely transparent to the guest. However, this design presents a couple of problems, first, because each register has its specific address, including the virtual ones we just made, we would need specific hardware to access them explicitly, also, from a programming standpoint, we would need to add this multiplexer every time any time a CSR from the figure 13 table is accessed, which is both error prone and time consuming, Then we came up with another design.

Another read through the specification revealed an important fact, The addresses for the Virtual registers are at a fixed offset of 0x100 from their non virtual counterparts, for example, *sie* is accessed at 0x104, while *vsie* is accessed at 0x204, with that information in mind, Figure 14 design proved to be a better approach,

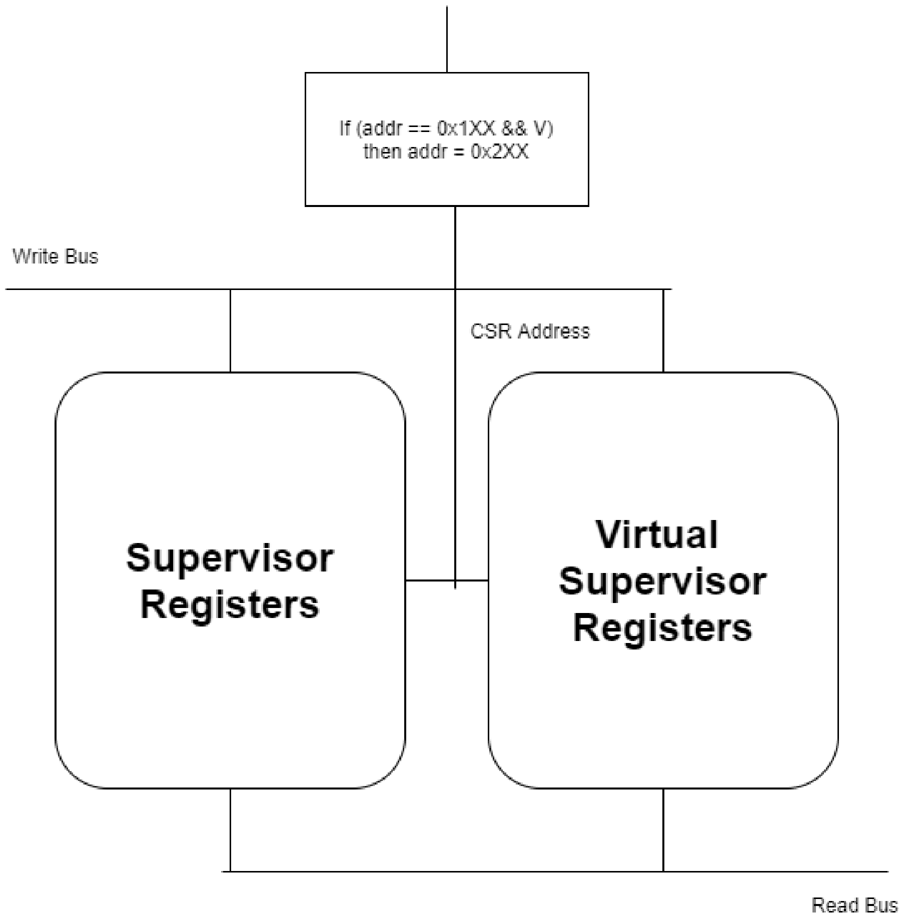


Figure 15: Address altering approach

now, we will read the address that the CPU wants to read, and if its a supervisor specific to S-mode and virtualization is enabled, The address will be shifted from the 0x1?? to the 0x2?? address, and the access will continue normally, this approach will still allow us to access the virtual registers explicitly, without any special hardware.

The performance of both systems should be about the same, and the code to implement it is a simple if statement placed in

```
riscv::csr_t  actual_addr;
csr_read // a valid read
csr_write // a valid  write
csr_addr // the original address
...
 if (csr_read|csr_write && virt_q && csr_addr.address[11:8] ==
4'h1)
        begin
            actual_addr.address[7:0] = csr_addr.address[7:0];
            actual_addr.address[11:8] = 4'h2;
        end
    else
    begin
        actual_addr = csr_addr;
    end
```

Figure 16: Code used to change the address while reading or writing a register

16.1.1 Reading and Writing to CSRs

In the Project the CSR interface exposes two buses for I/O, those are the r_data_core_iand *r_data_core_o* respectively, note that some CSRs (such as mstatus, are also exposed directly, luckily, this does not apply to the registers we need to implement. This is done using 2 switch cases, in one of them, you read the address of the requested CSR and fetch the data from the register, in the writing case, the output assignments are made, both to the output bus, and to the CSRs themselves, given that we alter the address itself, the process only consists in adding our newly created registers to those switch cases, as seen figure 17.

```
//virtual
riscv::CSR_VSSTATUS: begin
    csr_rdata = mstatus_q & def_pkg::SMODE_STATUS_READ_MASK;
end
riscv::CSR_VSIE:                csr_rdata = mie_q & mideleg_q &
    hedeleg_q;
riscv::CSR_VSIP:                csr_rdata = mip_q & mideleg_q &
    hideleg_q;
riscv::CSR_VSTVEC:              csr_rdata = vstvec_q;
riscv::CSR_VSSCRATCH:           csr_rdata = vsscratch_q;
riscv::CSR_VSSCRATCH:           csr_rdata = vsscratch_q;
riscv::CSR_VSEPC:               csr_rdata = vsepc_q;
riscv::CSR_VSCAUSE:             csr_rdata = vscause_q;
riscv::CSR_VSTVAL:              csr_rdata = vstval_q;
riscv::CSR_VSATP:               csr_rdata = vsatp_q;
//hyper
riscv::CSR_HSTATUS:           csr_rdata = hstatus_q;
riscv::CSR_HEDELEG:           csr_rdata = hedeleg_q;
riscv::CSR_HIDELEG:           csr_rdata = hideleg_q;
riscv::CSR_HIE:               csr_rdata = hie_q;
riscv::CSR_HCOUNTEREN:           csr_rdata = 64'b0;
riscv::CSR_HTVAL:           csr_rdata = htval_q;
riscv::CSR_HIP:           csr_rdata = hip_q;
riscv::CSR_HVIP:           csr_rdata = hvip_q;
riscv::CSR_HTINST:           csr_rdata = htinst_q;
riscv::CSR_VIRT:           csr_rdata = virt_q;
```

Figure 17: Read switch case snippet

Additionally, because the original code uses intermediate variables to store the values of the CSRs, we need to intercept those as well, as figure 18 shows.

```
if(virt_q) begin
            mstatus_d   = mstatus_int;
            mcause_d    = mcause_int;
            vscause_d    = scause_int;
            mtval_d     = mtval_int;
            vstval_d     = stval_int;
            mepc_d      = mepc_int;
            vsepc_d      = sepc_int;
        end
        else
        begin
            mstatus_d   = mstatus_int;
            mcause_d    = mcause_int;
            scause_d    = scause_int;
            mtval_d     = mtval_int;
            stval_d     = stval_int;
            mepc_d      = mepc_int;
            sepc_d      = sepc_int;

        end
```

Figure 18: Intercepting writes

16.1.2 Testing

In order to test the new registers, we program a test that, with the virtualization register enabled, will write to a supervisor specific register, for example sscratch, if the system works, reading from its virtual equivalent vsscratch, should return the same value. The code of figure 19 does that, first it enables virtualization writing 1 to the virt_q CSR [1]. then loads to a temporal register the value 1[2], then writes it to the sscratch register[3], then reads explicitly from the vsscratch register[4], if the CSRs work correctly the value from vsscratch will be the same as the value we written in sscratch (1), if not the test fails [5], else the test succedes[6] In the code of figure

```
csrw 0x60C, 1 //virt
li t1, 1
csrw sscratch, t1
csrr 0x240, t2 // vsscratch
bne t1,t2, fail
j pass
```

Figure 19: Simple Test for the new Virtual Registers

.

16.1.3 Modifying existing CSRs

In addition to the new registers, it is necesary to do minor modifications to the already existing registers, figure 20 shows which

Register	**Changes**
misa	set the 7th bit to show that the Hypervisor extension is implemented
mstatus	Add fields MPV and GVA to keep track of the previous virtualization mode and the guest page address when a trap is taken in M mode. Change behaviour of TSR and TVM.
mideleg	Set bits 10,6 and 2, if guest external interrupts are enabled,
mip and me	Add additional active bits.
mtval2	Modify to support guest page fault exceptions.
mtinst	Add support for the new trapped instructions.

Figure 20: Reference of CSRs to modify

16.2 Trap Handling.

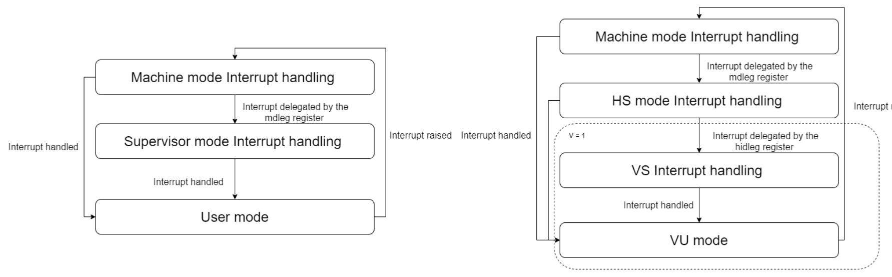

Machine mode Interrupt handling
Interrupt delegated by the mdleg register
Supervisor mode Interrupt handling
Interrupt handled
Interrupt handled
User mode
Interrupt raised
Interrupt handled
Machine mode Interrupt handling
Interrupt delegated by the mdleg register
HS mode Interrupt handling
Interrupt raised
V = 1
Interrupt delegated by the hidleg register
VS Interrupt handling
Interrupt handled
VU mode

Figure 21: Trap handling functionality

The trap handling needs to be expanded so that the existing interrupts can be delegated from the existing S mode (now HS) to to VS mode. if we decompose this further, what we need is the following:
First, we need to enter M mode and save the previous virtualization mode, from the virt CSR to the spv bit from *hstatus*.
Then we need to check if the exception is delegated to hs mode, this is done checking against *mideleg* or *medeleg*
only after that, we check if the exception has been further delegated, comparing against the hsdeleg CSRs.
Then we reneable virtualization, and load the specific handler, which will run inside the guest OS.

16.3 Two Step translation

- Implement *vstap* and *hgatp* to configure the pages.
- Implement *vsstatus* MXR field.
- Change the memory protection to take into account the guest memory protection schema
- add support for Guest Page Faults.

16.4 New Instructions.

16.4.1 Hypervisor Load and Store Instructions

For every load and store instruction exists a corresponding Virtual load and Store, these instructions are only valid in M or HS mode unless user mode Hypervisor is enabled.
The following CSRs are accessed:

CSR	**Field**	**Explanation.**
hstatus	**HU**	make the instruction legal in user mode.
	SVPV	Controls the privilege level of the access. VU if 0 VS othewise.
sstatus	**MXR**	makes execute-only pages readable on both translation stages.
vstatus	**MXR**	makes execute-only pages reable on the first stage of the translation.
∼Same aceesses as a regular load∼		

Figure 22: Reference of CSRs accessed by the Hypervisor Load/Store instructions

17 Deviation from initial planning

In this section, we'll go over the different tasks, and compare the estimates with how things turned out.

17.1 Project Management:

This section was allocated 68 hours total, and in the end it may have been an overestimate, the planning stages took fewer ours than expected, however, due to the extended guideline, more sprints than expected happened. leading to a slight overtime of 4 hours.

17.2 Spec analysis.

This part went without issue,However, the analysis revealed a larger number of subtasks at the design and implementation tasks than expected. The time spent was the 40 hours budgeted.

17.3 Choosing the Target System

This part had many complications, for one, there were multiple, delays accessing the Lagarto project and the alternative processors that we considered were mostly non-starters, most of them due to lacking a implementation of Supervisor mode, essential, for the hypervisor spec, and the development environments being hard to set up. this alone put a wrench in the project planning, and the time was extended to 25 hours, as different processors were evaluated without success until we were granted access to the project.

17.4 Development environment

Outside of a couple of difficulties with bugs and other technical difficulties, we didn't suffer many delays on this part, and because of that, we didn't go overtime.

17.5 Designing the Solution

I completely underestimated the complexity of the codebase, and how much out of my element I'll be, I spend a very large amount of time getting myself familiar with the it and its complexities, I ended up taking up close to 120 hours,

17.6 Implementing and Testing the solution.

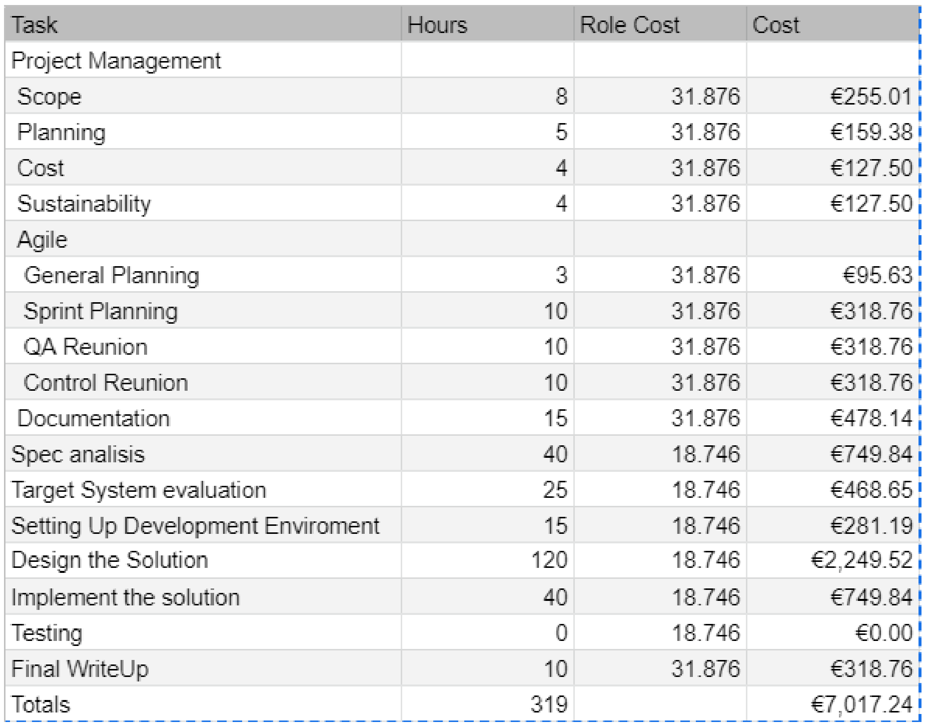
This task proved to be to large for the time remaining, a such I initially decided to cut the implementation. Even with that, only the new CSRs were successfully implemented successfully. this also took longer than expected, about twice the hours, 40.

17.7 Final Write up.

The write up itself has been completed on the allocated time,

17.8 Final Cost of the Project.

At first we budgeted 6.446.33€ to the project, including an 828.60€ of contingency, do to the extended deadline, the cost exceeded the budget, From figure 7, the generic cost remains the same, as the cost increase comes down to the increased hours.



Task	Hours	Role Cost	Cost
Project Management			
Scope	8	31.876	€255.01
Planning	5	31.876	€159.38
Cost	4	31.876	€127.50
Sustainability	4	31.876	€127.50
Agile			
General Planning	3	31.876	€95.63
Sprint Planning	10	31.876	€318.76
QA Reunion	10	31.876	€318.76
Control Reunion	10	31.876	€318.76
Documentation	15	31.876	€478.14
Spec analisis	40	18.746	€749.84
Target System evaluation	25	18.746	€468.65
Setting Up Development Enviroment	15	18.746	€281.19
Design the Solution	120	18.746	€2,249.52
Implement the solution	40	18.746	€749.84
Testing	0	18.746	€0.00
Final WriteUp	10	31.876	€318.76
Totals	319		€7,017.24

Figure 23: Human cost table

As seen in the table, the Additional Human Cost eats through our contingency, Taking it into account, it puts our project at $Budget - OriginalHumanCost + NewHumanCost - Contingency = (7017.24 - 5429.45) + 6446.33 - 828.63 = 7205.49$€, including a 21% IVA 8718.64€ I would like to note that while the Risk were successfully detected, The mitigation strategies proved to be insufficient.

18 Conclusion and future work

The main objectives of the project were the following:

- Study the Hypervisor specification.
- List requirements for the target implementation.
- Choose an initial RISC-V implementation.
- Study the implementation.
- Design the Hypervisor implementation.
- Implement the design and tests.
- Document the process.

From that list, most objectives have been fulfilled up and including a design for each of the Hypervisor components, however, due to a variety of circumstances, we were unable to complete the implementation task, Of course, the process itself has been documented in this very document.

From a personal standpoint, I have acquired a wealth of knowledge in quite a few areas, including how to interpret a specification, about Hypervisors, How to deal with large scale projects. and modifying real hardware.

As for future work, the current implementation is a proof of concept, another person interested in implementing this extension would benefit from the design insights, however my inexperience in SystemVerilog makes me believe that a better implementation is possible.

19 Bibliography

References

[1] *What is Amazon EC2? - Amazon Elastic Compute Cloud*. Amazon.com, 2020. URL: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html` (visited on 10/19/2020).

[2] *History - RISC-V International*. RISC-V International, Sept. 2020. URL: `https://riscv.org/about/history/` (visited on 09/29/2020).

[3] riscv. *riscv/riscv-tests*. GitHub, 2020. URL: `https://github.com/riscv/riscv-tests` (visited on 10/19/2020).

[4] *LinkedIn Salary - Overview — LinkedIn Help*. Linkedin.com, 2020. URL: `https://www.linkedin.com/help/linkedin/answer/85616/linkedin-salary-descripcion-general?lang=e` (visited on 10/19/2020).

[5] Cómo funciona la tarifa fija anual. *Tarifaluzhora.es*. tarifaluzhora.es, 2015. URL: `https://tarifaluzhora.es/info/tarifa-fija-anual-electricidad` (visited on 10/12/2020).

[6] the Chisel/FIRRTL Developers. *Chisel/FIRRTL: Home*. 2021. URL: `https://www.chisel-lang.org/`.

[7] 2021. URL: `https://www.veripool.org/wiki/verilator`.

[8] riscv. *riscv/riscv-gnu-toolchain*. Apr. 2021. URL: `https://github.com/riscv/riscv-gnu-toolchain`.