Universidade do Minho
Escola de Engenharia

Hugo José Duarte Ribeiro

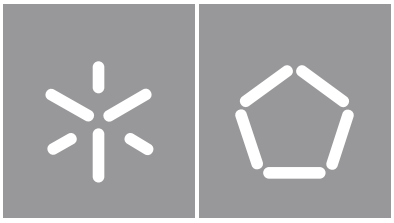
Adaptive Quantum for Parallel Full System Simulation

julho de 2023

Adaptive Quantum for Parallel Full System Simulation

Hugo José Duarte Ribeiro

UMinho | 2023



Universidade do Minho
Escola de Engenharia

Hugo José Duarte Ribeiro

Adaptive Quantum for Parallel Full System Simulation

Dissertação de Mestrado
Mestrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação de
Professor Doutor Jorge Miguel Nunes Santos Cabral
Professor Doutor Rainer Leupers

julho de 2023

Acknowledgements

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Abstract

In the recent years, the multiprocessor system on a chips (MPSoCs) complexity has been growing exponentially, however the performance of simulation tools are not following this growth, mainly because of their sequential simulation type. Therefore, simulation time increases each time a new MPSoC is developed. The Institute for Communication Technologies and Embedded Systems (ICE) Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen team developed a parallel version of the atomic mode of Gem5. It's based on a synchronous Parallel Discrete Event Simulation (PDES) which allows each simulation thread to run independently from the rest of the system for a time $t_{\Delta q}$ – the so-called quantum.

This dissertation aims to solve the problem of finding the optimal quantum, which can lead to the best trade-off between accuracy and performance. Nowadays, Gem5 is bound to a static quantum which cannot be changed during run-time, although an adaptive quantum can overcome the limitations of the static version. Thus, the main objective was to develop a flexible algorithm that can operate independently the used benchmark.

Keywords: Parallel Discrete Event Simulation, Gem5, Full- System Simulation, Quantum

Resumo

Nos ultimos anos, a complexidade dos MPSoCs tem crescido exponencialmente, contudo as ferrementas de simulacao destes nao tem seguido este crescimento, principalmente devido ao seu tipo de simulacao sequencial. Nesse sentido, os tempos de simulacao aumentam sempre que um novo MPSoCs é desenvolvido. A equipa do ICE RWTH Aachen desenvolvou uma versao simulacao paralela to modo atomico do Gem5. Esse modo é baseado numa simulacao paralela sincrona de eventos discretos (PDES) que permite cada thread de simulacao executar independentemente do resto do sistema por um tempo $t_{\Delta q}$, cujo nome é quantum.

Esta dissertacao tem como objetivo solucionar o problema de encontrar o quantum otimo, ou seja, o quantum que permita obter o melhor trade-off entre a performance e a precisao do simulacao. De momento, a versao oficial do Gem5 apenas permite a defenicao de um quantum estatico, pelo que este nao pode ser alterado durante o decorrer da simulacao, porém um quantum dinamico consegue ultrapassar as limitacoes do anterior. Portanto, o principal objetivoo é desenvolver um algoritmo flexivel que consiga operar independentemente do benchmark utilizado.

Palavras-chave:Simulacao Paralela de Eventos Discretos, Gem5, Simulacao de Sistemas Completos, Quantum

METER NO WORD PARA CORRIGIR ERROS E ACENTOS

Contents

List of Figures

List of Tables

Glossary

ADC	Anallog-to-Digital Converter
ANC	Adaptive Noise Cancellation
ANN	Artificial Neural Network
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BNN	Biological Neural Network
CES	Continuous Event Simulation
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CS	Context Switch
DAC	Digital-to-Analog Converter
DES	Discrete Event Simulation
DMA	Direct Access Memory
FFT	Fast Fourier Transform
FIFO	First-In First-Out
FS	Full-System
FSS	Full System Simulator
GEMS	General Execution-driven Multiprocessor Simulator
GPIO	General Purpose Input Output
GUI	Graphical User Interface
HDL	Hardware Description Language
I/O	Input/Output
ICE	Institute for Communication Technologies and Embedded Systems
IP	Intellectual Property
IPC	Inter Process Communication
ISA	Instruction Set Architecture
KPI	Key Performance Indicator
KVM	Kernel Virtual Mode

LED Light-Emitting Diode
LMS Least Mean Square
LoC Lines of Code
MIPS Millions-of-Instructions-Per-Second
ML Machine Learning
MPSoC multiprocessor system on a chip
NIC Network Interface Controller
NN Neural Network
NPB NAS Parallel Benchmarks
OS Operating System
PC Program Counter
PDES Parallel Discrete Event Simulation
POSIX Portable Operating System Interface
PTE Page Table Entry
RAM Random Access Memory
RTL Register-Transfer Level
RWTH Rheinisch-Westfälische Technische Hochschule
SDES Sequential Discrete Event Simulation
SE System-call Emulation
SoC System-on-a-Chip
SP Stack Pointer
SWaP-C Size, Weight, Power, and Cost
TD Temporal Decoupling
TDL Tapped Delay Line
TLM Transaction-Level Modeling
UART Universal Asynchronous Receiver-Transmitter
VP Virtual Platform

1 Introduction

A normal industrial design process is divided into five stages: define, ideate, prototype, production and deliver. One characteristic of this process is its non-linearity. There are an interaction between the ideate, prototype, and production stages, so-called interaction design. This happens because problems or upgrades can be found, leading to step back in the design process [1].

One example where this design process happens is in the development of an Application Specific Integrated Circuit (ASIC). As the name suggests, an ASIC is used for specific applications where dedicated hardware is required, for example, a critical system of a car. After the design, it's developed the prototype, on which will be running tests, or benchmarks, in order to understand if the developed ASIC satisfy all the requirements. Only in this stage it's possible to obtain some indicators such as power consumption or compute performance to evaluate if Size, Weight, Power, and Cost (SWaP-C) requirements match. However, with Virtual Platforms (VPs) or Full System Simulators (FSSs) it's possible to have those without the physical prototype. Therefore, the time to market can be accelerated because problems or upgrades can be spotted much sooner.

Thus, these simulation tools are useful for the design of modern massively parallel and complex multi-core systems. Nevertheless, the major problem is that, typically, many of these simulators can't execute a parallel simulation, in other words, they only execute the workload in one thread. Also, the complexity of the new systems increases due to the integration of more and more applications on a single chip [2], leading to unacceptable simulation times, for example, the case of the SPEC2017 integer benchmark, where it may take up to two years to complete the simulation [3].

1.1 Motivation

Gem5 is one FSS that cannot only execute a simulation with multiple threads. To solve this, the ICE RWTH Aachen team developed par-gem5 [3], a parallel version of the atomic mode of Gem5, that exploits the multithreading capabilities of modern host systems. It's based on a synchronous PDES which allows the parallelization of the system. Synchronizations are done periodically, according a defined time, so-called quantum or quanta.

High quantum allow for high simulation speeds, but negatively impact the simulation's accuracy, or, in the worst case, even break the system's functionality. If the quanta is too small, the accuracy is perfect,

although the simulation performance will be unsatisfactory. Thus, there is a tradeoff between accuracy and performance, and finding an optimal quantum is one of the main challenges when running synchronous PDES.

In the current state of par-gem5 (and as in other frameworks), the quantum is set once and then kept for the rest of the simulation. This brings several different problems. First of all, to know which is the best quantum, it's necessary to do the simulation in order to obtain the simulation results, and further evaluate if it was the best choice or not. Moreover, the quantum varies simulation to simulation, therefore one can be the best for a case, but for another not so much. All this try-and-error consumes a lot of time, thus this isn't the optimal option.

1.2 Goals and Contributions

A dynamic quantum could address these issues by adjusting the quantum value for each simulation in real time, leading to improved results. This approach is particularly beneficial for typical benchmarks that consist of multiple phases with distinct compute and synchronization characteristics, that is, for the computational part, the quantum can be increase, for the synchronization part, the quantum should be reduced.

In the context of par-gem5, with dissertation development it will be possible to automatic tune to the best quantum, without any user inputs or feedback. Furthermore, the quantum adaptation must be "on-the-fly" and be independent of the simulated system or benchmark, hence the algorithm must be flexible.

On top of that, the simulator, in the end of the benchmark execution, should give feedback to the user, by the creation of a statistics document. It also must include information related to the adaptive quantum, for example, the mean of the used quantums, by the reason of understanding how the algorithm performed.

In the end, the dynamic quantum should bring more advantages than the static version. It is expected that this algorithm solves the problem of finding the best compromise between performance and accuracy, allowing speedups in different simulations, and making it possible to simulate massively parallel and complex systems faster, without a break in the accuracy.

Regarding an industry point of view, this work will grant a faster development of new products, in such a way companies can be the market leaders of the technology. Time-to-market can is optimized since the product is finished earlier, giving a room of maneuver to commercialize it at the right moment, increasing the revenues.

1.3 Dissertation Outline

This section I will write when the thesis structure is approved

2 State of the Art

This chapter establishes the needed concepts to accomplish and understand this work. First of all, it will start with the simulation topic, what it is, and why it is so important. Embedded systems and machine learning are briefly contextualized, highlighting the connection they share with the previous topic. Then, simulation types and modes are introduced, with a special focus on the ones that will be used in this dissertation. Finally, the Gem5 simulator is presented in detail, since it is the framework where all the development will be done.

2.1 Simulation

Simulation is a very recent topic in the literature. The graph presented below attests to the fact that this subject began to be studied solely in the early 1950s. Prior to the advent of computers, it was unfeasible to replicate anything without the actual object or prototype, thereby rendering this subject of relatively low significance in the industry. However, with the introduction of computers, a new realm of opportunities emerged as a "virtual world" was conceived. As computers became smaller, more robust, and more cost-effective, a panoply of novel technologies and topics began to flourish, including simulation.

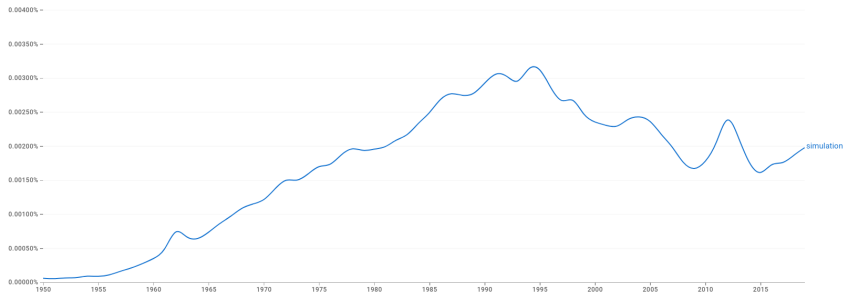


Figure 2.1: Evolution of the simulation topic in the literature by google books ngram viewer

Analyzing the market research report by ZION, is possible to conclude that the global virtual training and simulation market size was valued at $332.6 Billion in 2022 and is projected to reach $973.4 Billion by 2030.

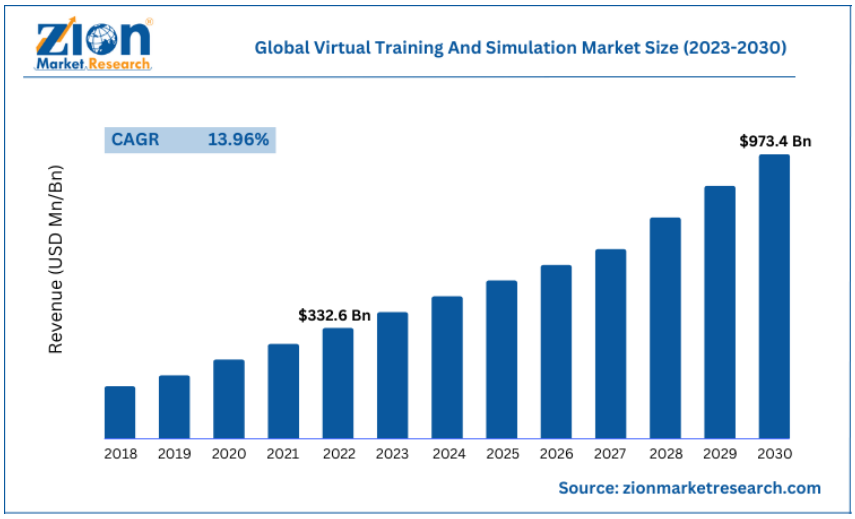


Figure 2.2: Market research report by ZION about simulation [4]

This growth and demand in the industry turn this topic into an important subject. For this reason, it is important to understand what is simulation and why is it necessary. In the following points, this will be covered in detail.

2.1.1 Definition

Accordingly the Cambridge dictionary, simulation can be defined as *a situation or event that seems real but is not real, used especially in order to help people deal with such situations or events*. Moreover, simulation can be also defined, in a more general way, as *an imitation of a system* [5]. Therefore, simulation is not exclusively confined to computers but also encompasses a diverse range of applications, whether physical or virtual. For instance, in the automotive industry, each new model that is designed must undergo a security test. One of the numerous examinations performed involve simulating a car crash into a wall and assessing the driver's resulting damages. As a result, conclusive determinations can be made, specifically in this case, regarding whether the car has successfully met the test's requirements or not.

Hence, simulation is a controlled verification technique which helps to understand how a system would behave in a real situation. As previously mentioned, simulation can be physical or virtual. Nowadays, the last is the preferable one, since it brings huge advantages when compared to the other. First of all, the cost. It is clear that physical simulations, like the one previously described, have a lot of associated costs. The car that is going to be destroyed, the workers to make sure everything goes as planned and to clear all the wreckage, and even the infrastructure, that require a designated area which involves costs and space. The second reason is the time. In a competitive industry, the first to have the product in the market can be the one that will have more success. Physical simulation can take a lot of time. It may be necessary, beyond the simulation itself, preparation, post-cleaning, license acquisition, weather conditions, etc. delaying the process. The last factor is the computer's evolution. As noted earlier, its evolution was crucial in the way that more complex simulation environments can be tested and obtain more accurate results.

To elaborate on the concept of virtual simulation, it can be classified into three categories. This classification is illustrated in the Figure 2.3, which outlines the taxonomy of virtual simulations.

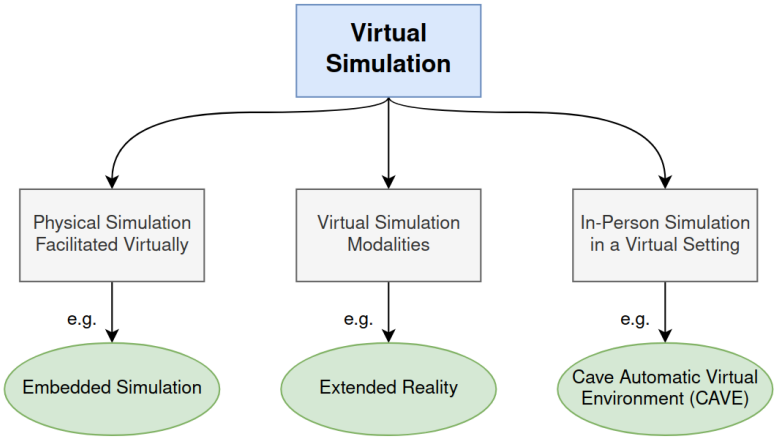


Figure 2.3: Taxonomy of virtual simulations (Adapt from [6])

As shown, it can be divided into physical simulation facilitated virtually, virtual simulation modalities, and in-person simulation in a virtual setting.

Physical simulation facilitated virtually is the process of utilizing virtual simulation to replicate a physical phenomenon or system. This is done through the creation of a digital model that imitates the physical object, and through various algorithms and computational methods, simulates its behavior in a virtual environment. In this way, it is possible to predict how the physical system will behave in various scenarios, without the need for physical testing, thereby saving time and resources. It is used, for example, in the development of an embedded system. Without it, it would be mandatory to have the physical prototype to evaluate if the system matches all the requirements.

This dissertation will focus on computer architecture simulation. Therefore, the concept of simulation can be redefined as *the imitation of the operation of a real-world process or system over time* [7]. The computer architecture itself becomes the system under study, and simulation enables the user to explore its capabilities and limitations in a simulated environment.

2.1.2 Importance of Simulation

To have a full scope of why simulation is important, three topics should be considered: how the nature of the system affects the simulation, and the advantages and disadvantages of this technique compared to other validation methods [5].

The first topic exhibits three characteristics: variability, interconnections, and complexity. Consider the production of a Central Processing Unit (CPU) chip as an example. The production process may have interconnections with other factors such as the chip manufacturers or the chip designers. These interconnections can make it difficult to predict the overall performance of the system, which in this example is the

CPU chip production, especially when variability is present. Furthermore, systems can be complex, and these complexities make it challenging to predict the performance of a system when changes are made or actions are taken. With simulation models, these problems can be solved in a way that they can explicitly represent the variability, interconnectedness, and complexity of a wanted system.

Keeping the example of CPU chip development, the process goes through various stages and, in the end, it is crucial to test the product before mass production. Thus, a prototype is built so tests can be made. However, building a prototype takes time and money. In the majority of cases, the prototype can be improved, so the developers need to go through the process again. A new cycle begins, which means spending more time and money. Simulation can solve this problem in the way that development and testing can be done in parallel. In other words, while the developers are creating the chip, it can be tested in simulation, even without the physical prototype. Moreover, simulation is flexible, which means small changes can be done easily, and it is controlled, that is, it can control the experimental conditions in the way direct comparisons can be done, and is transparent, so there are no subjective results.

In terms of drawbacks, it is possible that the development team consists solely of individuals specialized in CPU chip design. Therefore, the team would need to hire a new employee with expertise in simulation to fulfill the requirements. Furthermore, it requires powerful computers and a huge sufficient storage capacity, as simulation results can be in the order of dozens of gigabytes. The software to run the simulation and the creation of the model to test require investments. All these aspects represent an enormous investment for the company, where it will not get a direct profit. On top of that, simulations are not 100% accurate, meaning that they can produce results that are not the correct ones.

Considering all the aspects, simulation brings a lot of benefits when compared to other techniques, such as experimentation in real life. Nonetheless, the downsides must be taken into account and concluded whether this solution fits the needs or not. Banks in [8] mentions examples of applications where it can be used, like computer system performance, food processing, transportation systems, and embedded systems.

2.2 Embedded Systems

Embedded systems have a ubiquitous presence in modern life, from cars, computers, and televisions to smartphones and even toothbrushes. It is so common and present everywhere that the world without it wouldn't be the same. While this topic is not new to academia, with books like [7] mentioning it as early as the 19th century, it has gained more attention and deeper analysis since the 1980s until nowadays.

The following topics explain more about how can embedded systems be defined, and which are the development processes used in the industry. In the end, a specific topic about simulation in embedded systems will be covered.

2.2.1 Definition

The definition of embedded systems is not straightforward, that is, there is no agreement in the literature mostly because this term is being under development every day. One possible definition can be a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function [9]. Other authors prefer to highlight their relevant properties, like Raul Camposano in [10], where he focuses on correctness, real-time, and dedicated functions, among others, as the important aspects to retain when an embedded system is mentioned.

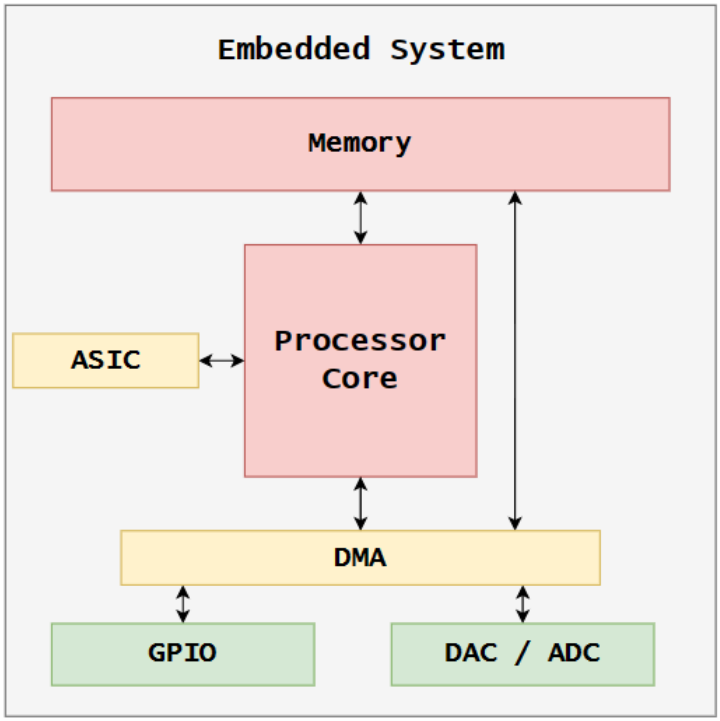


Figure 2.4: Typical embedded system (adapted from [10])

In the same work, he defined how a typical embedded system is, which is shown in the Figure 2.4. The colors depicted in the figure indicate the significance of each component in the embedded system. Red denotes an indispensable part that must be present. Yellow means that it is not mandatory, although it is recommended. Green stands for optional, as it can be used or not depending on the application. The arrows are used to demonstrate if the communications between the different parts are one-directional or bidirectional.

The processor core and memory are the two crucial components in an embedded system, as they are interdependent and cannot function without each other. Specifically, the processor requires the memory to access the data that needs to be processed, while the memory itself cannot perform any processing on its own. Therefore, the existence of both components is essential for the development of a functional embedded system. The Direct Access Memory (DMA) is a very important part of an embedded system. It is a hardware mechanism that allows peripheral components to transfer their Input/Output (I/O) data

directly to and from main memory without the need to involve the system processor. Thus, the significance of this lies in its ability to avoid a substantial processor overhead [11]. Application Specific Integrated Circuit (ASIC), as the name suggests, is used for specific applications where dedicated hardware is required. It can reduce significantly the execution time of certain applications [12] [13] [14]. For this reason, many systems require dedicated hardware however, it will become a smaller fraction as time passes [10]. General Purpose Input Output (GPIO), Anallog-to-Digital Converter (ADC), and Digital-to-Analog Converter (DAC) are examples of parts that establish an interface between the embedded system and the external world, and so they can be needed or not depending on the requirements.

The range of embedded systems is very wide. It starts from low power, like receiving data from ADC and sending it to a database, to multi-core applications, for example, laptops, smartphones, or even tables. Hence, when designing a system, it is important to comply with the SWaP-C constraints, which means that the system can only use the essential resources, commonly referred to as "resource-constrained devices" [15].

Furthermore, embedded systems should be dependable, that is, they should perform as and when required [16]. The following characteristics must be fulfilled for dependability, nevertheless, different applications can attend better to different topics. In the case of cars, trains, or planes, the focus should be more on safety, whereas databases or banks should prioritize security.

1. **Reliability:** Continuity of service delivery while in use, e.g., the probability of the system working properly since it worked after startup
2. **Maintainability:** Capability to be retained in, or restored to a state to perform as required, under given conditions of use and maintenance
3. **Safety:** Ability to not cause catastrophic effects on the environment as a consequence of a failure
4. **Availability:** Capacity to be in a state to perform as required
5. **Security:** Capability to provide communication confidentiality and authentication

2.2.2 Operating Systems

An embedded system can be complex. Take a laptop as an example. There are lots of different resources to manage, each one with its characteristics. Screen, mouse, keyboard, I/O devices, and network interfaces are some examples. Creating code to control all of them efficiently is almost impossible since the programmer needed to understand in detail every component. Operating Systems (OSs) came to solve this problem by providing a software layer that not only abstracts all the details for the user but can also optimally manage all the resources. The next picture presents how an embedded system with a OS can be divided. It is important to point out that every embedded system does not need to have an OS, for example, a device that only reads a ADC port and sends the information does not need that.

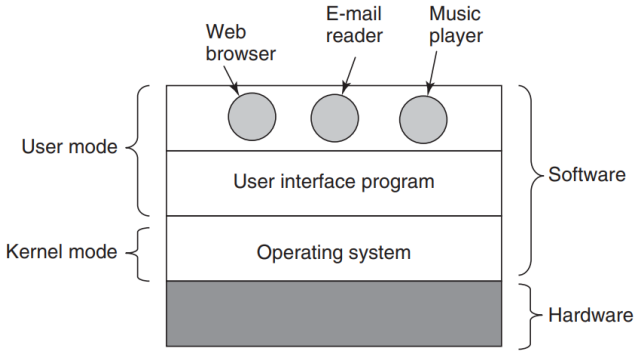


Figure 2.5: Layers of an OS [17]

Going into detail in the OSs, one important concept to keep in mind is the **process**, which is an instance of an executing program. Processes are crucial in a way that they allow the system to have multiple jobs at the same time. Following the previous example, a user with a laptop can play music and surf the internet at the same time. Even with one CPU, OSs have the capability of keeping track of multiple processes, by switching from process to process speedily, creating the illusion of parallelism. This concept is called **concurrency** and it is illustrated in the Figure 2.6

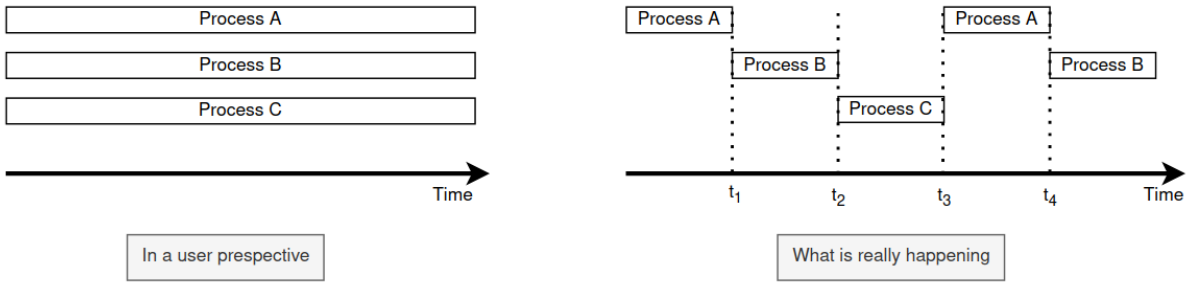


Figure 2.6: Concurrency between processes

Note that, in this example, it is considered a system with one CPU core. It is clear that if the system has more cores, which is common nowadays, the OS can split the work among them and have real parallelism. If the system has four CPU cores, each core can execute a process at a time, increasing the performance even more. To do this concurrency, the OS, more precisely the **scheduler**, manages the processes by attributing states to them. There are three states: running (the process is in execution at that time), ready (the process is not running at the moment but it is ready), and blocked (the process is unable to run until some external event happens).

When thinking about programs, it is clear that one can have multiple works to do at the same time, for instance, a web browser. It needs to display the content, receive the keyboard and mouse information, receive web packages, and so on. If it was only receiving the keyboard information, it wouldn't do the remaining jobs making the process inefficient. Therefore, processes can have multiple **tasks** that run within the process, which are called **threads**. Threads are essential for the following reasons [17].

1. They have the ability for the parallel entities to share address space and all of its data among themselves
2. They are lightweight, and easier/faster to create and destroy than processes (Around 10–100 times faster)
3. They can increase performance when there are substantial computing and I/O because it allows these activities to overlap
4. They are useful on systems with multiple CPUs, where real parallelism is possible

Processes are assigned with independent memory regions, and the threads within the process share the process memory between them. A process can have multiple threads, but a thread can have only one process associated. Without threads, a process contains one single stack and a set of registers. With threads, each of them has a stack and a set of registers. **Execution context** is represented with a combination of registers and stack, representing the CPU state for each task. Figure 2.7 illustrates the previous notion.

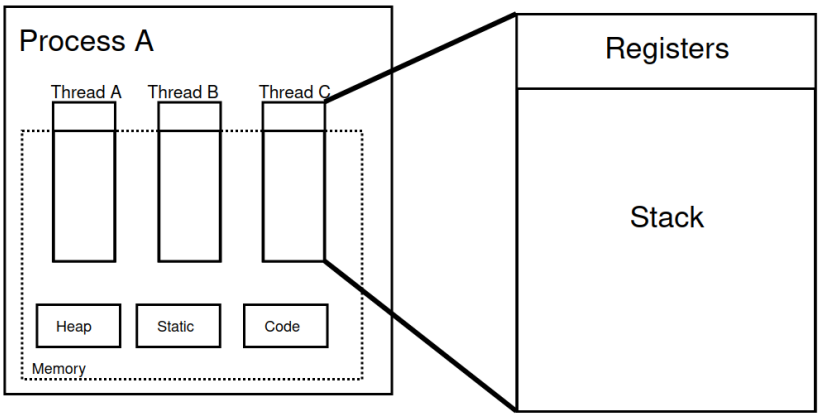


Figure 2.7: Process and threads block diagram

With threads can also be applied concurrency, in order to have parallelism and better computational performance. As processes, the scheduler defines which task will be executed. There are four states: run (the task is in execution at that time), ready (the task is not running at the moment but it's ready), wait (the task is waiting for data), and null (the task is created or killed). Figure 2.6 can also be applied to this context.

Nevertheless, in reality, it is not like that. When a task is running and another replaces it, there are some steps to pursue. First of all, it is necessary to save the CPU state of the running task. After that, the scheduler needs to select another to get running. Then, it restores the context of the selected one, so it can transfer the execution control. All this process is called **Context Switch (CS)**. The time consumption of the CSs are a problem, as present in the figure below, however the benefits of having concurrency, as explained earlier in this section, are massive, and thus the trade-off is positive.

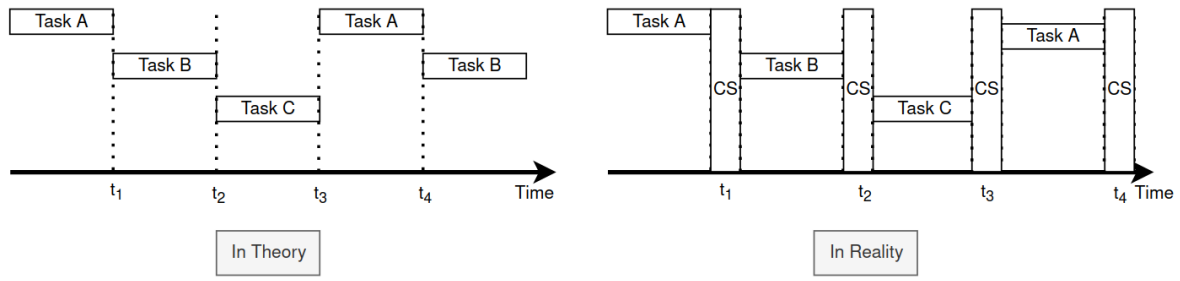


Figure 2.8: Context Switch (CS)

Another issue that can take plan is when two different processes are using the same memory. By definition, they are assigned with independent memory regions, but there are situations where it is not true. The most common is when two or more different processes interact with each other, where one, for example, writes in the memory and the others read from it. To control this **Inter Process Communication (IPC)**, two mechanisms were created: data transfer and shared data.

Data transfer involves the concepts of writing and reading. The most typical IPCs are pipes and message queues. The main difference between both is pipes are uni-directional communication channels, while message queues are bidirectional.

Shared data is a memory region that is shared by multiple processes. Thus, if a process wants to share some data, it only needs to make it available in the shared memory region. For this reason, it is the fastest IPC available because the data transfer occurs at the speed of memory access, but it is dangerous because it has the same meaning as global variables, so there is no control to access those. The next figure shows a possible problem that can happen with shared memory.

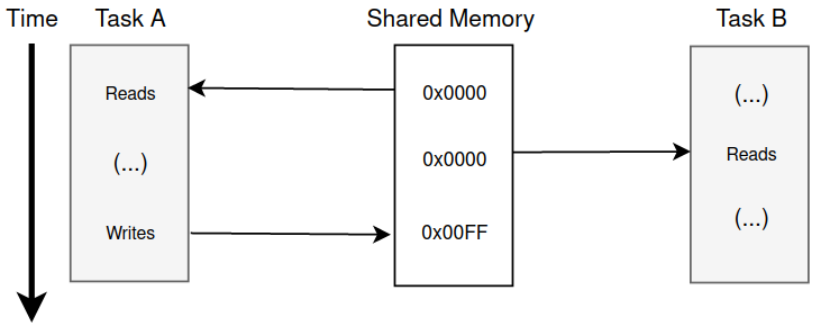


Figure 2.9: Two different tasks using the same resource

In this example, task A reads the value from the shared memory region, executes an equation, and writes the result in the same variable. For instance, this equation can be as simple as summing 0xFF (Y = X + 0xFF). Task B reads that value and prints it on the terminal. As demonstrated in the figure, task B will read 0x0000 even though task A was using the variable. In the case of multiple tasks and the result

of one matters to another, this issue can be a serious problem. This problem is called **race conditions**, and it is more critical with the increasing parallelism due to increasing numbers of cores.

To avoid that were created **the task synchronization objects**. The most important methods are semaphores and mutexes. The first one can be divided into binary semaphores and counter semaphores, and they are useful methods to synchronize interrupts with a given task. The second one is a shared variable that can be in one of two states: owned or free. Binary semaphores and mutexes share many characteristics. Both can be utilized for mutual exclusion, ensuring that only one thread accesses a shared resource at a time, but only the first can be used for synchronization [18]. Regarding the previous example, Figure 2.10 illustrates the same situation but using a mutex.

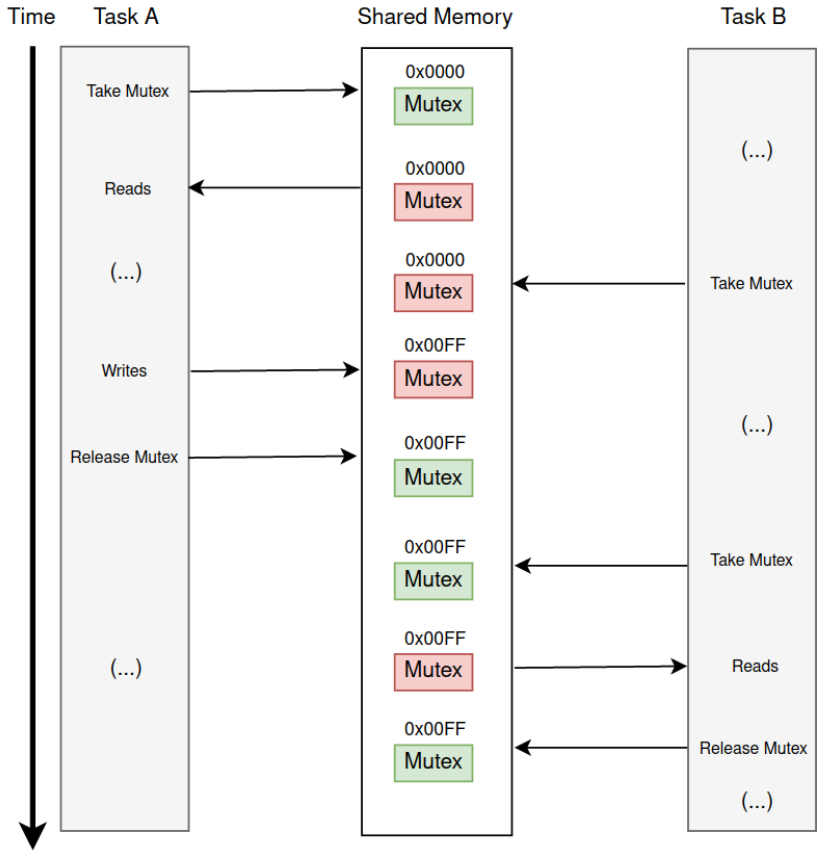


Figure 2.10: Two different tasks using the same resource with mutex

Now, both tasks, before using the shared variable, need to take the mutex. If the mutex was already taken (first trial of task B), the resource is blocked, meaning it is unavailable until the owner releases the mutex.

2.2.3 Development Models

When building an embedded system, several aspects should be taken into account, like its requirements, constraints, and levels of complexity, therefore one development model that works well for one

project may not work well for another with different requirements or constraints. This section will present some examples of models typically used in the industry and research teams. There are a lot more models with different characteristics, thus before the beginning of the project, all possibilities must be considered.

Waterfall model

The traditional waterfall model is linear, that is, there are well-defined stages and once the project moves to the next stage, it can not go back [19]. Figure 2.11 represents the different stages of the model.

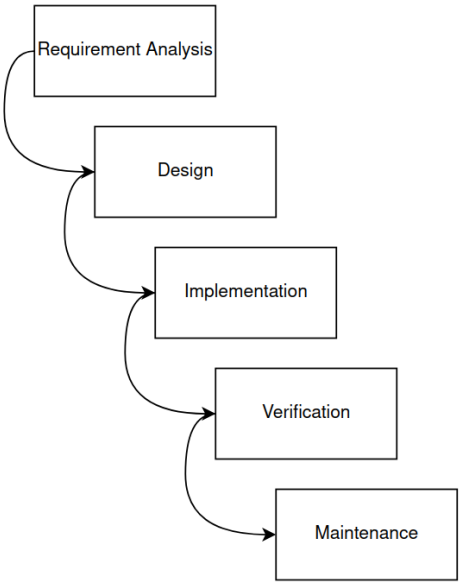


Figure 2.11: The waterfall model

Because of this linearity, no previous stages can be reviewed or improved, which makes the traditional waterfall model a good option for projects that have well-defined requirements and a short development cycle. Adetokunbo in [19] goes further and states that it is also viable to use it when altering the software after coding is very much prohibited.

Some variations allow to have an iterative relationship between phases and even add more phases. Royce in [20] presents and explains some of them in detail.

V-Model

The verification and validation model, more known as V-model, is a modified version of the Waterfall method [21]. It is non-linear, which means that allows step-backs in the development process. An important aspect of this model is that testing activities like planning, and test designing happen well before coding, preventing bugs or bad-functional systems. [22]. The following picture shows a typical representation of this model.

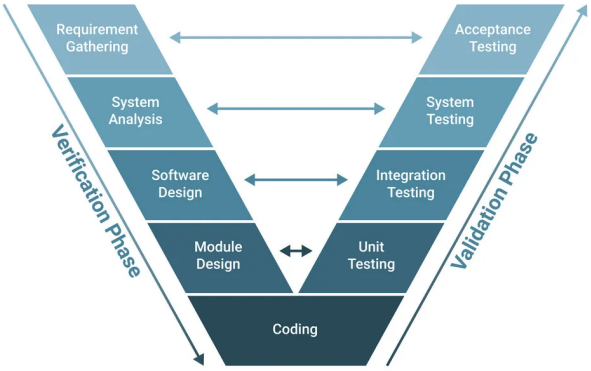


Figure 2.12: The V-Model

The main advantages are the proactive tracking of defects in the various phases, cost reduction in the correction of defects since these are spotted much sooner, and it is simple to understand and apply. On the other hand, it lacks flexibility as any changes require updating the majority of documents, demanding significant time and resources, which can make it challenging for companies to adopt. [21] [22]. Nevertheless, it is arguably the most traditional model utilized for software test management. [23].

Other variations try to improve these downsides. Some examples are the shark tooth and W-model [22].

Agile Model

There are lots of agile techniques, however, they share common characteristics, including iterative development and a focus on interaction, communication, and the reduction of resource-intensive intermediate artifacts [24]. Therefore, this model states that the project should be divided into mini-projects to remove unnecessary activities that waste time and effort. These mini-projects have requirements analysis, design, implementation, and test [24], and should not exceed 30 days [25]. In the end, all are combined to obtain the final project.

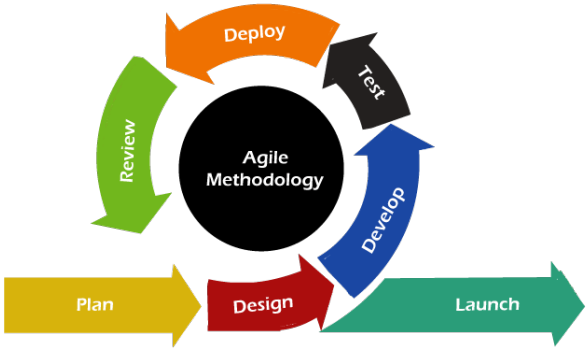


Figure 2.13: The agile model

Figure 2.13 shows the agile model. In the review stage happens an important aspect of this model which is the costumer interaction. The customer adaptively specifies his requirements for the next release based on the observation of the current release, rather than speculating at the start of the project [26].

Spiral Model

The spiral model is similar to the agile model, nevertheless, this one focuses on risk assessment and minimizing project risk. This can be achieved by breaking a project into smaller segments, which then provide more ease of change during the development process [27]. In Figure 2.14 is presented the spiral model. It is possible to notice that it requires a lot of time spent to evaluate the risks and planning or reevaluating the project every time a cycle is complete.

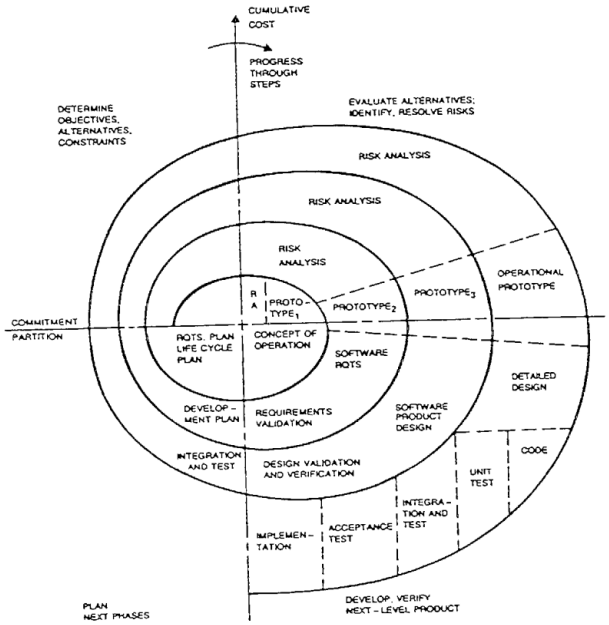


Figure 2.14: The spiral model [28]

Hence, this model is recommended to use for medium to high-risk projects, when risk evaluation and costs are important, and when significant changes are expected [27]. Moreover, it is also applicable to very large, complex, and ambitious projects [28]

2.2.4 Embedded Simulation

All the models presented in the subsection 2.2.3 have the verification stage, which is crucial to validate if the developed system meets the desired specifications. This validation normally requires the construction of a prototype in a way that engineers can evaluate its performance, test its functionality, and identify any potential issues or shortcomings. For example, in the automotive industry, where the most commonly

adopted model is the V-model [29], the construction of a mock-up is necessary to identify flaws, delaying the overall development process.

Therefore, simulation in the context of embedded systems is essential since the system can be tested without having the physical prototype, and some requirements, such as cache hits/misses, and computer performance, can be evaluated [3]. A simulator of this type is commonly referred to as a Full System Simulator (FSS) or a Virtual Platform (VP), and they can be described as a computer architecture simulator that simulates software in an electronic system, being this independent of the nature of the host computer. Usually, this term is mixed with emulation because it can also run software tests inside flexible, software-defined environments. Although, an emulator goes beyond by simulating both software and hardware configurations, being useful to test how software interacts with underlying hardware or a combination of hardware and software.

There are two types of FSSs, full system hardware simulator and full system software simulator. The hardware version offers a cycle-accurate simulation. As the name suggests, this technique is employed to perform a comprehensive analysis of the simulated embedded system at the clock level. Therefore, it is possible to accurately simulate hardware state transitions, obtaining specific information as the state of all logic gates or registers. Nevertheless, it can have very poor simulation performance because hardware-level simulators are not suited for the simulation of such complex systems as the embedded ones [30]. Figure 2.15 shows the simulation of a system using a hardware simulator. Some examples of this kind of simulator are GHDL [31] and Icarus Verilog [32].

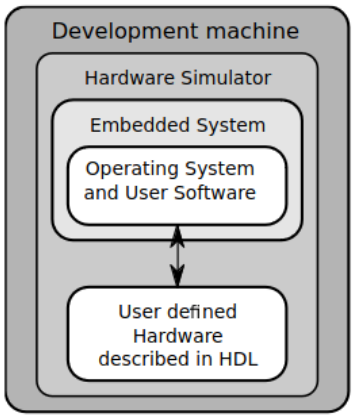


Figure 2.15: Full system hardware simulator [30]

The software version typically employs instruction-accurate simulation, where computations are performed according to the instruction set. However, this type of simulation does not provide information about the execution time of each instruction, resulting in a less detailed simulation that runs faster. Because of that, the hardware is not described in Hardware Description Language (HDL), unlike the previous version, as presented in the Figure 2.16.

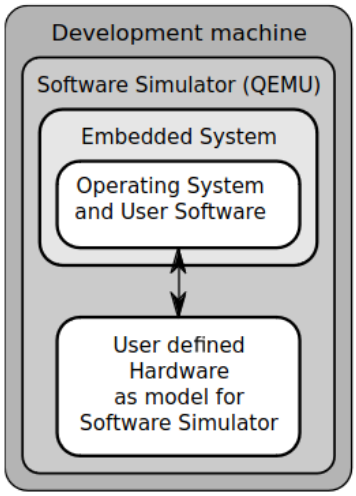


Figure 2.16: Full system software simulator [30]

The hardware model must be defined for the software simulation, thus hardware modules may or may not be available for the target machine. It requires the selection of a simulator that aligns with the application scenario or offers the flexibility to expand the range of supported hardware modules, such as QEMU [33] and Gem5 [34].

As modern embedded and integrated systems become increasingly complex, an escalating number of design companies are embracing virtual prototyping methods [35]. VPs should follow this trend, that is, as embedded systems are becoming more complex, VPs should be faster yet accurate, so they can be a reliable option.

The problem is VPs are not following this evolution, which results in low performance, and thus high simulation times [3] [35] [36]. There are several works that try to solve the problem, although none of them have been able to provide an effective solution.

2.3 Discrete Event Simulation

Simulations, while valuable, cannot guarantee 100% reliability due to certain scenarios that may be challenging to evaluate. For instance, certain physical phenomenon modeling may not be fully accurate when comparing with real-world behavior. However, to enhance reliability, an emulator should strive to accurately represent real-world conditions. Discrete Event Simulation (DES) aligns with these requirements by simulating system dynamics event by event and providing comprehensive performance reports.

A DES can be defined as a simulation technique where state changes (events) happen at discrete instances in time. Events take zero time to happen, and it is assumed that nothing happens between two consecutive events that is, no state change takes place in the system between the events. A group of events organized by execution order is called an event queue or process [37] [5]. From this point forward, whenever the term ”process” is mentioned, it specifically refers to the event queue.

Consider a supermarket as an illustrative example. A supermarket system consists of one employee and two cashiers, PAY1 and PAY2. In this context, three events can be specified, and they are related to each other as shown in the Figure 2.17.

- Costumer arrives at the supermarket (CA)
- Costumer goes to the cashier (PAY1 / PAY2)
- Costumer leaves the supermarket (CL)

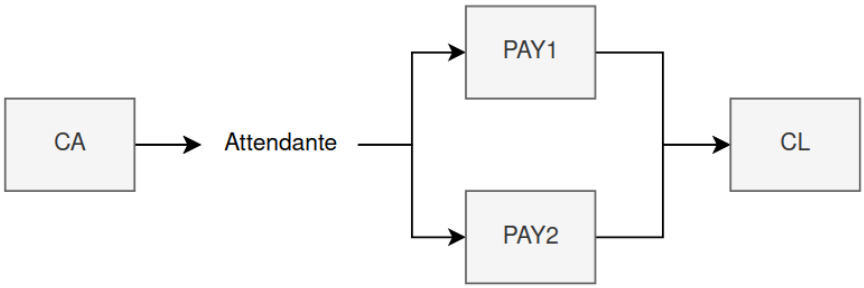


Figure 2.17: Supermarket flow schematic

Clients arrive randomly at the facility, and an attendant directs them to a cashier, accordingly to the following rules: Whether both cashiers are available, the customer goes to PAY1; If only one of them is free, the client goes there; If both are occupied, the buyer waits until one is ready; If another buyer wants to pay, a queue is created following a First-In First-Out (FIFO) style. The next table presents a situation assuming five clients and a payment time of five units of time.

Table 2.1: Example of a sequence of events in a supermarket

Event number	Time	Event description
1	1	CA1
2	1	CA1 goes to PAY1
3	3	CA2
4	3	CA2 goes to PAY2
5	4	CA3
6	5	CA4
7	6	CL1
8	6	CA3 goes to PAY1
9	7	CA5
10	8	CL2
11	8	CA4 goes to PAY2
12	11	CL3
13	11	CA5 goes to PAY1
14	13	CL4
15	16	CL5

When the time between arrival and reaching the attendant is considered to be zero, it indicates that there is no significant activity occurring. Otherwise, an additional event would have been included. In addition, when an event occurs, it is referred to as an event timestamp, typically measured in the same unit as the time within the model, known as the simulation time. Wall clock time or CPU time refers to how long the simulation program has been running and how much CPU time it has consumed on the target system. On the other hand, the the amount of time spent, known as host or real time, represents the actual duration taken to perform the simulation.

In this example, some events depend on others, for instance, event number eight is executed only when event seven has been executed as well. However, others are independent, like events one and two. So, even in this simple case, there are relationships between events that can complicate the concurrent execution of events, not allowing for a reduction in wall clock time.

This method has applications not only in the context of embedded systems but also in a wide area of topics. It is traditionally used for industrial applications, for example, in the design and evaluation of new manufacturing processes, and in the establishment of optimum operational policies [38].

2.3.1 Continuous Event Simulation

Besides DES, there is another simulation technique, which is the Continuous Event Simulation (CES). The perfect one does not exist, since they should be chosen to take into account the application, therefore

a brief explanation will be made to understand when one is better than another.

Continuous simulation is used for modeling systems with continuous or analog behaviors. The state changes are continuous since they are modeled by differential equations. [39] and [40] are examples of applications where the authors decided to use this technique.

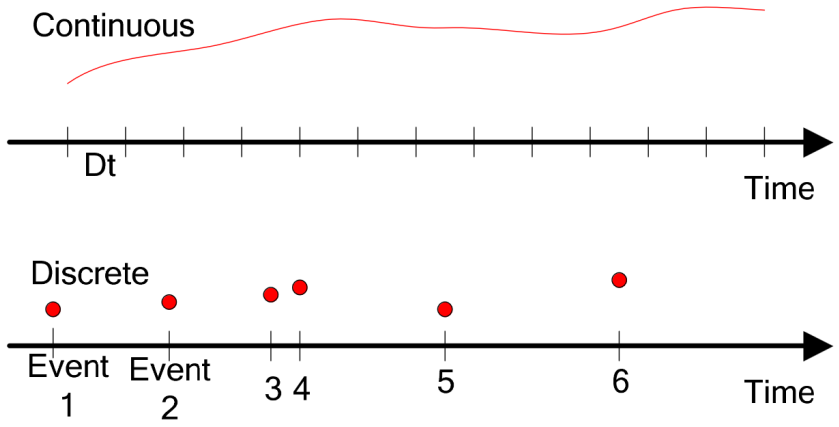


Figure 2.18: Updating state over simulated time in continuous and discrete simulation [41]

2.4 Simulation Modes

Different FSSs can use different modes to perform the simulation. These modes have an impact on the performance and accuracy of the simulation. For example, while Gem5 is limited to sequential simulation [34], QEMU offers the flexibility to be used in both sequential and parallel simulation [42]. Depending on the application and the requirements of the project one option can be better than another.

According to [43], there are two dominant approaches to advance time within a simulation, asynchronous and synchronous methods. In the first one, each thread communicates with the others in a way that each one can determine when it is secure to execute an event. The number of synchronizations can be reduced by far, although knowledge about the communication behavior of models and deadlock mechanisms, such as roll-back, is required. The second one uses global synchronization times to synchronize all threads. Between these times, all events that will be executed between them can be parallelized. This approach is simpler, nevertheless, it may incur high simulation overhead.

This section will be presented the two available modes, the sequential and the parallel mode. Nonetheless, here will be also explained the Temporal Decoupling (TD) technique, since both can use it in order to obtain more performance. Moreover, all further context will be related to the DES technique and the synchronous approach.

2.4.1 Sequential Simulation

Sequential simulation, or Sequential Discrete Event Simulation (SDES), is the most simple and accurate simulation mode. It executes the workload sequentially, that is, each event executes at its simulation time, resulting in a perfect simulation without any error.

Going into detail, the simulator runs the process corresponding or sensitive to an event that should be executed at a specific timestamp. This process will run until a certain time when another event must be executed, and it is not related to the process in execution. In this process exchange, there are a Context Switch (CS), where happens a synchronization with the rest of the system. All variables are updated, thus, when a process reads or writes a variable, it accesses the state of the variable as it would be at the current simulation time. The coming image shows a sequential simulation with two different processes.

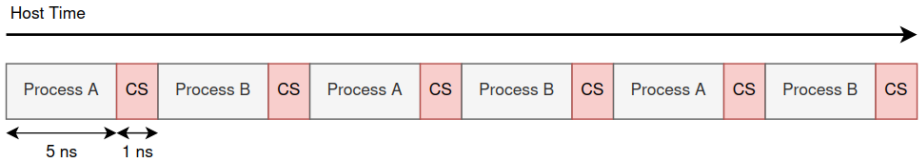


Figure 2.19: Example of a sequential simulation

Picture a scenario of a dual-core system with two processes where each event corresponds to an execution of an instruction with constant time (five nanoseconds), and each CS takes one nanosecond. If the emulation lasts for one min, ten seconds would be just for CSs. It is visible that it spends a lot of time simulation time, and it tends to get worst if more processes are needed. The overhead of event scheduling and process context switching becomes the dominant factor in simulation speed, leading to a huge host time.

2.4.2 Temporal Decoupling

A technique used by systemC, which is a standard C++ class library for system and hardware design, to improve the performance is Temporal Decoupling (TD) [44]. TD is a technique where individual processes are permitted to run ahead in a local time, without actually advancing simulation time, until they reach the point when they need to synchronize with the rest of the system. This time slice, from the beginning of the execution until the synchronization, is called **quantum**. The usage of TD may result in a faster simulation in some cases since it increases the data and code locality and reduces the scheduling overhead of the simulator. Figure 2.20 shows the application of TD to the previous example.

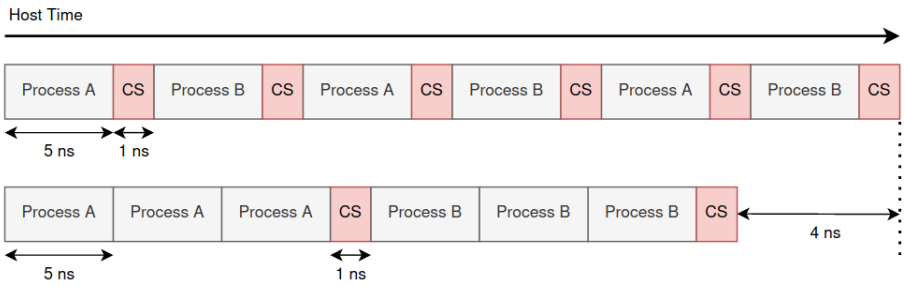


Figure 2.20: The principal of temporal decoupling

Now, for the same process execution time, the host time was smaller ensuring the existence of less CSs. In this example, the reduction was four nanoseconds (four CSs), circa 11% of host time, compared to the approach without TD. It can be even higher if more processes and CSs were being simulated.

As mentioned earlier, the process is permitted to advance beyond the current simulation time until it requires interaction with another process. This interaction could involve reading or updating a variable that belongs to another. At the moment, two things can happen: either access the current value and proceed, sacrificing accuracy, or request synchronization and continue its process when the simulation time aligns with its local time. Proceeding with the current value entails making assumptions about communication and timing within the modeled system. It assumes there will be no adverse consequences when sampling or updating the value either too early or too late. Typically, this assumption holds in the context of a virtual platform simulation, where the software stack is designed to be independent of the intricate timing intricacies of the underlying hardware.

Each process is responsible for evaluating whether it can progress beyond the current simulation time without compromising the functionality of the model. This is a systemC characteristic because it guarantees functional congruency with the standard semantics of the simulator, thus, it is not a mandatory feature to implement in other situations.

Nevertheless, a problem can occur if the process does not respect the previous rule and run with no restriction. Other processes will not be able the run, compromising the system's functionality. A solution is to define a global quantum that forces the synchronizations. This global quantum, in the original version of systemC, is static, which means it is defined by the user before the simulation and never changes. Forcing the synchronization results in another problem which is the trade-off between speed and accuracy. A small global quantum guarantees that the processes are using always the updated values and the simulator does not crash, therefore the accuracy will be high. However, the simulation speed will be sacrificed as more CSs are occurring. On the opposite side, if the time slice is big, means that the system might introduce timing inconsistencies, which can lead, in the worst case, to a crash in the simulator. Hence, this value must be chosen carefully in order to have a fast yet accurate simulation.

The systemC reference manual [44] enforces that some processes cannot be temporally decoupled due to their characteristics, thus they might become a simulation speed bottleneck.

From here onwards, whenever the word "quantum" is mentioned, it refers to the global quantum.

2.4.3 Parallel Simulation

In the parallel mode, or in a PDES, as the name suggests, the simulation uses more than one simulation thread in order to have real parallelism. In consequence, the host time will be smaller compared to the sequential mode, since multiple processes can be running at the same time. The next figure shows the speed difference between the two modes.

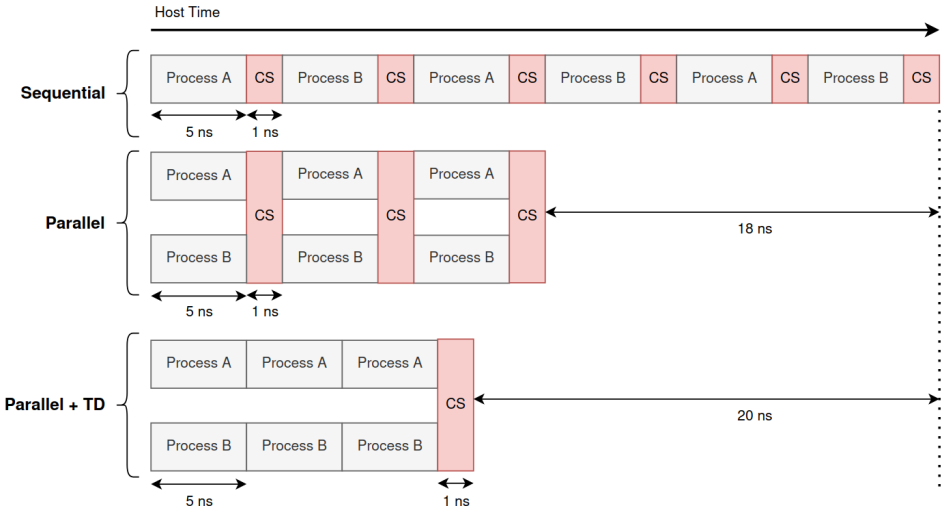


Figure 2.21: Sequential VS parallel simulation with and without TD

Continuing the previous scenario, with the parallel mode, the simulation can be fully optimized in a way that each process can have its execution thread. For this reason, the performance of the simulation increases a lot. In this case, for the same number of executed processes, the parallelized version was able to reduce eighteen nanoseconds of host time, which means a reduction of 50%. With TD, the gain was small compared to the approach without it, however, if more processes were used, the gain would be much higher. But it does not only offer advantages. Several challenges make achieving PDES difficult. Typical problems are load imbalance, limited parallel work, and causality errors [45] [46].

When working with tasks there is always the problem of balancing the workload among them to make an efficient use of today's multicore computers. An inefficient distribution can create performance bottlenecks. For instance, a system is programmed to read two different GPIO ports, do a logical XOR between both, and write in another GPIO port the result (to turn on a Light-Emitting Diode (LED)) and send it to the terminal. This workload results in four events, which will be assigned to two processes. If process A is attached with only one event, e.g. sending the information to the terminal, process B will be overloaded, and vice-versa. This issue is easy to solve in this simple example, but in programs with thousands Lines of Code (LoC), it is a challenging job. [47] and [48] are examples of works that help to detect and measure the work imbalance.

It is rational to think that, after observing Figure 2.21, more parallelism will provide an even faster simulation, but it is not true. Normally programs can be parallelized until some point, like this example, it is possible to simulate this system with four cores, however, it is clear the extra two cores will not be executing anything, because there are no more processes to run. For this reason, the performance will not be improved. It can happen the opposite, that is, it being even worst [49]. Some works identify this scalability problem to show the programmer how it can be improved [50] [51].

The last problem arises when there are causality errors. Causality errors happen when one event in the future affects an event in the past. Sequential simulation ensures that event B will execute after event A if the timestamp of event A is smaller than the timestamp of event B. In this mode that cannot happen, and if event B can change the state variables used by event A, a causality error happens. The following image illustrates the previous scenario, with events colored in green denoting "in execution," and events in gray indicating "scheduled".

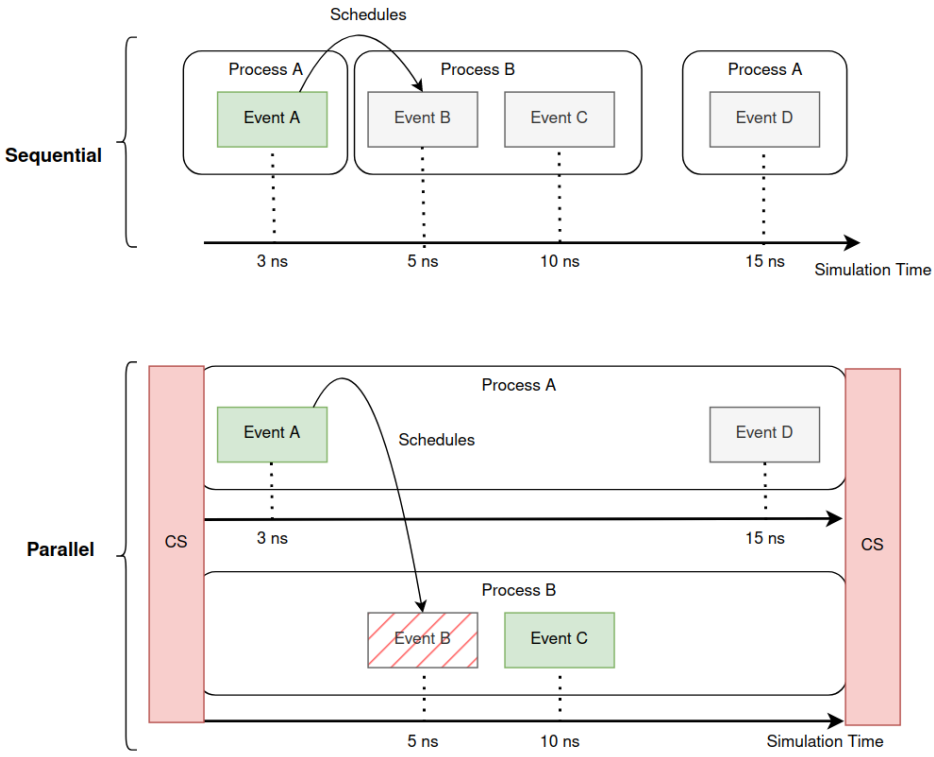


Figure 2.22: The causality problem

When the simulation starts, event A is executed. Meanwhile, it triggers event B, which has an event timestamp of five nanoseconds. In the sequential mode, since it executes one event at a time, this is not a problem. In the parallel version, it can be, because each execution thread has its own time. Can be that one thread is ahead of another, e.g. thread 1, which is executing process A, is at three nanoseconds, and thread 2, which is executing process B, is at ten nanoseconds, and when event A schedules event B to five nanoseconds, this event will not be executed for the reason that a DES cannot execute events in

the past. On top of that, as mentioned before, if this event B modifies the state variables of event C, the problem arises.

The usage of TD in both modes, may increase the occurrence of causality errors. Not only that, it can create this type of error in another way. Take the coming scenario as an example. A simulator is using the parallel mode to test a system that is writing values into a DAC to play a song in an analog speaker, and, at the same time, solve complex mathematical equations to perform that audio. Supposing the wanted output frequency was the gold standard audio (44.1 kHz) [52], the associated event needed to update the GPIO port every 2.27 microseconds. If the defined quantum is higher than that value, the event will not be executed at this event timestamp. Hence, the causality error was caused by the chosen quantum time. In this particular example, the consequence would be the production of audio that differs from the expected output. Although, in other cases, these errors can break the system functionality and crash the simulator.

As peer with Fujimoto in [53], there are two mechanisms to solve the problem: optimistic and conservative methods. In the first one, causality errors are not prevented at all, although they use detection and recovery strategies to correct that. The correction can be done with a rolling back method, which returns the simulation to a point before the causality error and sets the tighter synchronization. A problem with this method is the performance cost, because of this rewind in the simulation. The second option, on the other hand, avoids the possibility of any occurrence of these errors. An evaluation is done on the event, thus can be identified if the event is safe to process, that is, it is ensured that all events that should influence the given event have been processed before its execution. Some works with optimist approaches are [54] and [55]. For [56] and [57] are used conservative methods.

2.5 Gem5

One available FSS used in academia and industry is the Gem5 simulator [34][58]. It has its roots in 2011, from the merge between M5 [59], a CPU simulation framework, and the multifacet General Execution-driven Multiprocessor Simulator (GEMS) toolset [60]. With this combination, it is possible to use the best of both frameworks, the effective support of multiple Instruction Set Architectures (ISAs) and the cache memory and cache coherence models. It is also the result of the work between AMD, ARM, HP, MIPS, Princeton, MIT, the Universities of Michigan, Texas, Wisconsin, and many other institutions.

In addition, Gem5 came to overcome a demand in the market at the time. The main highlights are the flexible framework for researchers to evaluate their design in several different ways; Licensing terms that allow researchers and academia to work together without the pressure of revealing proprietary information, in the industry case, or not getting credits for their contributions; Defined code style that ensures the code quality remains good to new collaborators understand faster and better the code [34].

This section will explain in detail what are its capabilities and how can it be used. Then will be presented a work done by the ICE team of the RWTH Aachen University, which will be the reference work for this

dissertation. To conclude, other simulators will be given to have a wide perspective of what exists in the market and when and where one can be better than another.

2.5.1 Overview

Gem5 is a DES platform that has as its main goal to be a community tool focused on architectural modeling. It has a set of CPU models and memory systems, and two different system modes, the System-call Emulation (SE) and Full-System (FS). Figure 2.23 represents the different simulation configurations regarding the trade-off between speed and accuracy.

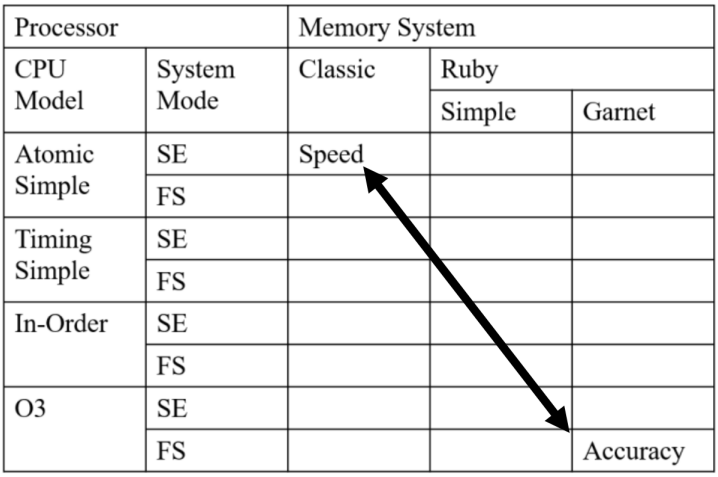


Figure 2.23: Speed vs. accuracy spectrum [34]

As mentioned in the introduction of this section, one characteristic of Gem5 is flexibility. Several simulation configurations allow an adaptation to fit the requirements of the project or the level of wanted detail. A project where scalability is a must-have feature, for example, will not require a detailed CPU model. The CPU models will be explained with more detail in subsection 2.5.2.

In this simulator, components can be rearranged, parameterized, extended, or replaced easily to suit the project's needs. This is possible thanks to its pervasive object-oriented design, with the mix of Python and C++. All major components, like the memory, are SimObjects and share common behaviors for configuration, initialization, statistics, and serialization. Moreover, a user can create a customize SimObject, increasing even more the flexibility. To do that, it is required to define in a Python file the SimObject's parameters, such as instantiation, naming, etc., and in the C++ files define its behavior.

Another remarkable characteristic is the support of different ISAs, counting with Alpha, ARM, SPARC, MIPS, POWER, x86, and recently RISC-V. Currently, not all possible combinations of ISAs and other components are known to be functional, as demonstrated in the Table 2.2

Table 2.2: Overview of the supported architectures in Gem5 [61]

ISA	Level of ISA support	Full-system OS support
Alpha	High	Linux
ARM	High	Linux, BSD, Android
MIPS	Low	None
Power	Low	None
RISC-V	Medium	None
SPARC	Low	None
x86	Medium	Linux, BSD

Furthermore, it is also possible to simulate more than one computer system in various ways. It is done by instantiating another system and connecting it via a network interface, creating a client/server pair that communicates with the TPC/IP protocol. The results of the simulation keep deterministic, in other words, for a particular input, it will produce always the same output.

Gem5 uses other tools to perform specific jobs. The most important ones are the pybind11 [62], a lightweight header-only library that exposes C++ types in Python and vice versa, and SCons [63], an open-source software construction tool that orchestrates the construction of software in a way that solves several problems compared to the other build tools. With the SCons scripts present in Gem5 is possible to build different binaries for different purposes.

- **Debug:** This mode has no optimizations, and thus it is the slowest one. It is mostly used when the opt version does not provide enough detail in the debug session.
- **Opt:** It has most of the available optimizations but still with same debug information. Compared to the debug version, it is much faster.
- **Fast:** The fastest binary available. All optimizations are on, and there are no debug symbols. In addition, asserts are removed, although panics and fatals are still included. This binary should only be used when it is desirable for maximum performance, and the code is very unlikely to have bugs.

2.5.2 Simulation Capabilities

As shown in the Figure 2.23, this simulator provides different features that should be used regarding the project requirements. Those are the support for different ISAs, CPU models, and execution modes. Also, it has the capability of modeling multiple systems at the same time, supports multiple devices, two different network models, and different cache coherence protocols [34].

While the event queue is responsible to execute its associated events, the CPU is answerable to schedule the events, thereby, it is possible to keep running the simulation even if no CPUs are working.

There are four types of CPU models. Atomic simple, timing simple, in-order, and out-of-order (O3). The atomic and timing models are the simplest ones. The main difference between them can be seen in the figures below. Meanwhile the first completes all memory accesses immediately, which makes it a proper option for tasks like fast-forwarding (a technique used to warm up micro-architectural state), the timing mode models the timing of memory accesses, providing a more accurate simulation.

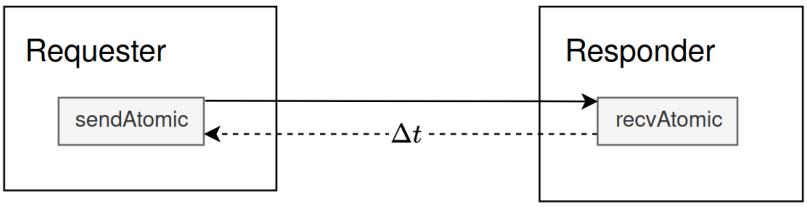


Figure 2.24: Atomic mode

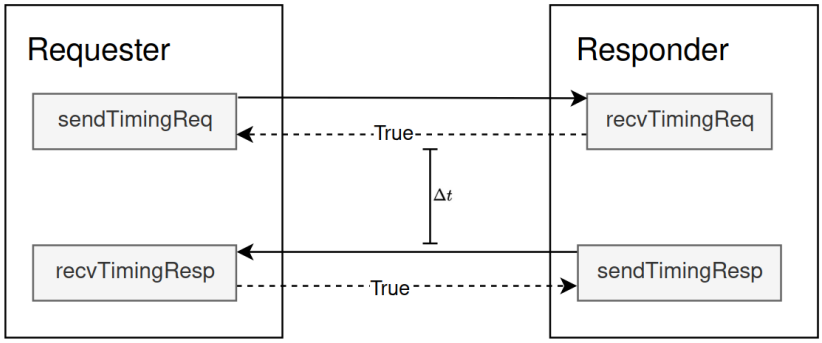


Figure 2.25: Timing mode

The in-order and O3 models are "execute-in-execute" models, which means that instructions are executed only in the execute stage once all dependencies have been resolved. For this reason, those are highly accurate.

Gem5 can operate either in FS and SE mode. The FS is more complex and complete, in a way that it supports interrupts, exceptions, I/O devices, and so on. As the name suggests, the SE simulates the system calls, like write(). This characteristic can be a bottleneck to multi-thread applications, nevertheless results in a faster simulation. If the workload has lots of I/O iterations or OS services, the FS is the proper choice, otherwise the SE can be considered, even because some ISAs do not support the FS mode yet.

Another important capability is the support to the classic and ruby memory system, due to the integration of both works (M5 and GEMS). On one hand, the classic system is fast and easy to configure. On the other hand, the ruby system is more flexible and accurate. Gem5 also offers an extension for the simple version named Garnet. At the time, Gem5 supports HeteroGarnet or Garnet 3.0 which brings improvements compared to the previous version, for example, it can enable accurate simulation of emerging interconnect systems.

Even though Gem5 is a very flexible simulator, accordingly to [34], there are other capabilities the developers want to include. A first-class power model, full cross-product ISA/ CPU/memory system support, and parallelization are some examples. It is evident that there is still much work to be done, but compared to the first review, there are lots of advances and improvements, like the support for the RISC-V architecture and the Kernel Virtual Mode (KVM) [58].

2.5.3 Usage

To start using Gem5 it is necessary two things. First of all, the simulator's binary for the wanted ISA is necessary. As mentioned before, there are three different types of binaries, therefore they should be chosen mindfully. Then, it is mandatory the definition of the system. It describes which parts the System-on-a-Chip (SoC) consists of and how they are connected. Gem5 provides in its tutorials the simplest system a developer can use, and it is presented in the following picture.

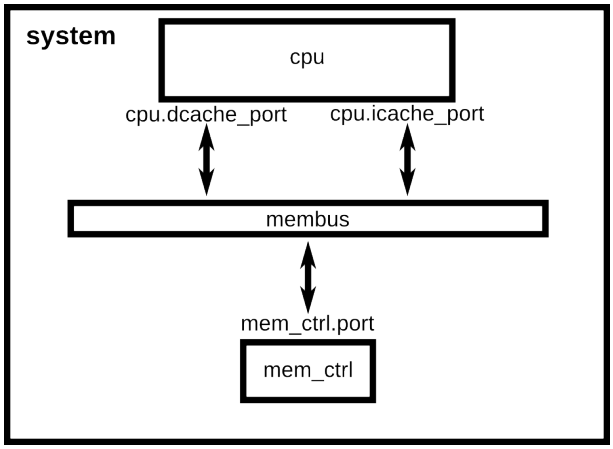


Figure 2.26: Simple system configuration diagram

All the configuration is performed in a Python script, while the functionality of the simulator is implemented in C++. Making changes to the latter requires creating a new binary, whereas in the former case, it does not. The next code shows what the Python script should have to instantiate and execute the simulation.

```
#create the system to simulate
system = MySystem ( opts )

#set up the root SimObject and start the simulation
root = Root(full_system = False, system = system)
m5.instantiate()

#instantiate all objects above
print("Beginning simulation!")
exit_event = m5.simulate()
```

```
11
12    #After execution
13    print('Exiting @ tick {} because {}'
14      .format(m5.curTick(), exit_event.getCause()))
```

Code 2.1: Script to instantiate and execute the simulation

Finally, to run Gem5, the command should first have the directory of the binary, and then the directory of the script. Furthermore, it can have more arguments to specify the system or the simulation modes, e.g. the CPU model.

2.5.4 Par-gem5

A huge problem of Gem5 is its speed. At the moment, its official version only offers a single-threaded simulation. Even with the best CPU or the fastest memory on the host machine, the simulation can not get close to one Millions-of-Instructions-Per-Second (MIPS), which results in long execution times. For instance, while the SPEC2017 integer benchmark can take ten minutes in a host computer, with Gem5 the same workload can take more than two years [3]. As referred in the subsection 2.5.2, parallelize Gem5 is one desired feature to have, however, more than ten years passed and there are no developments on this topic, only the support for KVM.

Par-gem5 [3] arises to solve this problem. It was developed by the ICE team of the RWTH Aachen University, with the collaboration of Huawei. It utilizes the multi-threading capabilities of the host system through a modified conservative, synchronous PDES approach, which allows the execution to be dispatch to multiple simulation threads that to run independently from the rest of the system for a time $t_{\Delta q}$ – the so-called quantum or quanta. At the time, only dist-gem5 [56], a work that focuses on simulating distributed systems connected via a Network Interface Controller (NIC), has implemented a parallel extension. Nonetheless, its application is very strict, since can only be used in this type of system, thus par-gem5 comes to overlap this problem and be a generalized parallel version.

In the initialization phase, each CPU is assigned to a dedicated event queue, and all other objects are assigned to the default event queue (q0). Thereby, the total number of threads will be the number of cores plus one. This version cannot be classified either as conservative or optimistic for the reason that it allows causality errors to occur.

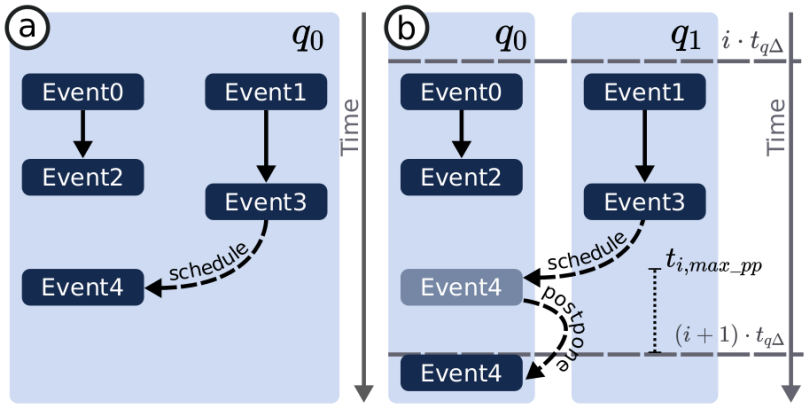


Figure 2.27: Example of scheduling events in Gem5 [3]

Scenario A represents the typical sequential simulation, and scenario B the par-gem5 scheduling method. When an event is scheduled to another CPU, it can be either executed or postponed to the following synchronization. The event is postponed when its timestamp has passed. Consequently, it can lead to a chronologically incorrect execution order of events, affecting the simulation's accuracy, or in the worst case, flaws the functionality of the simulated system. Nevertheless, in [3] is proven that the inaccuracy can be kept within a single-digit percentage.

The results of this work were very positive. For example, a speedup of 24.7x was achievable without losing accuracy significantly. Moreover, it has been demonstrated that there exists a saturation point in the definition of the quantum. Based on their tests, this point falls between ten microseconds and one millisecond, indicating that increasing the quantum beyond this range does not result in performance gains. Another important conclusion was the relationship between cores and inaccuracy. If the quantum does not change, when the number of cores increases, the inaccuracy also increases.

One way to solve the problem is to set a smaller quantum, but getting the perfect quantum is hard. One can be perfect for one benchmark and bad for another simultaneously. Zurstraßen et al. [64] goes further and performs a study where this trade off is evaluated. In the end, it was concluded the speedup of the simulation as a function of the quantum behaves similar to a sigmoidal or "S"-shaped curve, while the simulation inaccuracy, on the other hand, grows linearly. Moreover, for the cases studied, the 95% of the maximum attainable speedup was obtained at a quantum between 1000 and 10000 instructions, as different quantums suit better for different workloads.

Finding a quantum that allows high accuracy with high speedup is one of the main challenges when running a PDES. There are some works that try to explore this problem and come up with solutions, but none of them at the moment, as shown in the table below, is generalized, real-time, and for PDES simultaneously.

Table 2.3: Overview of the different works regarding the quantum definition

Work	Real-Time	Supports PDES	Generallized
Jung et al. [55] (2019)	X		X
Glaser et al. [65] (2015)	X		X
Jünger et al. [36] (2021)		X	X
dist-gem5 [56]	X	X	

The first work uses a technique that tries to achieve 100% accuracy by rectifying causal errors when they happen, forcing the system to simulate again the faulty part with a smaller quantum. This approach is named as rollback mechanism. One of the best-known methods of this type is the “Time Warp algorithm” [66]. Despite this concept is not new, it is still very present nowadays [54]. Beyond the non- PDES support, a criticism of this method is the performance cost. Because it ”rolls back” the simulation every time a causality error happens, the execution time will be much higher.

The second work uses a wiener filter to update the quantum within the simulation. This work was able to obtain great performance since it does not require lots of computational resources. Here, there are two simulators, systemC, and Verilog, that communicate between them by a ADC. Hence, the quantum that leads to high accuracy should be equal to the smaller latency. To get that, it assumes a stationary process and model order is known, which sometimes is not possible. Moreover, this work is done in a context of a single-threaded simulation, so there is no IPCs being evaluated.

In the context of PDES, Jünger developed a method whose principle is to classify events as relevant or irrelevant. An event is considered irrelevant to others if its synchronization is not related to them, for example, a timer interrupt. With this information, each local quantum is adapted accordingly to the following event. In consequence of that, it is possible to obtain perfect accuracy, without performance costs. To do this classification, firstly the simulation must be run, and then with the results, classify the events. This turns the technique not so desirable since the user needs to execute twice the same workload.

2.5.5 Other Simulators

Like Gem5 other simulators are used for the same purpose. This subsection will discuss some of them, their advantages and disadvantages compared to the simulator under study, to have a better perspective of what is the offer about this subject at the moment.

Starting with QEMU (Quick EMUlator) [33], as the name suggests, it is a versatile full-system emulator that can emulate various architectures, including x86, ARM, PowerPC, and more. As explained earlier in the subsection 2.2.4, there is a difference between an emulator and a simulator, yet QEMU was considered due to other similarities, like the open source availability, the support to different architectures, and the active development community. José Morales in [67] compared these two simulators in detail and, in the end, the conclusion of his study com be summed up in the next figure.

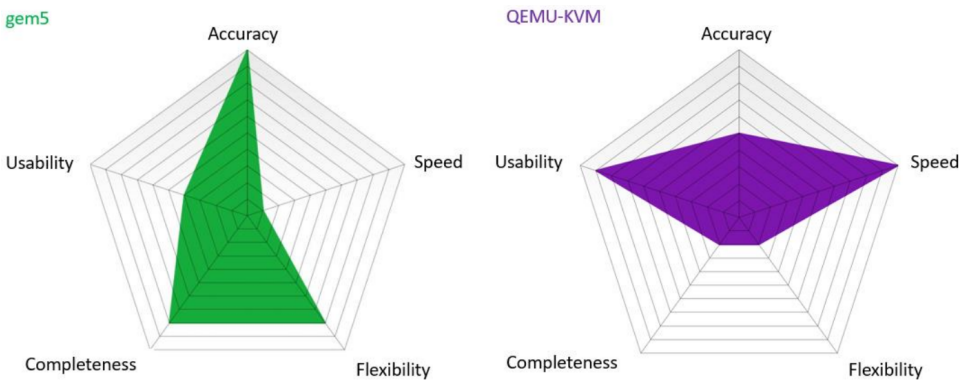


Figure 2.28: Final assessment charts [67]

One characteristic of QEMU is its loosely timed coding style, which provides high simulation speed environments. Cycle-accurate coding style, as Register-Transfer Level (RTL) simulators, on the other hand, cover fine-level Intellectual Property (IP) tuning. This is useful to validate hardware design. Gem5 can be positioned within the spectrum as an intermediate solution, offering a balance between performance and power exploration for early system-level solutions.

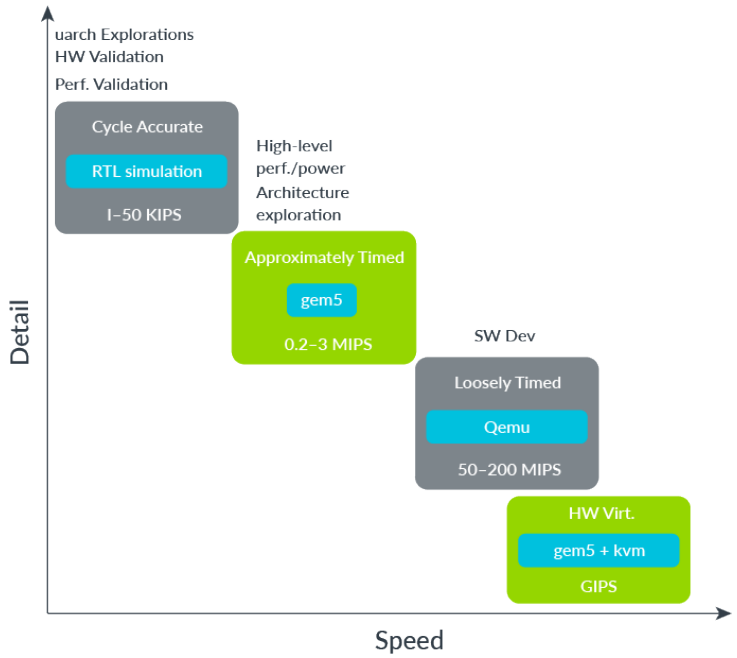


Figure 2.29: Modelling solutions spectrum [68]

Another platform already mentioned is SystemC [44]. It is a set of C++ classes and macros which provide an event-driven simulation interface for system and hardware design. Its main purpose is to provide a C++-based standard for designers and architects who need to address complex systems that are a hybrid between hardware and software. Furthermore, it can be used also as an HDL, still, it may require an evaluation due to its syntactical overhead compared to other options like Verilog or VHDL. It also supports Transaction-Level Modeling (TLM)-2.0, which allows us to model the communication as

function calls. Events are applied to an entire transaction payload, rather than to individual bus signals, and at protocol phase boundaries, rather than at clock edges [69]. Thus, the simulations have much better performance, 100-10000 times faster, when compared to the cycle-accurate version. The main advantages are its standardization in the industry, lots of documentation and work, and the capability to quickly simulate hardware and software systems on different levels of abstraction. Contrarily, it is not so flexible, hence, it requires external modules to complement itself, e.g., it can not perform cycle-accurate simulations, and most of the modules/ IPs are not free of charge, meaning extra costs.

Ayaz Akram and Lina Sawalha developed a work where a comparison of x86 computer architecture simulators is done [70]. In this study are evaluated four simulators: Gem5 [34], Multi2Sim [71], Sniper [72], and PTLsim [73]. The authors selected these simulators based on their diverse design approaches regarding the level of detail and abstraction. All of them are modern simulators that are actively being developed, except for PTLsim, which is currently not undergoing active development but is still widely utilized. The following image presents a features comparison between the previous simulators and ZSim [74].

Feature	Gem5	Sniper	PTLsim	Multi2Sim	ZSim
Platform support	P++	P	P	P+	P
Target support	T++	T	T	T+	T
Full system	✓	X	✓	X	X
Fast forwarding & cache warmup	✓	✓	X	✓	✓
Checkpointing	✓	X	X	✓	X
Trace generation	✓	✓	✓	✓	✓
Details of generated performance stats.	D++	D	D+	D+	D+
Pipeline depth configuration	✓	X	✓	X	✓
Energy and power modeling	E++	E	E	E-	E
In-order pipeline support	✓	✓	X	X	✓
HMP support	M,G,S	S	X	M,G	S
GPU-Modelling	✓	X	X	✓	X
Multi-threaded app. support	✓	✓	✓	✓	✓
Community support	C++	C++	C-	C	C+
Note: [feature's 1st letter]++ is better than [Feature's 1st letter]+ which is better than [feature's 1st letter] which is better than [Feature's 1st letter]- , S=Single-ISA, M=Multi-ISA, G=GPU					

Figure 2.30: Feature comparison between simulators [70]

In their work, these platforms were tested with three different benchmarks. In the end, was possible to conclude that Gem5 is a good option when very detailed results and multiple ISAs support are wanted. Sniper is specifically designed for multi-core simulations and is considered the most accurate among the simulators examined. However, it lacks the capability to generate detailed performance statistics for the simulated system and is comparatively less flexible than the other simulators. Then, if the focus is a CPU-GPU architecture simulation, Multi2Sim proves to be a great preference. Finally, PTLsim did not get a best-case scenario to be used, nevertheless, it is the base of other x86 diverse simulators, such as the MARSSx86 simulator [75].

2.6 Machine Learning

Machine Learning (ML) is at present-day an extremely important subject, encompassing applications in both smaller devices like watches and larger ones such as cars. Autonomous driving is one example of what can be done with machine learning [76]. Although this theme emerge recently, with the evolution of computers, in 1959 it was first defined by Arthur Samuel, who says that it is a field of computer science that gives computers the ability to learn without being explicitly programmed [77].

There are various types of ML algorithms, such as classification, error cancellation, and prediction. Depending on the project requirements, these algorithms can be utilized individually or combined in a manner that complements each other to address complex problems, like [76]. The following picture shows some of the commonly used algorithms in ML.

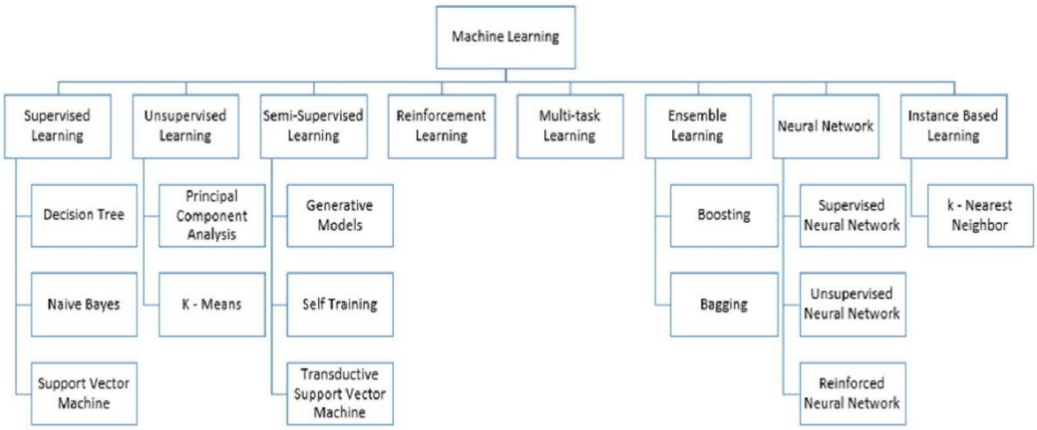


Figure 2.31: Some ML algorithms [78]

Within this dissertation, only neural networks will be considered. They can be divided into Artificial Neural Networks (ANNs) and Biological Neural Networks (BNNs). A Biological Neural Network (BNN) is a network of neurons that are connected by axons and dendrites, and the connections between neurons are made by synapses. A Artificial Neural Network (ANN) will be explained in detail in the next subsection. Also, it will be presented how ML can be applied to simulation and the improvements it brought.

2.6.1 Artificial Neural Networks

ANNs are types of ML that are inspired by the operation of the human brain. They try to model it to implement a particular task or function of interest. To do that, the biological neuron was studied in detail to understand how it works. How the information flows from one to another, how it adjusts the importance of several inputs, and so on. Figure 2.32 and Table 2.4 represent how a biological neuron can be transformed into an artificial one.

Table 2.4: Comparison table between a biological and an artificial neuron

BNN	ANN
Dendrites	Inputs
Soma / Body Cell	Summing and Activation Junction
Synapses	Weights or Interconnections
Axon	Output

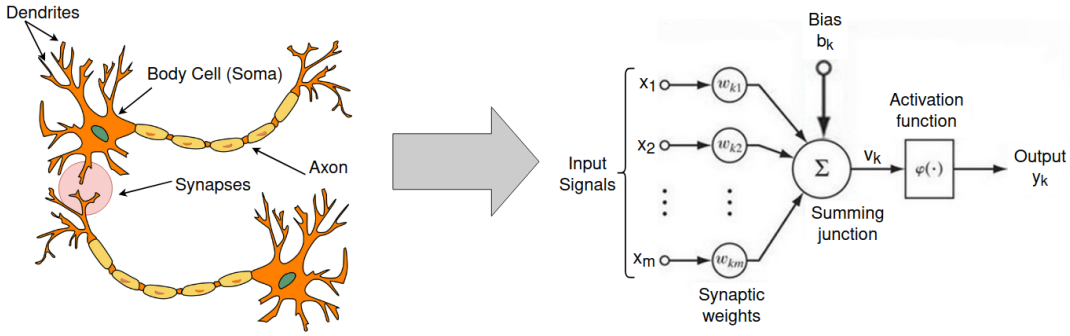


Figure 2.32: Comparison diagram between a biological and an artificial neuron

Like the human brain, ANNs also needs to learn. There are three ways to train a ANN, which are also inspired by human actions. Supervised learning is a method that teaches the Neural Network (NN) by providing the outputs for certain inputs. The predicted outputs are compared to the real ones, and then weights are adjusted according to the error. In the real world, it can be compared to heating food. The human knows how warm wants his food, but he cannot calculate how much time should it be warming in the microwave. Thus, it predicts a time, and based on what result, he redefines the time.

In the opposite way, unsupervised learning is a technique where the NN only knows the inputs. The training is done only with the input information, which is a good solution for clustering or density problems, where the data is categorized based on similarities. Humans face problems like this as well. For instance, when it is required for a person to sort apples, the color can be a characteristic to distinguish them.

The last method is the reinforced NN. There is an agent that controls the outputs of the system. The NN may receive a reward or a penalty for each action it performs. With this information, it adapts its behavior to receive the fewest penalties possible. A comparison can be made to a relationship between a mother and her son, where the mother supervises the actions of her son, and reward or alert him when he does something.

Also, ANNs can be classified into two different classes. The Feed-Forward NNs and the Recurrent NNs. In the first one, the information pursuers a linear path, where the output of one neuron will be the

input of another. The other way around, the output of a neuron is used as input of the same neuron, which is appropriate for non-linear applications.

2.6.2 Learning Rules

As illustrated in the Figure 2.32, in a ANN the inputs, before summing, suffer an adjustment by the synaptic weights. To define these weights, learning rules are used, that is, self-adaptive equations that update the weights and bias levels of neurons, enabling a neural network to learn from its input data and enhance its performance. There are diverse types of learning [79]. Some examples are:

- Hebbian learning rule
- Perceptron learning rule
- Delta learning rule (Widrow-Hoff rule)
- Competitive/correlation learning rule (Winner-takes-all)
- Outstar learning rule (Grossberg learning)

For this thesis, only the delta learning rule or Least Mean Square (LMS) algorithm will be considered. It was introduced by Widrow and Hoff in 1960 in [80], and its objective is to minimize the sum of squares of the linear errors over the training set. Additionally, a typical neuron used in NNs is the "ADAptive LInear NEuron" or ADALINE. It consists of an adaptive linear combiner cascaded with a hard-limiting quantizer, which generates a binary output (-1 or 1) used, e.g., for classification problems. A schematic can be seen in the figure below. If it is removed, the output will be analog, which may be useful, for instance, to noise-canceling applications [81]. Further, it is possible to have multiple layers of neurons to address more complex systems (Multiple ADAptive LInear NEuron, or MADALINE).

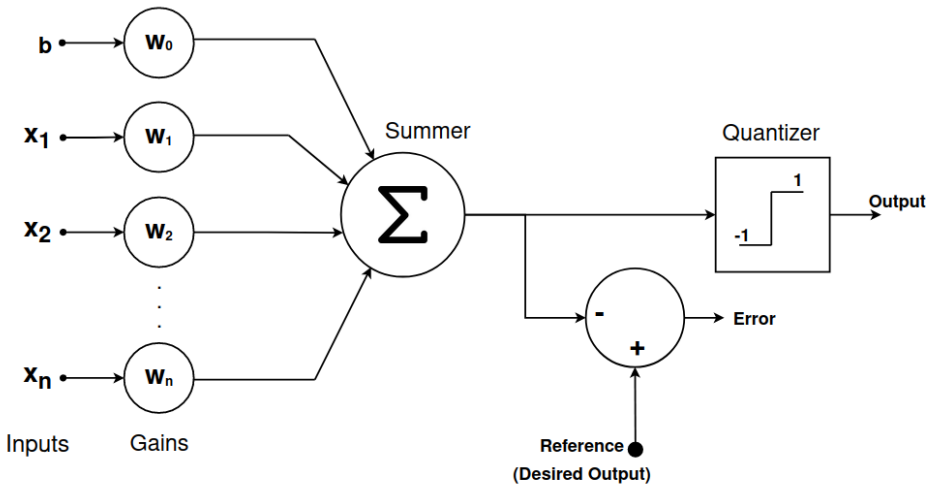


Figure 2.33: Schematic of ADALINE [80]

The linear error is defined to be the difference between the desired response and the linear output. After that, this error signal is used to train the NN, regarding the Equation 2.1 [82].

$$Wi_{k+1} = Wi_k + 2\mu\varepsilon_k Xi_k \tag{2.1}$$

Where Wi is the weight associated with the input i, k defines the time, μ is a parameter that controls stability, called learning rate, ε_k is the linear error, and Xi_k is the input i.

2.6.3 Machine Learning in Simulation

ML and simulation are two different topics that can complement each other. For example, when evaluating risk management, where the behavior of the system is based on causal relationships, hidden dependencies, etc. [83], a combination of both worlds could be beneficial. L. von Rueden in [84] presents a work that defines three subfields when combining both, simulation-assisted ML, machine-learning-assisted simulation, and hybrid.

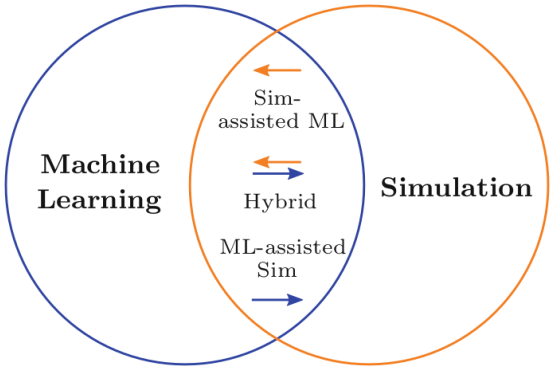


Figure 2.34: Subfields of combining ML and simulation [84]

Machine learning in simulation is often used to detect patterns in data or support the solution process. It can integrate all four components of the simulation. These components are: model reduction [85], input parameters [86], numerical method [87] and simulation results [88]. Because of this versatility, ML is an important part of the simulation world and should be used to obtain better conclusions and performance. Yet, it is not being fully exploited, that is, simulations, especially in the industry, are run with very specific analysis goals, ignoring further analysis that could be interesting for other contexts [84].

2.7 Co-Simulation

The development of truly complex engineered systems that integrate physical, software and network aspects is on the rise [89]. The simulation of these all together is not reliable, hence the need to divide

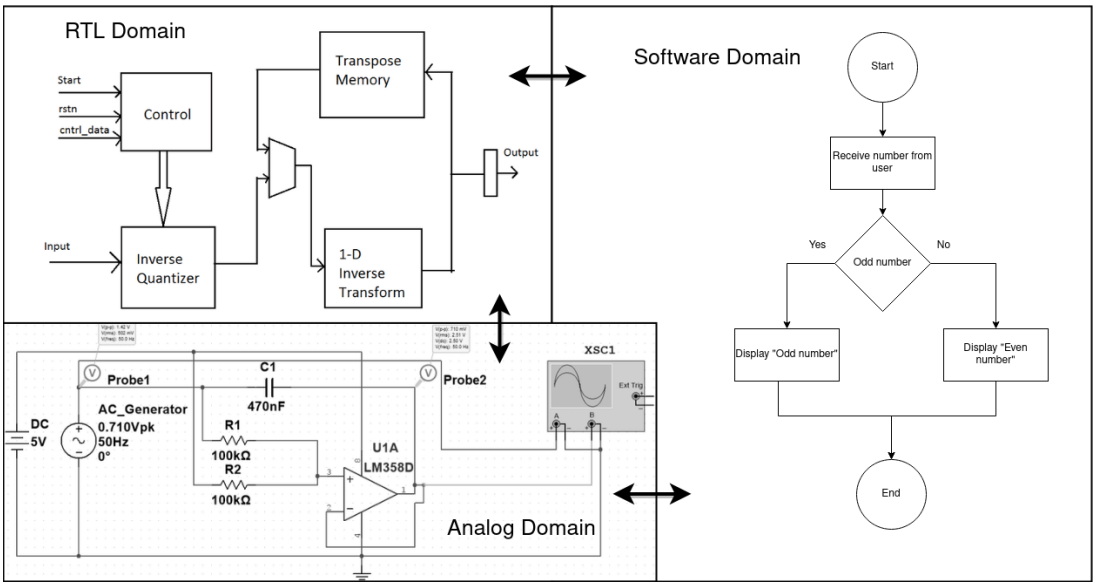


Figure 2.35: Different domains of a complex system

the system into different teams arises. Each one develops a part of the final solution, that, in the end, is integrated into the others. Different abstraction layers may demand distinct simulation tools, as some are finely tuned for specific domains, yielding results that closely mock real-world behavior. The Figure 2.35 shows an example of different domains in a complex system.

Normally, each part is tested and verified alone, that is, without iterations from other modules [89]. However, information on another part of the system may be needed, for example, a wave input to produce a sound. A possible solution can be the generation of local files containing the necessary information nevertheless, interactions are not thoroughly tested. This may create validation failures, which can result in delays in the development and extra costs in the development.

Co-simulation is proposed as a solution to overcome the latter issue [89]. It consists of a combination of simulators that execute concurrently, with one providing information to the others and vice versa. This approach is used in different domains, as shown in the Figure 2.36. To have the co-simulation environment it is required an Application Programming Interface (API) that creates the interface between the simulators like LibSystemCTLM-SoC, which contains various SystemC/ TLM-2.0 modules that enable co-simulation of Xilinx QEMU, SystemC/ TLM-2.0 and RTL [90]. If no API is provided, the simulator can also be extended to allow such interactions, as long as it is open-source.

One of the challenges when running a DES co-simulation is synchronization. As previously mentioned, there are different iterations of different abstraction levels, which can result in temporal misalignment. In other words, causality error can happen due to a distinct time granularity. This problem is similar to the one faced when executing a parallel simulation, and the solutions applied to address them are remarkably consistent.

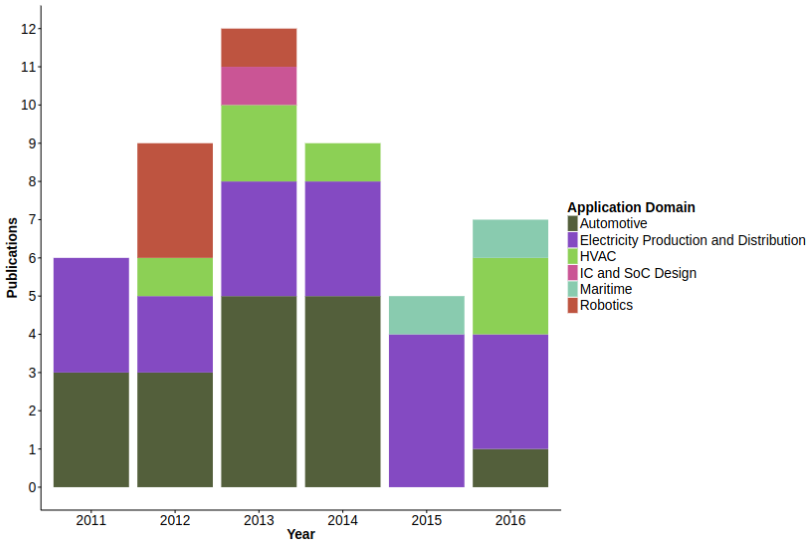


Figure 2.36: Research publications of co-simulation applications over five years[89]

2.7.1 Gem5 and SystemC

SystemC is being widely used in the industry and research hardware design space exploration. [91]. Nonetheless, there is a notable scarcity of precise, freely available, adaptable, and lifelike SystemC models for contemporary CPUs. Christian Menard in [91] developed an API that allows full interoperability between the two simulators.

Figure 2.37 presents three possible scenarios for the co-simulation environment. The issue here is that achieving this co-simulation requires compiling Gem5 as a library and then using it in SystemC. This does not fully adhere to the previously mentioned co-simulation definition because the two simulators are not running simultaneously in separate processes. This limitation can lead to verification failures as direct inputs from other simulators to Gem5 are not possible.

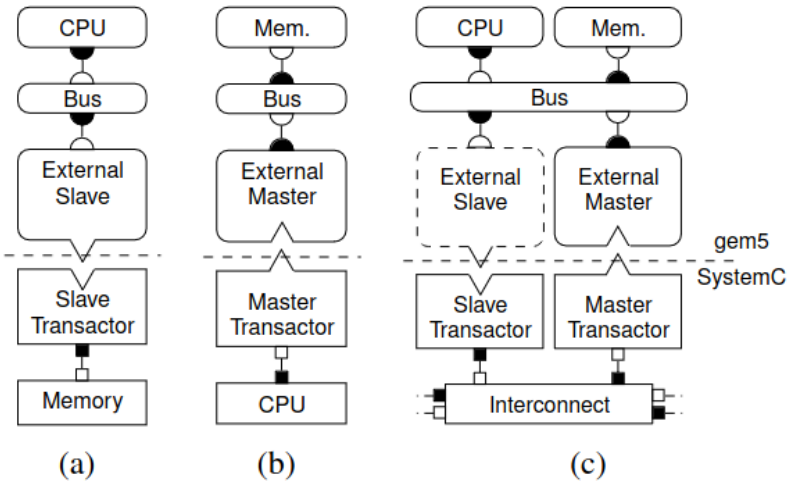


Figure 2.37: SystemC and Gem5 interoperability [91]

3 Pareleliammscpvsdj vbsuidb

The development of par-gem5 [3] solved one problem of Gem5 [34], which is the low performance even with a power full host machine. As exhibited in the subsection 2.5.4, the accuracy is no longer perfect, and a trade-off between these occurs. Concerning the current works on this subject, none of them fulfilled the three characteristics presented in the Table 2.3, therefore they can not be employed in the par-gem5.

To overcome these limitations and find the optimal quantum automatically, there were developed some algorithms in the scope of this dissertation, and in this chapter, they will be described and explained in detail. These algorithms are called: ADALINE, increment, Program Counter (PC), and repetition. As the name suggests, the first one is based on the ADALINE NN. The following is used by the previous to dynamically adapt the increment value. PC algorithm analyses the executed instructions to identify if their execution may result in inaccuracy. These two provide support for the remaining ones, meaning they cannot function independently. The last one tries to identify loops within the execution of the simulation to accordingly adapt the quantum.

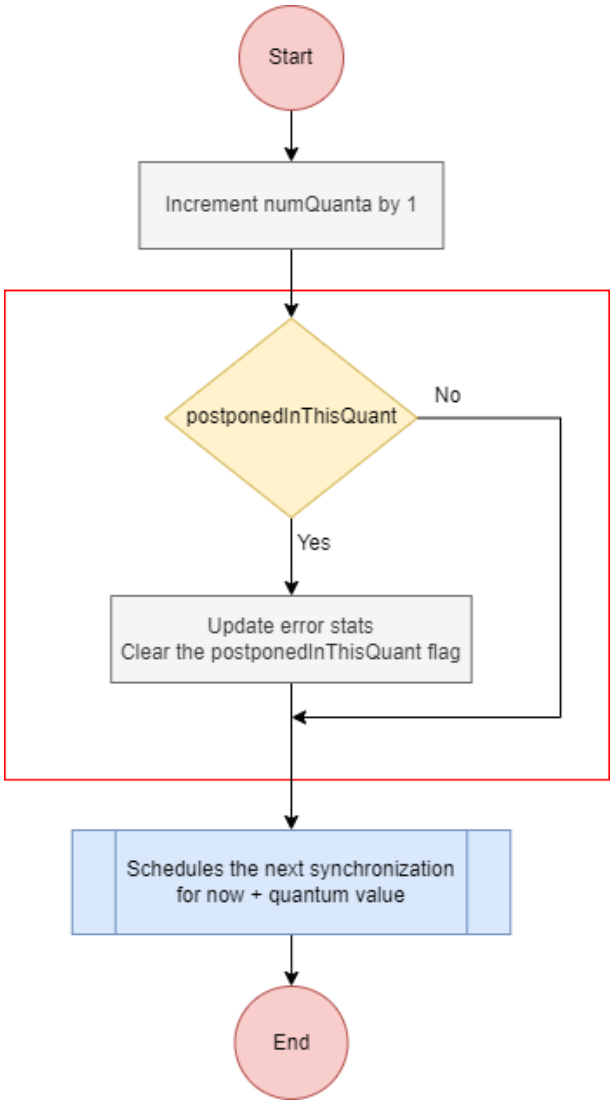


Figure 3.1: Quantum definition in the synchronization process

In the current state of par-gem5, the quantum attribution is done as presented in the latter image. The simulator enters a synchronization state that establishes whether the simulation should continue or not. If affirmative, the next synchronization is scheduled for the current time plus the pre-defined quantum. If not, it finishes the workload. The red rectangle defines the zone where the adaptive quantum should be implemented, in a way that the schedule function will use, for the next synchronization event, the quantum resulting from the algorithm.

3.1 Benchmarks

When an engineer designs an ASIC, for example, in the end he needs to verify if the project meets the requirements. Therefore, the engineer needs to execute tests that are called benchmarks. They are designed to measure the performance, capabilities, or efficiency of a system, component, or software

application. There are different types of benchmarks, each one emphasizing an area of interest. CPU, network, and storage are some examples.

In the context of this dissertation, it will be used two CPU benchmarks, the bare-metal bubble sort and the NAS Parallel Benchmarks (NPB). The host and the target systems have the configurations present in the Table 3.1a and Table 3.1b respectively.

Table 3.1: System Configurations

(a) Host

CPU	AMD Ryzen 3990x (64 cores, 128 threads)
RAM	128GB of 3200MHz DDR4-DRAM
OS	Ubuntu Linux 20.04

(b) Target

CPU	ARM64, AtomicSimpleCPU @ 2GHz
Caches	64kiB L1-D, 32kiB L1-I, 2MiB L2 shared
Main Memory	DDR3 RAM @ 1600MHz
Periph. Sub-system	Real View Virtual Express V1

3.1.1 Bare-metal Bubble Sort

The main objective of this benchmark, as implied by its name, is to rearrange an array in such a way that elements with higher values are positioned at the top. The algorithm employed for this task is quite straightforward. It involves a loop that iterates through the input list, examining each element in turn. During each iteration, the algorithm compares the current element with the one that follows it and, if necessary, swaps their values. This process continues until the entire array is sorted according to the desired criterion. It was designed to attain a near best-case simulation throughput, meaning that thread synchronizations and accesses to shared memory are reduced to a minimum.

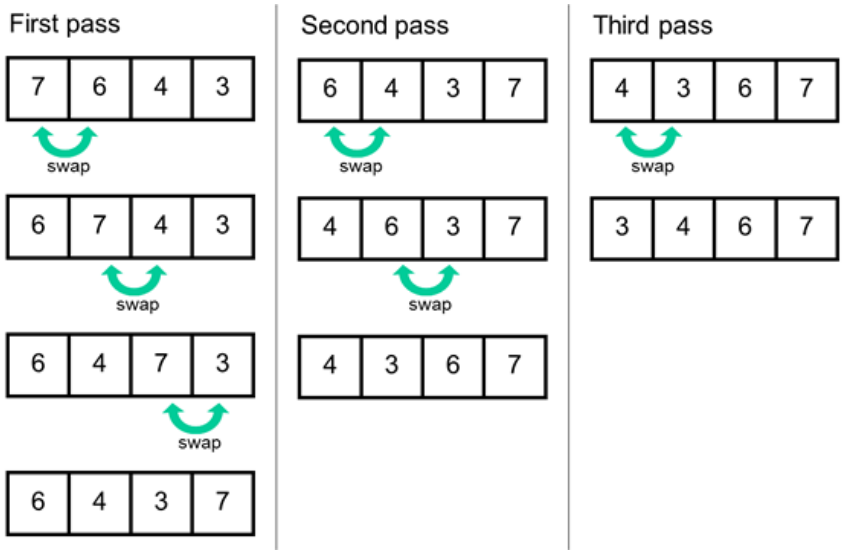


Figure 3.2: Bubble sort example

This benchmark operates in a bare-metal environment, which means it is implemented without the use of an operating system (OS). In a bare-metal setup, the program runs directly on the hardware without

the mediation or assistance of an operating system. This approach is often chosen for performance-critical or resource-constrained applications where direct control over the hardware is necessary, and there is no need for the additional services and abstractions provided by an operating system.

3.1.2 NPB

The NAS Parallel Benchmarks (NPB) [92] is a group of a small set of programs designed to help the performance evaluation of parallel supercomputers. In total, there are eight benchmark specifications, five kernels, and three pseudo-applications. The kernel benchmarks are:

- **Integer Sort (IS):** Performs a sort operation where both integer computation speed and communication performance are tested.
- **Fast Fourier Transform (FT):** Solves a three-dimensional partial differential equation using Fast Fourier Transforms (FFTs). Long-distance communication performance is the main evaluation point.
- **MultiGrid (MG):** It is a simplified multigrid kernel that requires highly structured long-distance communication, thus short and long-distance data communication are evaluated.
- **Conjugate Gradient (CG):** Computes an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. The main goal is to test irregular long-distance communication.
- **Embarrassingly Parallel (EP):** It provides an estimate of the upper achievable limits for floating point performance, with no significant inter-process communications.

The pseudo-applications are the **Block Tridiagonal (BT)**, **Scalar Pentadiagonal (SP)**, and **Lower-Upper symmetric Gauss-Seidel(LU)** benchmarks. Each of these benchmarks is designed to solve a synthetic system of nonlinear partial differential equations using a distinct algorithm. The names of the benchmarks correspond to the specific algorithms employed in solving these mathematical problems.

Moreover, NPB implemented benchmark classes, that is, the problem size of each workload can be modified. There were implemented 8 problem sizes (S, W, A, B, C, D, E, and F), where S is the smaller one and F is the bigger one. In the context of this dissertation, only the W size will be considered.

3.2 ADALINE

Adaptive filters are commonly used for the reason that they possess the ability to automatically adjust their own parameters and their design necessitates minimal or no prior knowledge of signal [93]. One application of this filters can be in the development of a Adaptive Noise Cancellation (ANC) algorithm. A generic scheme can be found in the image bellow.

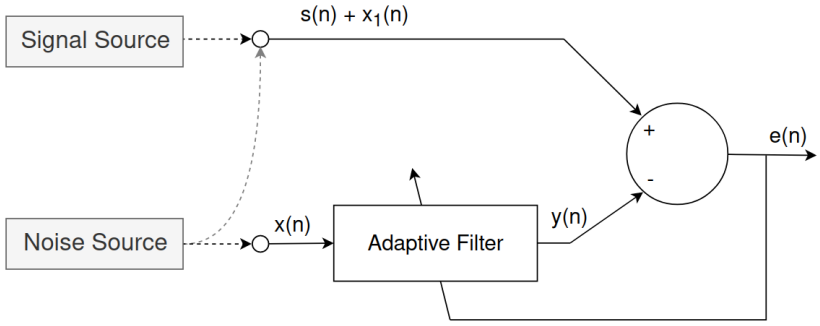


Figure 3.3: ANC scheme

$S(n)$ is the input that has the signal mixed with the noise, and $X(n)$ receives only the noise, hence it is a noise reference. If the noise in $S(n)$ was the same as the one presented in $X(n)$, it would only be needed to subtract the $X(n)$ to $S(n)$, nevertheless this noise is different because it is attenuated, delayed, and filtered by noise path. For these reasons, an adaptive filter should be utilized, allowing $Y(n)$ to closely resemble $X_1(n)$. $E(n)$ will be the output without the error. It is also used by the filter in order to adapt their weights.

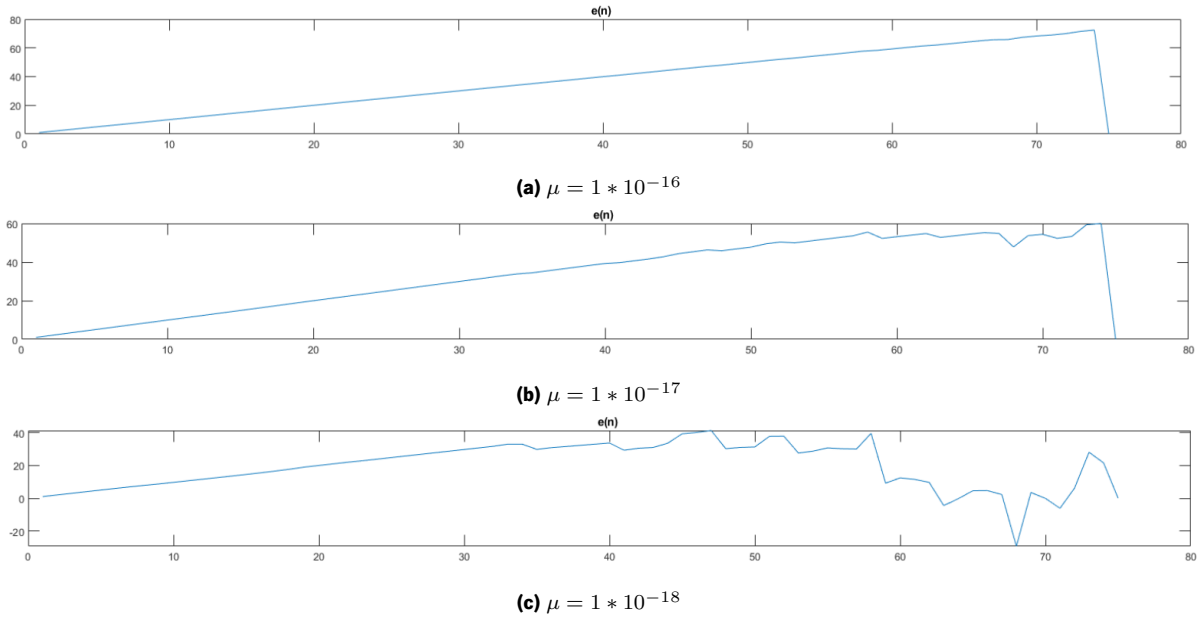
A comparison can be made to the par-gem5 world. The signal is the quantum, the noise is the simulation error, and the output is the quantum without the error. Also, the $X_1(n)$ and $X(n)$ are different due to the impossibility of calculate exactly $X_1(n)$ in runtime. When a benchmark is running, the simulation error obtained is refereed to the worst case scenario. In other words, the simulator analyses when there is a cross-schedule event, and the time difference between when that happens and the next synchronization is considered the error, since it considers that event must be postponed. However, most of the times it is not true, and the event is not postponed, causing a smaller error. To accurate analyse what was the impact of inaccuracy, to obtain $X_1(n)$, it would be needed to run firstly the sequential simulation, killing all the benefits of par-gem5 [3]. The next equation describes how the worst case error estimation is calculated.

$$e_{rel,t} = \frac{t_{sim,meas}}{t_{sim,meas} - \sum_{i=0}^{Q} t_{i,max\_pp}} - 1 = \frac{t_{sim,meas}}{t_{sim,est}} - 1 \tag{3.1}$$

In each quantum the postpone-mechanism records which event experienced the most significant time shift caused by the postponement, t_{i,max_pp}. Q is the number of simulated quanta and $t_{sim,meas}$ is the measured simulation time.

3.2.1 Adaptive Filter

Following the Figure 2.34, the adaptive filter will be responsible for adapting the weights of the ADALINE NN. It uses the Equation 2.1 for the training, where it is essential to carefully choose an appropriate learning rate.



(a) $\mu = 1 * 10^{-16}$

(b) $\mu = 1 * 10^{-17}$

(c) $\mu = 1 * 10^{-18}$

Figure 3.4: Control action with different learning rate values

The learning rate is a constant that is chosen at the beginning of the simulation and it is not modified. This parameter compromises a trade-off between control speed and stability. With lower values, the learning process is fast nevertheless, it is likely to become an unstable control system. With higher values, stability is granted, but the learning process is very slow, taking a lot of time to be operational. Finding the best value is not linear, therefore an iterative approach was used.

A small simulation was done and in each quantum synchronization, the values of the quantum and the error were recorded. Furthermore, in this test, the quantum was incremented 1 microsecond every time a record was done. Then, the ADALINE algorithm was implemented in Matlab, obtaining the results illustrated on Figure 3.4.

The aforementioned mention trade-off can be observed, where for $\mu > 1 * 10^{-16}$, the results were similar to Figure 3.4a, and for $\mu < 1 * 10^{-18}$, the system become even more unstable. To choose the learning rate, the logarithmic scale is used, as small variations in the μ do not result in a significant change. From the results was concluded that the best value for μ is $1 * 10^{-17}$.

Although on this test when μ was equal to $1 * 10^{-17}$ did not result in an unstable control system, there is a possibility that it could happen, for example, when there are huge error variations. Typically, these variations occur when the synchronization time is much longer than the ideal case, which requires to reset the algorithm. Regarding the results on [3], where it was observed the inaccuracy grows with increasing quantum and number of cores, it was also implemented a *safeQuanta*, a value that is smaller than the actual quantum. It depends on the previously mentioned characteristics, hence it can adapt to the present situation. The next flowchart illustrates how the reset system works.

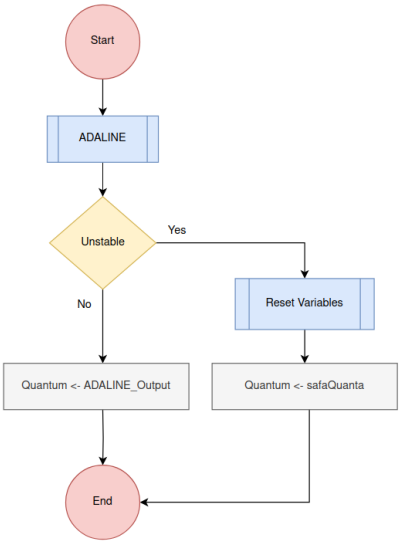


Figure 3.5: Reset system in ADALINE algorithm

If the ADALINE output is negative, means it becomes unstable, and needs a reset. In the reset, all the variables used in the calculation are set into their initial conditions, sacrificing simulation performance. For this reason, it is desirable to have the least resets possible.

3.2.2 TDL

As the simulation precedes, there is more data to analyse, taking away performance from the NN. Stella et al. [94] presented one example of an application where this problem is also present. The solution to make full use of the ADALINE network as an adaptive filter was the implementation of a Tapped Delay Line (TDL). Its operating method is described in the figure below.

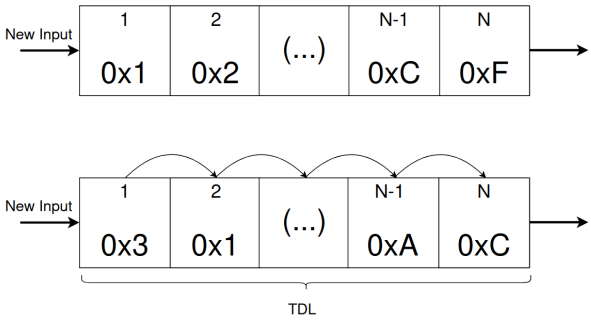


Figure 3.6: TDL working method

One input is considered N times, being N the size of the TDL. When a new input arrives, the oldest one is discarded. In the end, the TDL have always the signal at the current time, the previous input signal, and so on. Combining the TDL with the ADALINE, efficiency, and performance are no longer sacrificed. This approach was implemented in the noise source since it receives new data in each new quantum evaluation.

TDL size varies from application to application. In this case, when the size is small, the learning process may be subpar, but it also becomes less sensitive to variations in the noise. Conversely, a larger size leads to more effective weight adjustments through improved learning, but it makes the NN more susceptible to noise variations. To choose the best size for this scenario, a small test was done. The results are presented in the following graph.

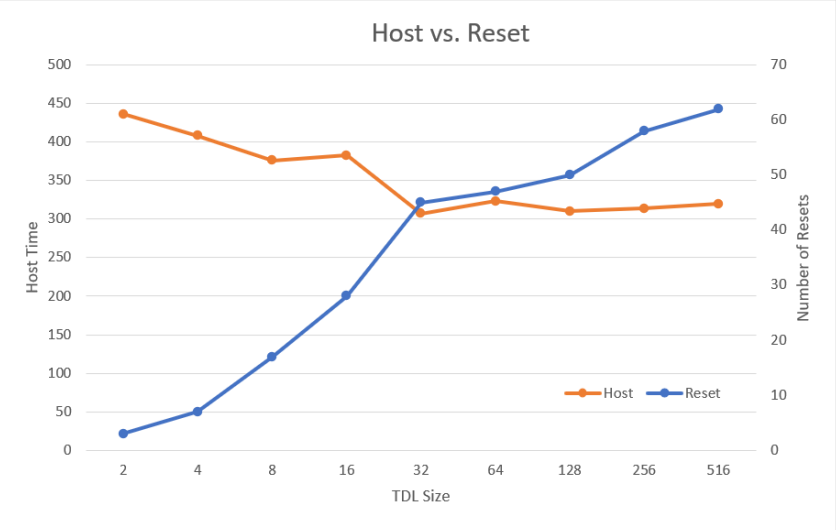


Figure 3.7: Reset vs. Host

To have maximum performance on the simulator, only base 2^n sizes were chosen, where $n \in N$. Observing the outcome, it can be seen that there is a point where the host time does not improve significantly, however the number of resets continues increasing. Each reset has a performance impact, as it requires resetting every variable to their initial conditions. Therefore, it is desirable to minimize the number of resets whenever possible. In the end, it can be concluded that the best size to be used is 32, and for this reason it was used for the subsequent tests.

3.2.3 Quantum Increment

The ADALINE's algorithm output, that is, the filtered quantum, will be always lower than source. Using this new quantum implies that there will be a point in time when it becomes too small, primarily because it fails to exhibit a value increment. This situation inevitably leads to a trade-off, wherein performance is compromised. To strike the optimal balance, it becomes necessary to increment the quantum value. This adjustment should occur when no errors are detected in the system. Figure 3.8 presents the flowchart for the ADALINE algorithm.

An error exists when an event must be postponed. Par-gem5 [3] introduces a flag called *postponedInThisQuanta*, that triggers if the previous scenario occurs. When the simulator enters in the synchronization state, this flag is verified to decided if the quantum should be incremented or not. The increment value should have a balance, in the way that with a smaller increment the performance may

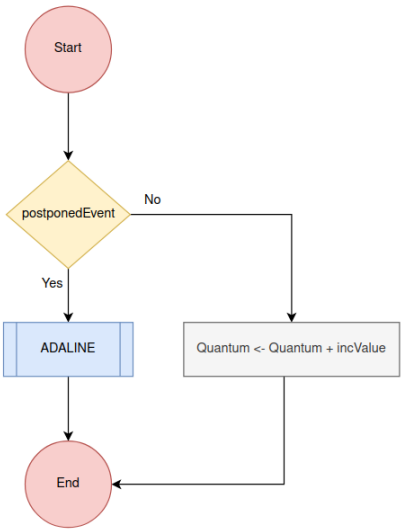


Figure 3.8: ADALINE flowchart

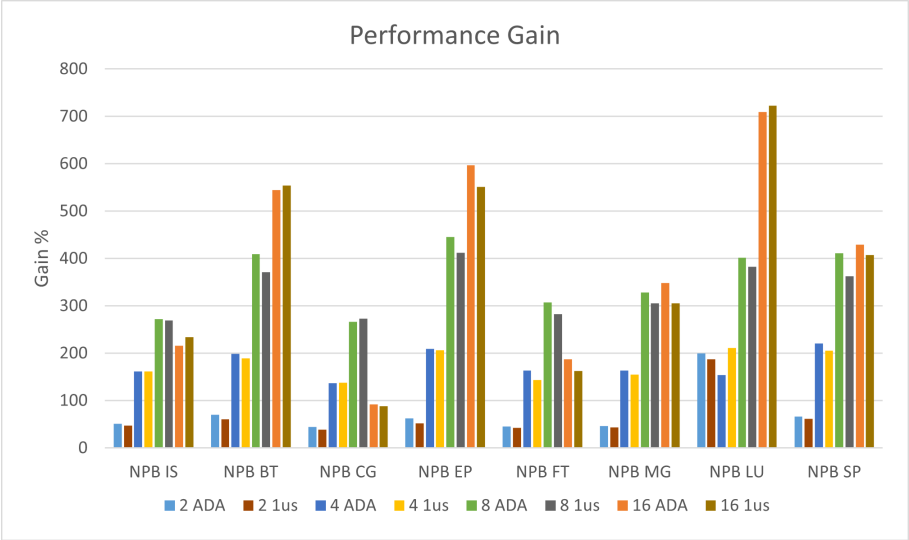
not be improved, and with a high value the accuracy may be ruined. According to [3], the quantum of 1 microsecond gives a good trade-off between the those two, thus it was chosen to use a increment 10 times lower. This value enables a significant balance without causing any harm.

3.2.4 Results

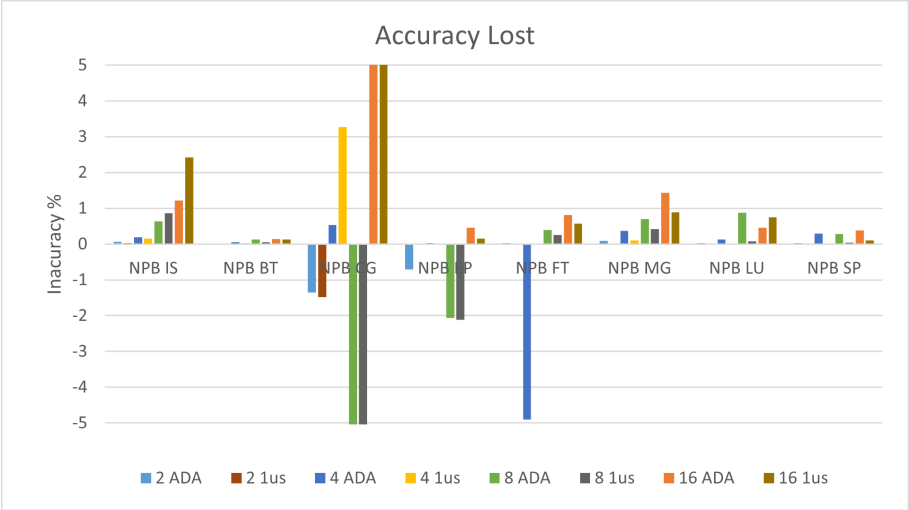
The aforementioned benchmarks executed the developed algorithm with different number of cores. The results obtain are depicted in the Figure 3.9.

The horizontal subtitle illustrates the executed benchmark along with the number of cores and the corresponding algorithm used, hence 4 ADA means it were simulated four cores with the ADALINE algorithm. To assess the performance of ADALINE, the static version was also executed with a quantum set to 1 microsecond. In all cases, the comparison reference is the result obtained from sequential simulation. Additionally, the results for a single simulated core are not presented because in all conducted tests, the sequential simulation exhibited similar performance. Hence, this scenario is not relevant for evaluating the algorithms.

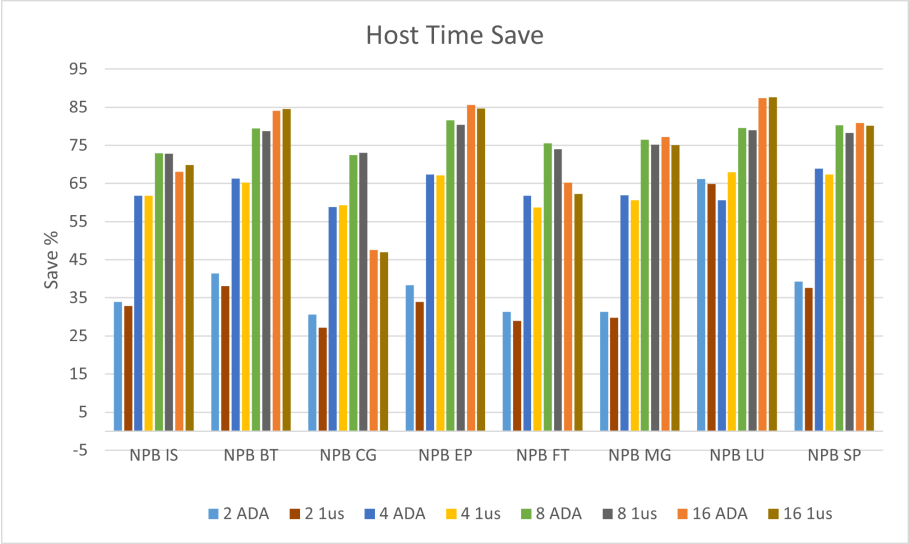
When it comes to performance, it's noticeable that the algorithm yielded better results in certain benchmarks such as baremetal, NPB EP, and SP. However, in other cases like NPB IS and CG, it did not show significant improvements. Shifting the focus to accuracy, the static version generally outperformed the dynamic approach. Nevertheless, the dynamic approach consistently maintained an accuracy of under 2%, with the exception of NPB CG, where both approaches performed poorly. Lastly, when comparing the host time to performance, similar outcomes were observed, though certain benchmarks exhibited superior results to others.



(a) Performance gain



(b) Accuracy lost



(c) Host time save

Figure 3.9: ADALINE algorithm results

This performance issue may be related with the previous quantum increment consideration. It is clear this value for some workloads is good, but for others may not be the most adequate. Therefore, having a dynamic value would solve this problem, resulting in more flexible algorithm. Moreover, special attention should be given to its design, as excessively large values can lead to an unfavorable trade-off.

3.3 Increment Algorithm

As shown, it was verified in some cases where the performance was equal or lower when compared with the static approach. Concerning the quantum increment, a fixed value was defined following the aforementioned philosophy however, a dynamic approach may bring better results, as different benchmarks require different needs.

Therefore, the ADALINE algorithm was updated to integrate this new functionality. Observing the Figure 3.8, the calculation of the increment value (incValue) will be done before the if statement, so it can still be used in this quantum's calculation. The new flowchart is represented in the next figure.

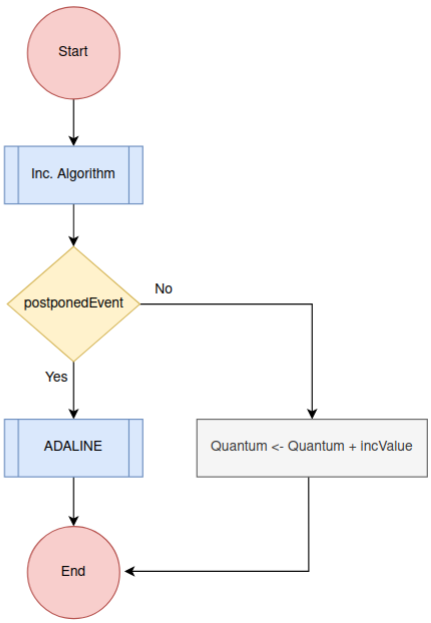


Figure 3.10: ADALINE with increment algorithm flowchart

The concept is to commence with a gradual increase in value initially. If there are no postponed events during this period, then the increment can be escalated. It can be seen as an exponential however, an approach like this would scale too fast, affecting the accuracy. A solution can be the definition of a threshold, where the increment value only increases if no accuracy issues have happened N times in a row. An example is illustrated in the Figure 3.11.

The incValue starts in the smaller value possible, which is equal to the clock period, and grows until it reaches a maximum, that is 100 microseconds. It was defined regarding the results on [3] and [64],

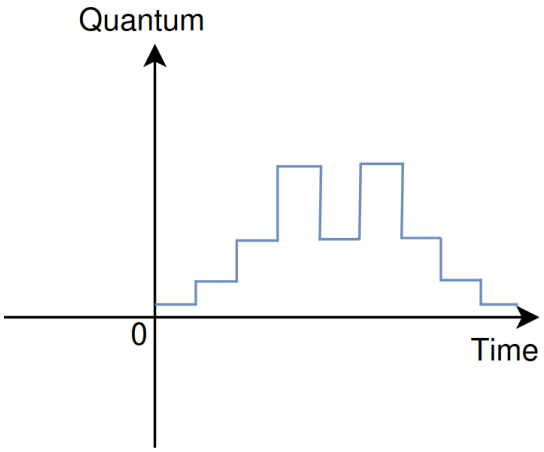


Figure 3.11: Example of evolution of the increment value with a threshold

where the benefits of a quantum of this magnitude are few. The addition method respects the subsequent equation.

$$incValue = 2 * incValue \tag{3.2}$$

Oppositely, the decrement is done by dividing by two the actual incValue, until it reaches the minimum. The threshold value depends on how many cores are used by the simulation since the inaccuracy grows with the increasing number of cores. The more cores are being used, the greater the threshold. Moreover, the increment value only changes (increases or decreases) if it fulfills the condition for N consecutive times. The later image shows the algorithm's flowchart.

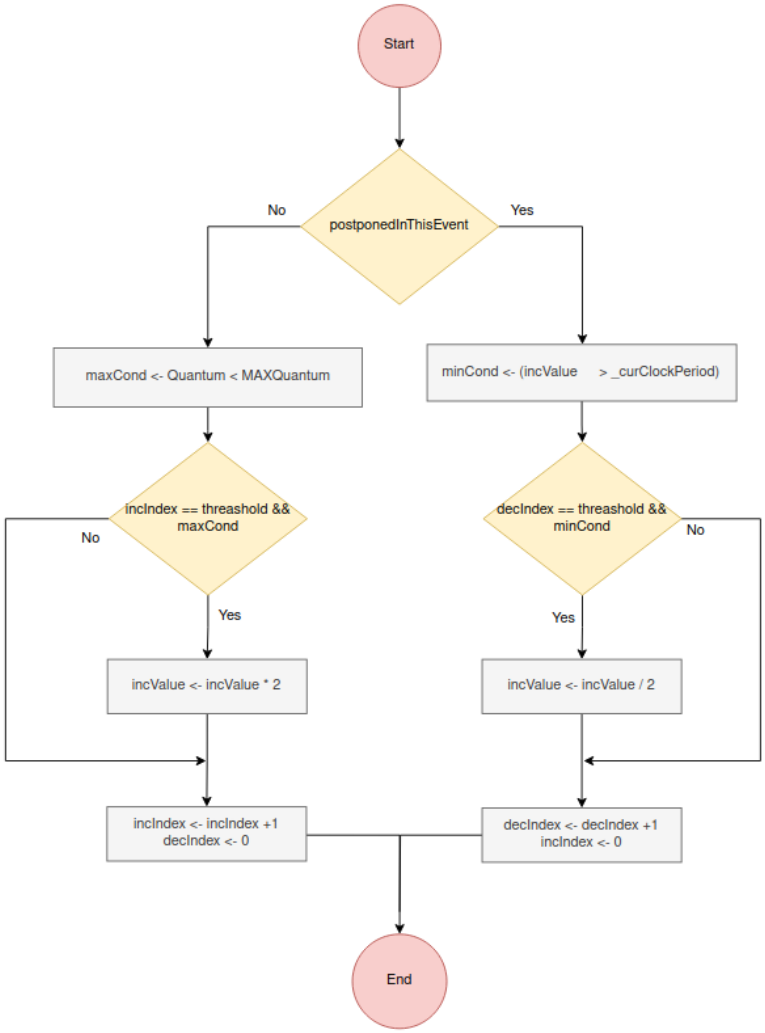
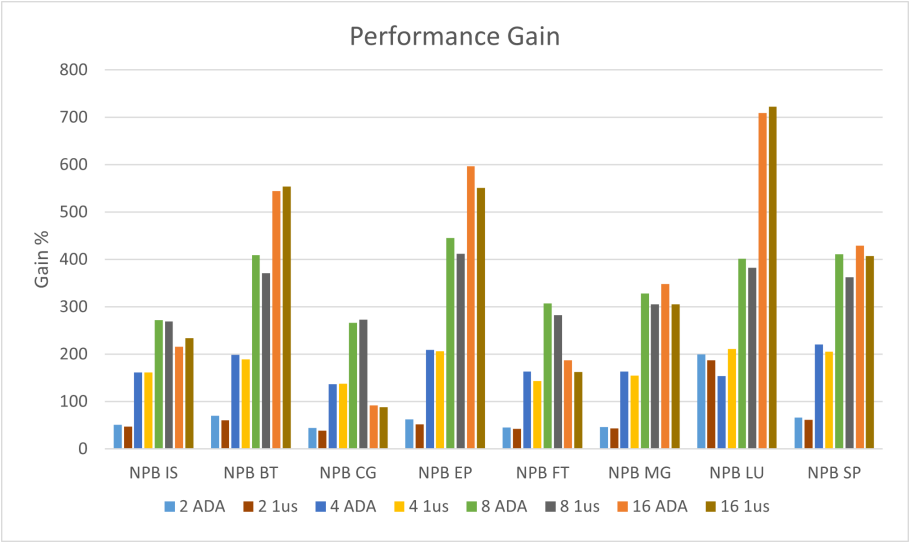


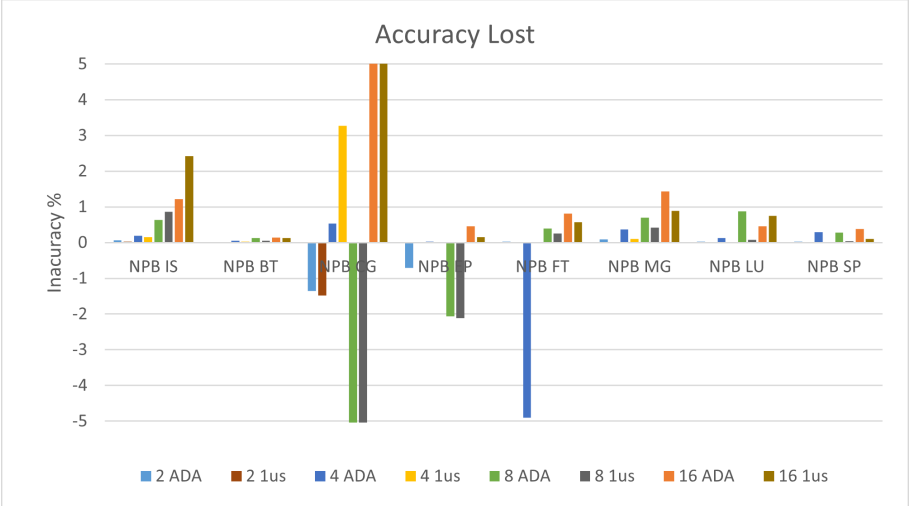
Figure 3.12: Increment algorithm flowchart

3.3.1 Results

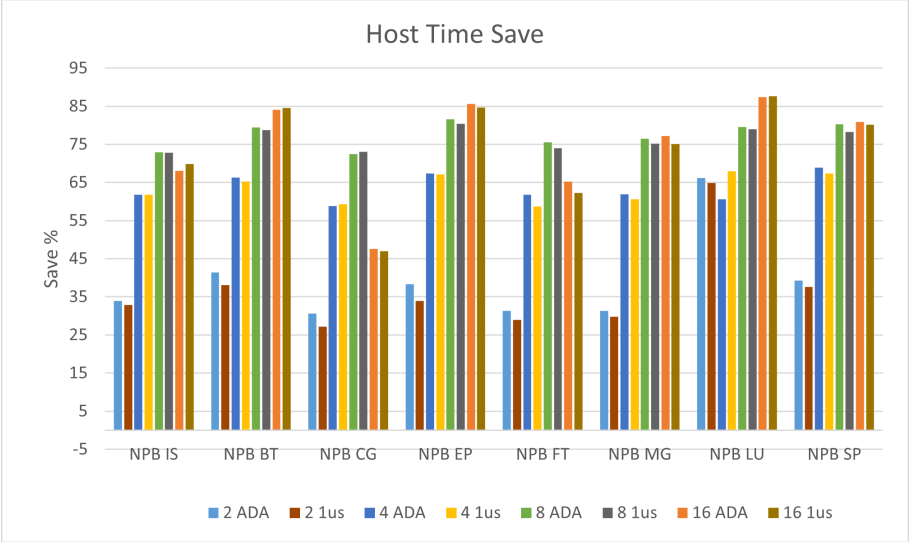
The previous benchmarks executed the new approach with different number of cores. The results obtain are depicted in the Figure 3.13.



(a) Performance gain



(b) Accuracy lost



(c) Host time save

Figure 3.13: Dynamic increment results

The ADA refers to the first results, while A-I is related to the new configuration. As expected, the dynamic approach in the increment value resulted in better performance. On average, the performance increment was about ..... Additionally, as a consequence of that, the host time was lower, reducing it by .....

Unfortunately, the same did not occur in terms of accuracy. In all cases, except for the NPB.LU benchmark with sixteen simulated cores, there was a reduction. In some cases, such as the NPB.FT and NPB.EP with sixteen simulated cores, experienced a significant drop, with accuracy falling below 95%. A possible explanation for this could be the nature of the algorithm itself. The adaptive filter may not be designed to handle drastic interventions. As shown in Figure 3.4b, the control action appears to be smooth, even though there are moments, for example, in inter-process communications when it should be more aggressive to prevent postponed events in cross-scheduling. The problem was not evident at the beginning, however, with the dynamic increment, it becomes more notable.

One of the requirements for this dissertation is to maintain a maximum inaccuracy of 5%, thus the current approach does not meet this criterion. It is crucial to explore alternative methods and optimizations to ensure that the desired level of accuracy is achieved.

3.4 PC Algorithm

As mentioned before, one problem of the previous algorithm is the incapability to reduce the quantum significantly in a short period. A potential solution for this issue could involve predicting when this situation is likely to occur. In other words, an attempt can be made to assess when a postponed event will take place. If it becomes possible to forecast when such an event will be postponed, adjustments can be made to the quantum before it happens. This preemptive action can help prevent significant inaccuracies from occurring.

Before the simulation initiates, the commands destined for execution are loaded into memory and remain unaltered throughout the simulation. These, when executed, can lead to a postponed event. An example of that is the Wait for Event (WFE) instruction, where the CPU enters in low-power standby state. Furthermore, in general, loops are the backbone of benchmarks, whether due to multiple iterations or the benchmark's inherent characteristics, such as testing the transfer of memory blocks between different locations. As a result, the same instruction address can be encountered more than one time.

By evaluating the PC, it is possible to understand if the processing of one instruction can result in a postponed event (PPaddr), and so avoid its execution with a large quantum. With this idea in mind, the PC algorithm, as the name suggests, will analyse the PC during the simulation, in order to perform the prediction of when these instructions will be executed. Figure 3.14 exemplifies an example how the algorithm works.

In the beginning, every instruction is considered risk-free, but as the simulation proceeds, it can be changed. When it happens, the processed instruction becomes a spotted address, marked in red in the

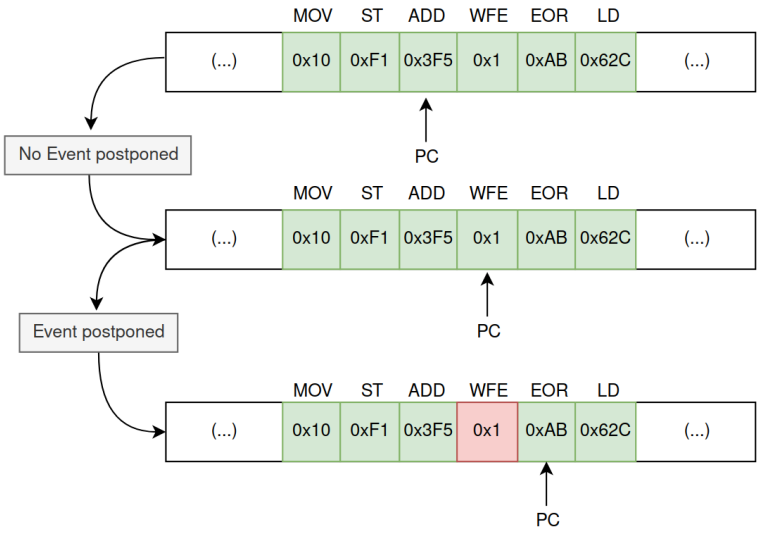


Figure 3.14: PC Algorithm

figure, until the end of the simulation. All these addresses will be taken into account in the new quantum calculation, thus these should be acknowledged as genuine "postponed event generators".

3.4.1 Forecast

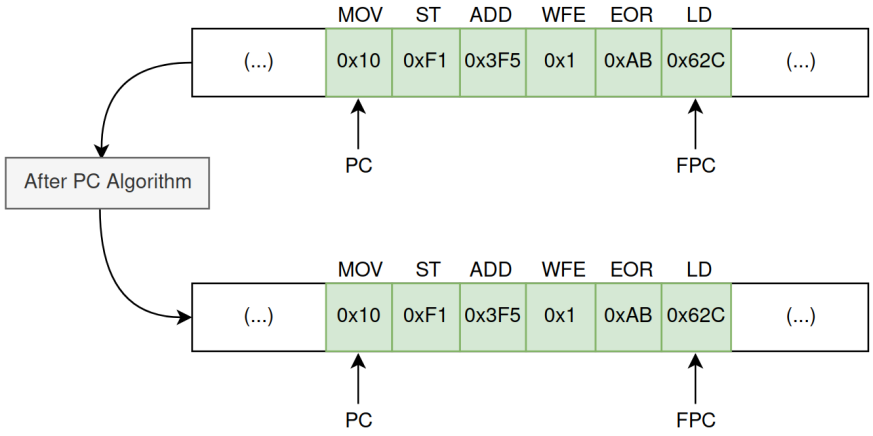
Workloads are not straightforward in a way that the PC do not follow a linear path. Instructions like "jump" (JMP) or interruptions have the potential to redirect the execution flow to different memory locations. Understanding exactly what type of instructions the simulator will execute and tracking them is computationally heavy, meaning performance costs. Since the algorithm should be lightweight, it is considered that the PC is linear, that is, the next PC will be the actual plus the instruction width.

First of all, is used an analytic method to find where the program counter will be in the future. As a result of the previous consideration, the Equation 3.3 was developed.

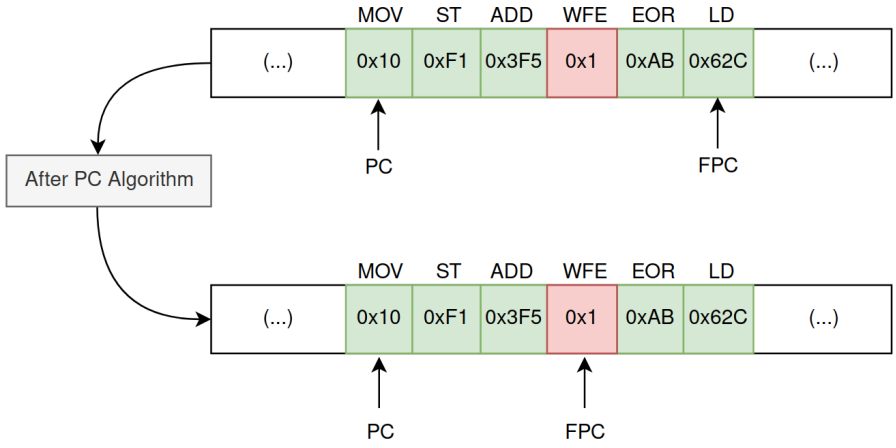
$$FPC = PC + \left(\frac{Quantum * InstWidth}{CyclePeriod} \right) \tag{3.3}$$

$$Quantum = \frac{(FPC - PC) * CyclePeriod)}{InstWidth} \tag{3.4}$$

Thereafter, one of two scenarios can unfold, as depicted in the next pictures. First, nothing may transpire if no addresses are identified between the PC and FPC (Forecasted Program Counter). Alternatively, if the FPC corrects its value to the nearest identified address, a quantum recalibration is necessary, achieved using Equation 3.4. Hence, the synchronization will occur right after the execution of the spotted address, avoiding inaccuracy as the cross-schedule event is inserted in the target event queue at the expected time.



(a) Scenario 1



(b) Scenario 2

Figure 3.15: Possible scenarios after the forecast

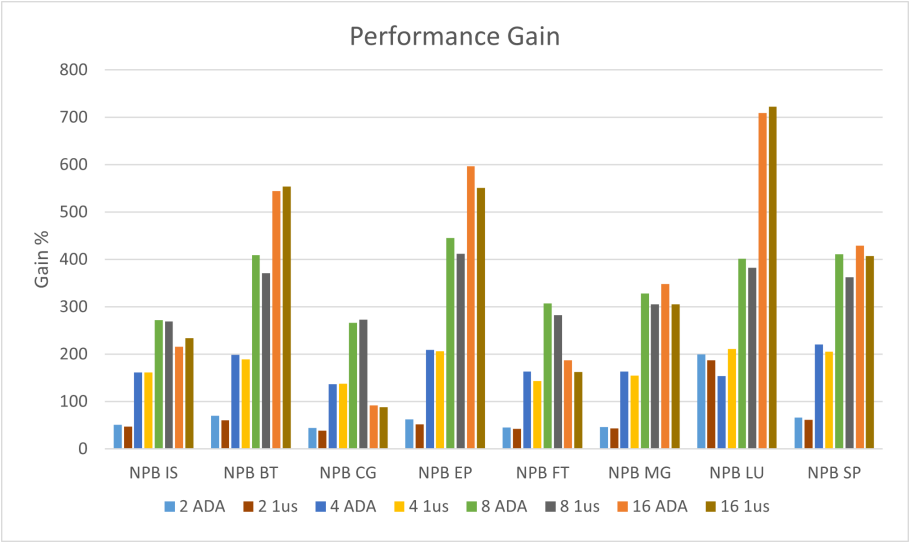
3.4.2 Increment Algorithm Update

With the development of this new algorithm, there is a new way to verify whether the quantum is reduced, meaning the old one was greater than the desirable. By this reason, the increment algorithm will also verify this situation, counting as ”inaccuracy problem”, contributing for the reduction of the increment value.

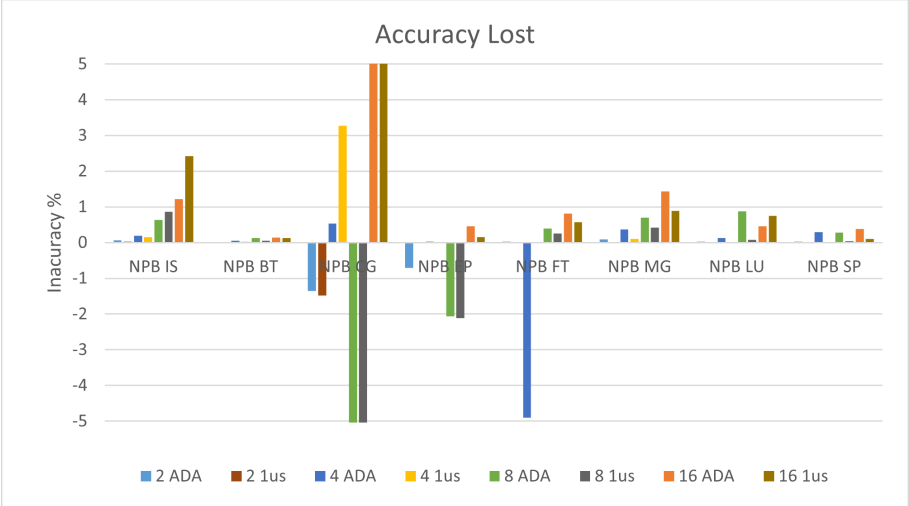
To determine whether the previous scenario occurred, the quantum value before the PC algorithm's action is stored and compared to the current. If the stored value is equal to or higher than the current quantum, no action is taken. However, if the stored value is lower, it indicates that the quantum was reduced, triggering the flag.

3.4.3 Results

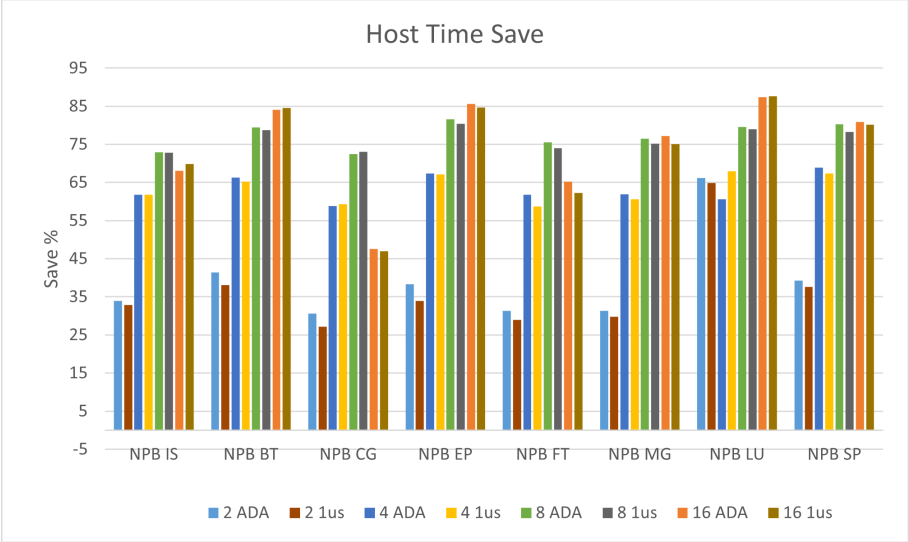
The results of the bare-metal bubble-sort and NPBs benchmarks execution are presented in the Figure 3.16. As previously, A-I refers to the ADALINE with the increment algorithm results, and A-I-P points to the results with the PC algorithm intervention.



(a) Performance gain



(b) Accuracy lost



(c) Host time save

Figure 3.16: PC algorithm results

Starting with the performance gain, it did not have a significant drop, only about ..... %. In practice, in most workloads this reflected in an increment of the host time of ..... seconds approximately. Nevertheless, there are a few cases in the algorithm's action conducted into better performance, in particular the NPB.BT over eight simulated cores.

The most distinguished side is the accuracy, where the PC algorithm inclusion allowed an inaccuracy reduction of ..... %. The previously mentioned cases that passed the 5%, are now within the limit. Moreover, analyzing the worst case of accuracy loss, a reduction can be seen in most scenarios.

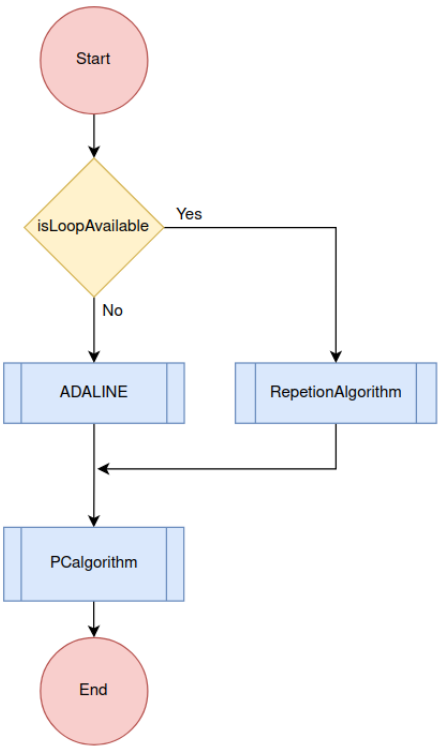


Figure 3.17: Worst case of accuracy lost comparison

3.5 Repetition Algorithm

As mention earlier in this chapter, loops are strongly present in benchmarks. Thinking in this perspective, if a record of every executed instruction is made, a loop can be identified in real-time. Combining this with the previous algorithms, the quantum could be adapt in a way to have the higher value possible. Furthermore, the accuracy would be nearly perfect since all cross-schedule events would be known, allowing for precise adaptation.

When the simulation starts, there are no information about loops or PPaddrs. To have this it would be needed to execute the simulation once and then obtain the results, what does not match with the requirements. One solution is to conduct the simulation in a regular manner while simultaneously recording the executed instructions. When a loop is detected, the algorithm begins utilizing the loop to calculate the

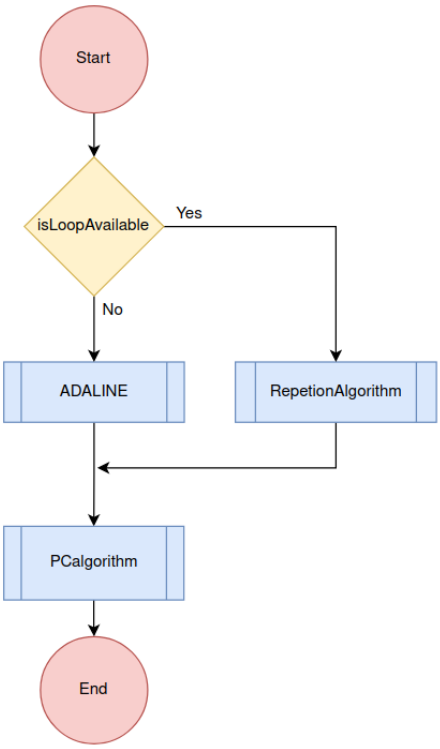


Figure 3.18: Quantum definition with repetitions algorithm

quantum until the PC no longer follows the loop's path. At that point, the loop is reset, and the process starts anew.

3.5.1 Hare-Tortoise Algorithm

The detection method must be light, otherwise the simulation performance is set in danger. One algorithm with this characteristic is the Floyd's Tortoise and Hare technique. It consists in two pointers, one twice faster than the other. If the two match at some point, means that there is a loop, as shown in the figure bellow.

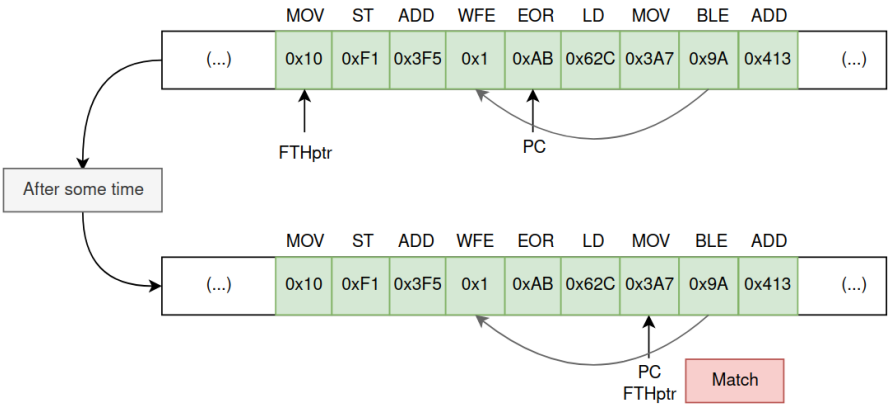


Figure 3.19: Hare-Tortoise Algorithm

In this case, the faster pointer is the PC, and the slower one is the FTHptr. A match occurs when both are pointing to the same memory at the same time. To delve further into the process, loop delineation can be achieved through two methods. Firstly, an analysis can be conducted on the preceding instruction to precisely define the loop boundaries. Alternatively, after the initial match, the FTHptr remains in the same position while the PC continues execution. When they match once more, the loop becomes well-defined. In terms of performance, it is trivial the second approach is more underweight, reason why it was chosen. After the detection, it is necessary to keep track of the PC, as referred earlier. It is done by comparing the actual PC with the expected one. Whether matches, nothing happens. In the opposite way, the loop is discarded and the hole process of detection starts again. The flowchart present in the Figure 3.20 describes the detection flow.

Although this technique is simple and very effective, it has some inherent problems. The first issue pertains to the inability to detect nested loops. Another challenge is the incapability to identify subsequent loops following the one that has been detected. The most crucial problem is the difficulty in detecting conditions within loops. All of these can be solved with the PC track nevertheless, these solution may cut the benefits of the algorithm, since these problems can be recurrent in the benchmark.

Another problem is the multi-thread environment. When more than one CPU is active, the PC executes more than one event queue, meaning an unpredictable pointer. In this situation is mostly impossible to identify any loop, thus a solution can be only apply the detection when only one CPU is on. If there is a loop defined and the other CPUs wake up, this is automatically discarded, and the algorithm stops, restarting again when the the later conditions revert.

3.5.2 Quantum Calculation

The quantum calculation is done in the synchronization point. It can be divided into 2 distinct tasks. The first one is to classify of the instruction of the loop. The second is the calculation itself of the new quantum.

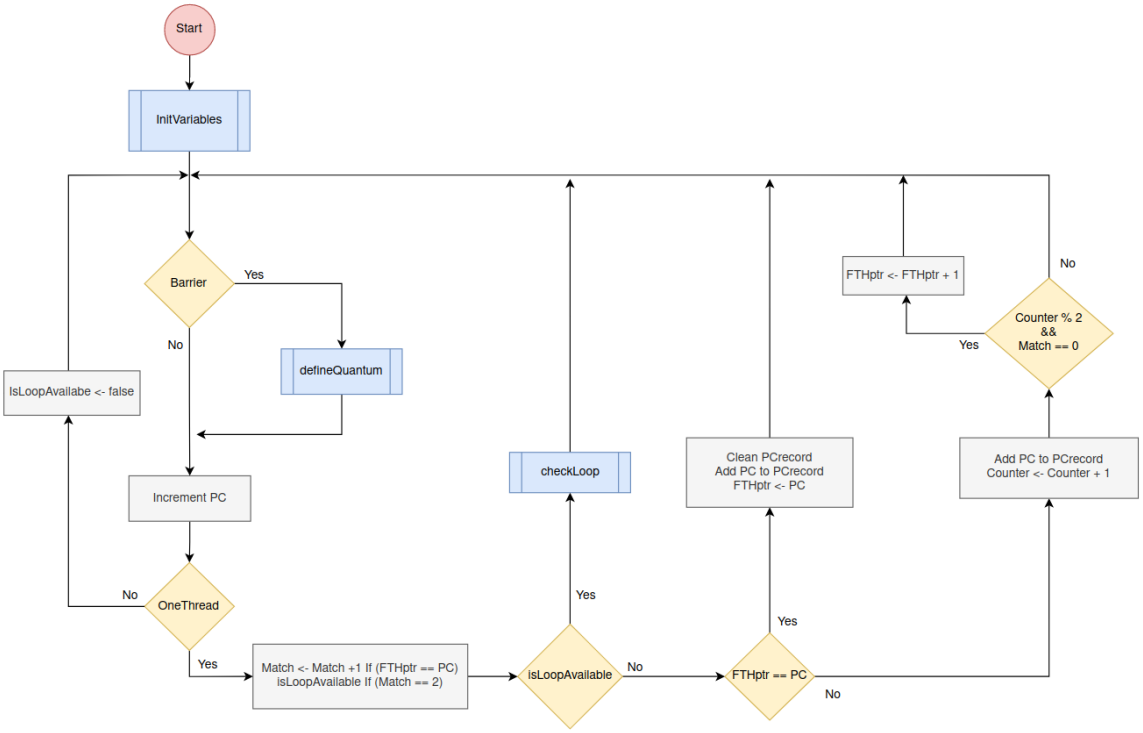


Figure 3.20: Loop detection flowchart

Regarding the primary task, it is only needed when the loop is new, so it is only done once per loop. The loop's size is utilized to determine whether it is the same loop or not. This approach is chosen for its simplicity and the low likelihood of encountering two different loops with identical sizes. After this verification, an iteration within the loop is made, comparing these with the spotted addresses. If there is a match, that value is associated as a PPaddr.

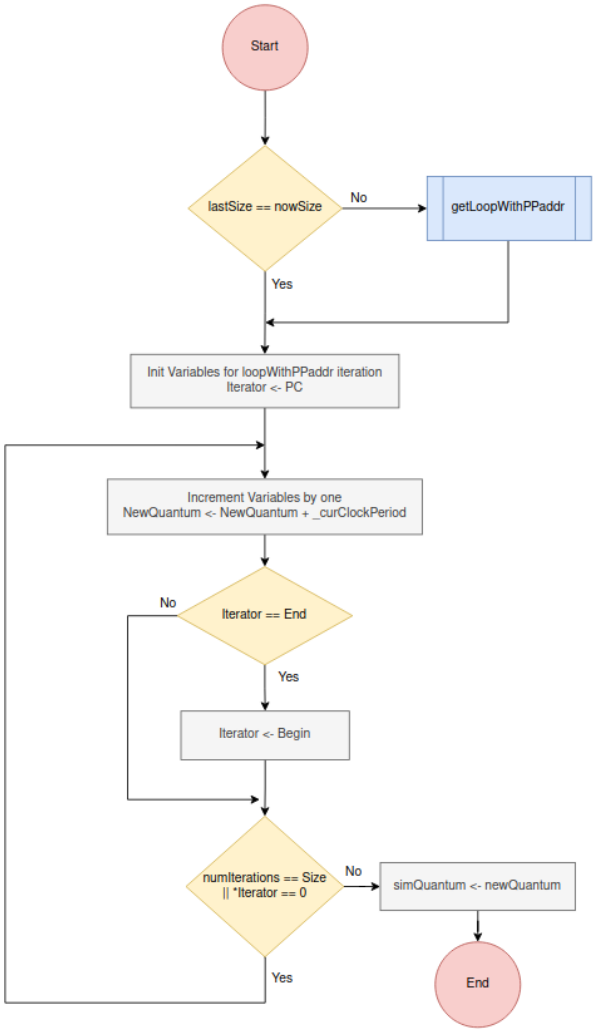
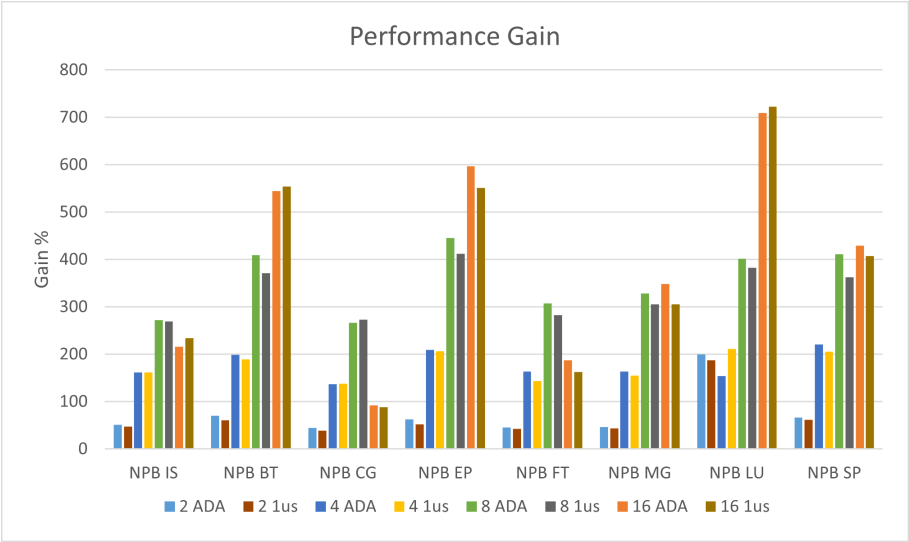


Figure 3.21: Quantum calculation flowchart

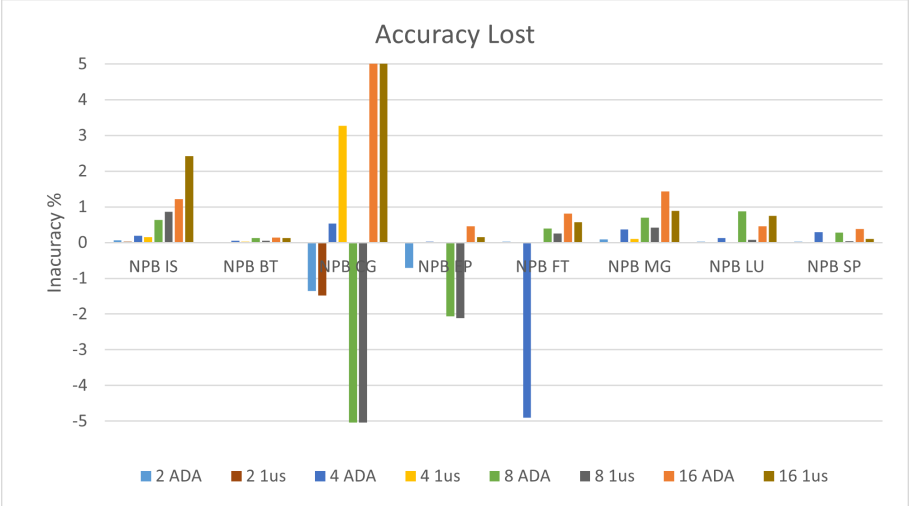
To calculate the quantum, the last tasks uses a iteration technique, as shown in detail in the Figure 3.21. The iterator starts where the PC is, and pursues the loop until it either finds a PPaddr, or reaches the starting point. The quantum starts with the minimum value, which is the clock period, and increments that value in each iteration done.

3.5.3 Results

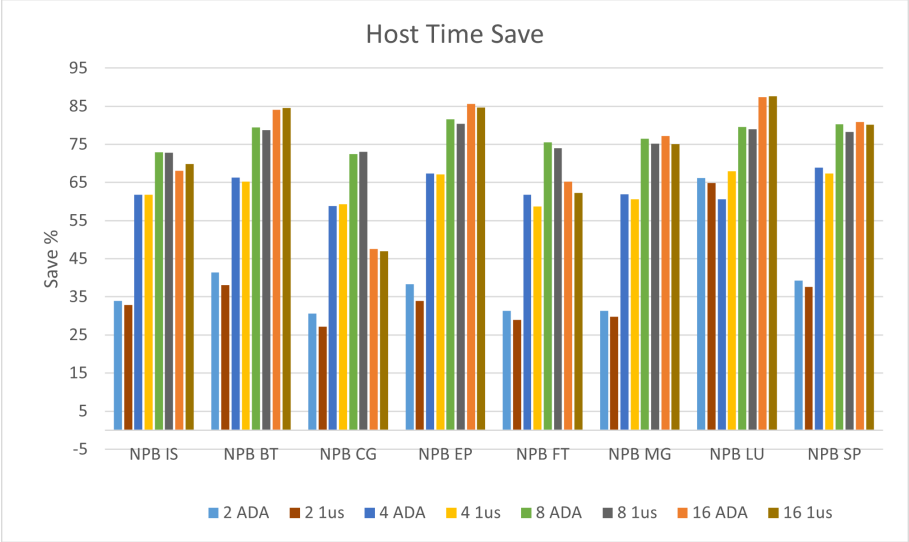
Figure 3.22 exhibits the results of the executed benchmarks with the previous (A-I-P) and new (REP) configurations.



(a) Performance gain



(b) Accuracy lost



(c) Host time save

Figure 3.22: Repetitions algorithm results

3.6 Final Algorithm

After the development and testing of all aforementioned algorithms, it was determined the better algorithm is the combination of the ADALINE, the dynamic increment, and the PC analyses. The results show they complement each other, providing a great trade-off between performance and accuracy. The repetition algorithm was not integrated into the final solution due to its weak performance and accuracy gain.

The synchronization process represented in the Figure 3.1 can now be redefined to integrate the dynamic approach, as illustrated in the following flowchart. The static version was not removed because if there is a-priory information about the benchmark, the best quantum can be calculated before simulating. One example is a network application, where the communications delays are well-defined. If the quantum is set with the smaller delay, a perfect accuracy can be achieved [56].

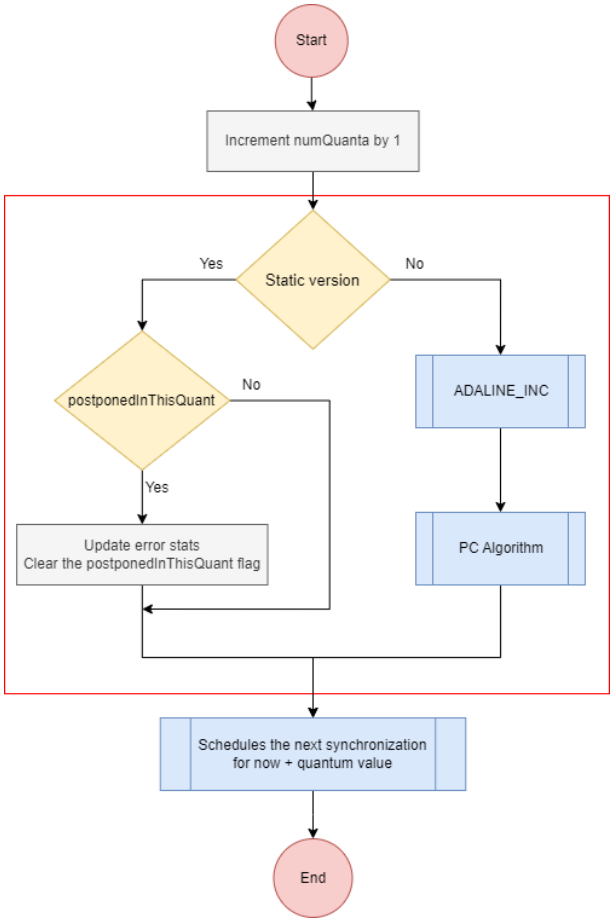


Figure 3.23: New quantum definition in the synchronization process

3.6.1 Results

4 Co-Simulation

This chapter presents the chosen case study, which serves as a stimulus for system design and demonstrates the practical application of the developed work. The chosen case study focuses on the Cyclic Redundancy Check (CRC) peripheral, which is present in several microcontroller families, such as STM32 [95] or Xilinx Zynq [96] nevertheless, the VExpress_gem5, which is a board introduced by Gem5, does not have it.

The CRC was created in 1961 by William Wesley Peterson [97]. As the name suggests, it utilizes systematic cyclic codes to encode messages by incorporating a fixed-length check value. In the end, his work contributed significantly to simplifying and enhancing the detection of accidental errors/changes in communication networks. CRC uses a generator polynomial, which is known by the sender and receiver, and it is used to perform the calculation. There are different standards however, the most common ones are the CRC-8, CRC-12, CRC-16, CRC-32, and CRC-CCIT [98]

Another application for the CRC is the storage integrity. Due to defective components or electromagnetic fields, bits can change their value without notice. In the presented case study, this scenario will be explored, where the CRC peripheral is used to maintain a specific memory state and verify if there have been any changes. Furthermore, to showcase the advantages of the previously developed work, the application will perform other tasks simultaneously.

4.1 CRC peripheral

As mentioned earlier in this dissertation, co-simulation involves the integration of multiple simulation tools from different domains. Because of that, it is required to develop a system that can execute and communicate among them. The following picture demonstrates how the different tools will communicate with each other.

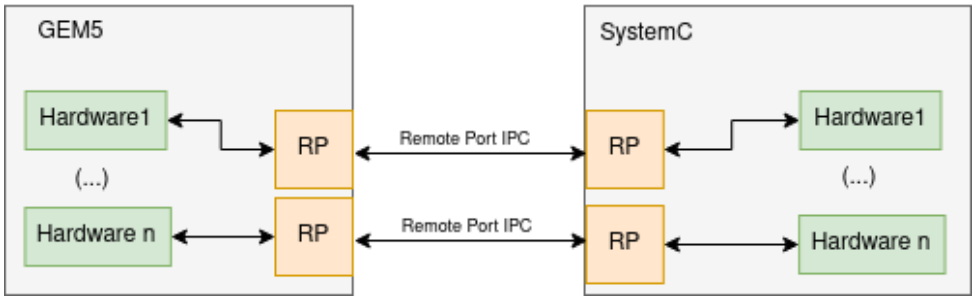
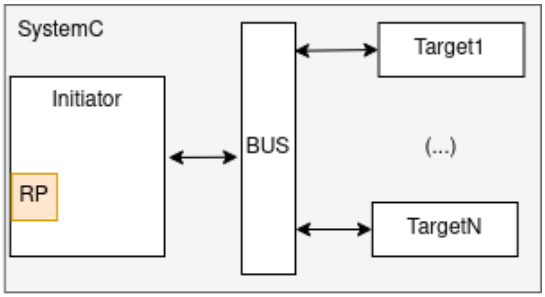


Figure 4.1: Co-simulation design

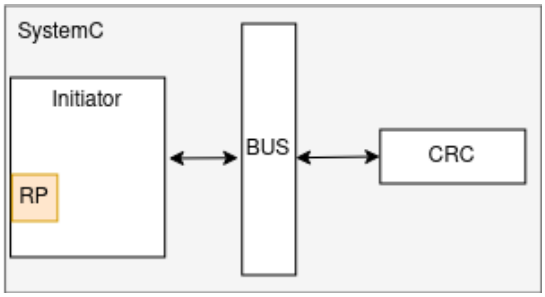
The communication will be done using a remote port IPC. This should be created and connected before the beginning of the simulation, in a way that transactions occur. Each hardware device will have a unique connection to avoid concurrency problems. It is important to remember that this hardware is not presented on the board, reason why is needed another simulator. The subsequent sections will delve into the detailed explanations of the CRC component. It's worth noting that the design process took significant reference from the STM32 microcontroller family's reference manual [95].

4.1.1 SystemC

The peripheral development was carried out using the SystemC tool. In this example, it will only simulate the CRC but, it has the capability to simulate more than just one peripheral. For instance, picture a scenario where the workload also requires an ADC. This device is also not included in the VExpress_gem5 board, so it would also require its implementation. For this reason, flexibility is a requirement, and with this idea in mind, the subsequent design was developed.



(a) General SystemC design



(b) SystemC design with CRC

It is composed of three main components. The initiator, the bus, and the targets. Beginning with the first, it serves as the initial point of interaction for the tool, as it handles incoming frames received through the remote port. The remote port operates independently of the initiator, enabling asynchronous reception and transmission of bytes through the receiver and transmit buffers, respectively. Additionally, it also performs an interpretation of the received trama and creates the TLM transaction. It analyses various parameters such as the operation type, the desired target, and the memory region, among others, with the help of TLM wrapper, which will presented in the subsection 4.1.3.

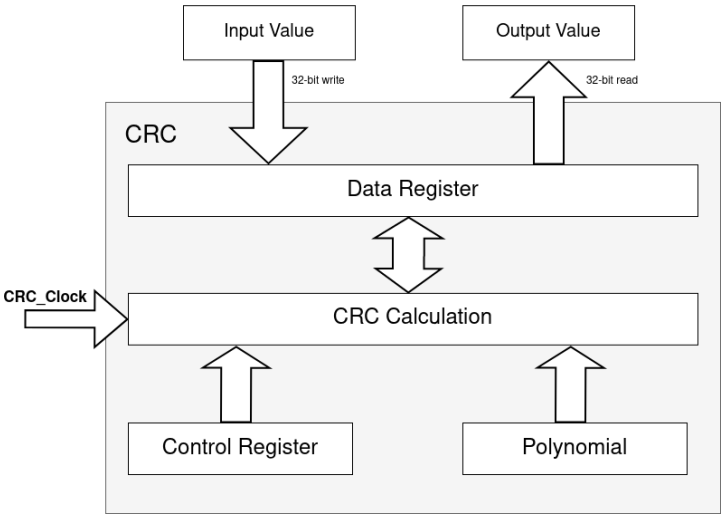


Figure 4.3: CRC block diagram

The bus is responsible for forwarding the TLM transaction to the right target. Each target has a unique ID, that is attributed to it at the beginning of the simulation. Gem5's board is a 32-bit processor, meaning that each transaction is 32-bit in length. However, SystemC TLM transactions can be 64-bit, leaving 32 bits unused. Before the transmission, the initiator uses these bytes to set the target_ID, which will be utilized by the bus to identify the targets.

Lastly, the targets are the peripherals, that can be different from each other, as mentioned in the previous example. These receive the TLM commands and act accordingly. Since in this dissertation, the focus will be the CRC, a single target will be implemented thus, the Figure 4.2a can be redefined to the Figure 4.2b. This will have the following characteristics [95].

- Uses CRC-32 (Ethernet) polynomial: 0x4C11DB7
- Programmable CRC initial value
- Single input/output 32-bit data register
- CRC computation done in 4 clock cycles
- General-purpose 8-bit register (can be used for temporary storage)
- Reversibility option on I/O data

The peripheral will work as demonstrated in Figure 4.3. First of all, the user needs to write the input value in the data register (CRC_DR). After four CRC clock cycles, the correspondent CRC value is fully calculated, and its value is stored in the CRC_DR. In this case, the CRC frequency with be equal to the CPU. The data width must be 32-bit, hence whether the user needs a CRC for five bytes, for example, two different computations will be needed.

Moreover, the input and output data can be reversed, to manage the various endianness schemes. For the input data, the reverse operation is controlled by the REV_IN[1:0] bits, and for the output data, the REV_OUT bit is used. These, along with the reset bit used to reset the CRC, are located in the control register (CRC_CR). An example of a reverse operation can be found on Table 4.1.

Table 4.1: Reverse operation

(a) Input

REV_IN[1:0]	Input	Reverse Action	Reverse Input
0 0	0x1A2B3C4D	Not affected	0x1A2B3C4D
0 1	0x1A2B3C4D	Bit-reversal done by byte	0x58D43CB2
1 0	0x1A2B3C4D	Bit-reversal done by half-word	0xD458B23C
1 1	0x1A2B3C4D	Bit-reversal done by word	0xB23CD458

(b) Output

REV_OUT	Output	Reverse Action	Reverse Output
0	0x11223344	Not affected	0x11223344
1	0x11223344	Bit-reversal done by word	0x22CC4488

By default, the polynomial coefficients are defined by 0x4C11DB7 nevertheless, it can be fully programmable through the CRC_POL register. It is important to mention that modifications in this register when a CRC computation is ongoing are not permitted, as it would compromise the output value. To conclude the available registers, are missing the CRC_INIT and CRC_IDR, which are used to initialize the CRC calculator in the reset, and to hold a temporary 8-bit value related to CRC calculation, respectively.

Summing up, the Figure 4.4 and Figure 4.5 demonstrate how the peripheral will behave depending on the defined settings and desired operation. Before any execution, offset/size out-of-bounds, and permissions are verified to maintain the device's integrity. The functionality of these parameters will be explained further.

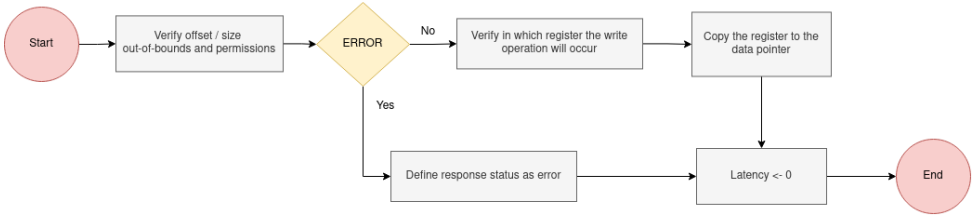


Figure 4.4: CRC read operation

4.1.2 Gem5

As aforementioned, Gem5 has available as a target platform the VExpress_gem5 board. It is based on the ARM Versatile Express RS1, which consists of a motherboard and one or more daughterboards. The system provides a range of both on-chip and off-chip devices. On-chip devices include the Generic Interrupt Controller (GIC), an LCD controller, and system timers. Off-chip devices contain the Keyboard and Mouse Interface (KMI), Real-Time Clock (RTC), and Universal Asynchronous Receiver-Transmitter (UART). The platform's memory map is divided into the next sections.

1. Boot memory .....0x00000000 to 0x03FFFFFF

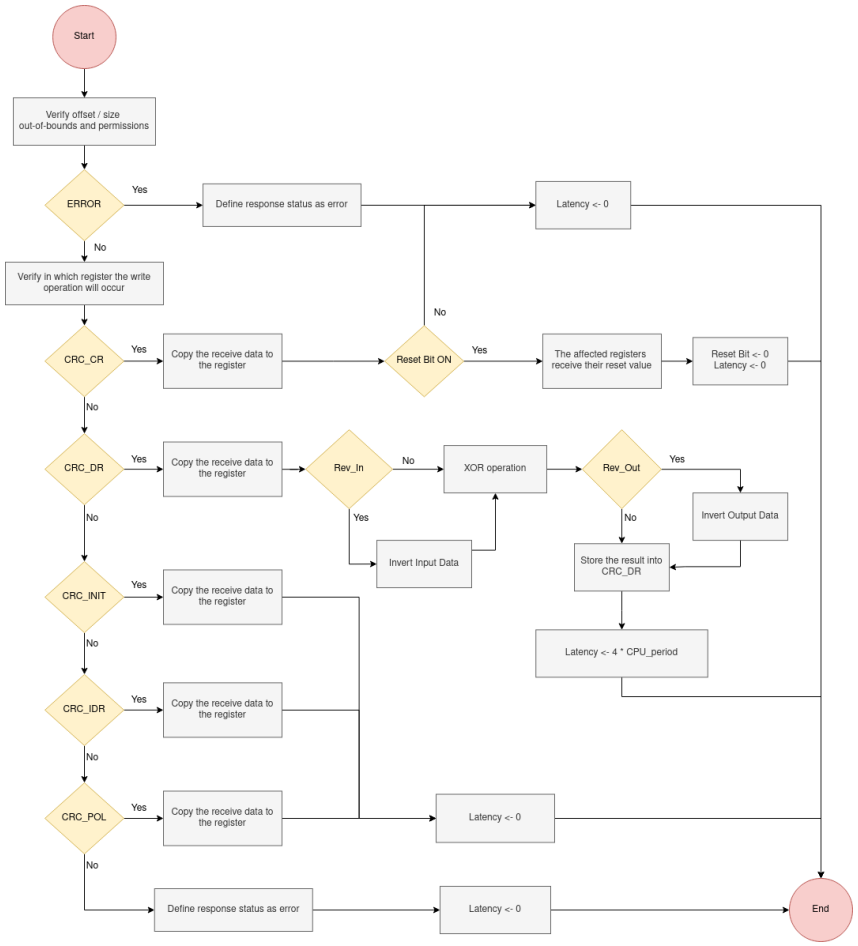


Figure 4.5: CRC write operation

2. Reserved .....0x04000000 to 0x0FFFFFFF
3. Off-chip peripherals .....0x10000000 to 0x1FFFFFFF
   (a) Gem5-specific peripherals .....0x10000000 to 0x13FFFFFF
      i. Energy controller .....0x10000000 to 0x1000FFFF
      ii. Pseudo-ops .....0x10010000 to 0x1001FFFF
      iii. MHU .....0x10020000 to 0x1002FFFF
   (b) Reserved .....0x14000000 to 0x17FFFFFF
   (c) VRAM .....0x18000000 to 0x19FFFFFF
   (d) Reserved .....0x1A000000 to 0x1BFFFFFF
   (e) Peripheral block 1 .....0x1C000000 to 0x1FFFFFFF
4. On-chip peripherals .....0x20000000 to 0x3FFFFFFF
5. PCI memory .....0x40000000 to 0x7FFFFFFF

6. DRAM .....0x80000000 to 0xFFFFFFFF

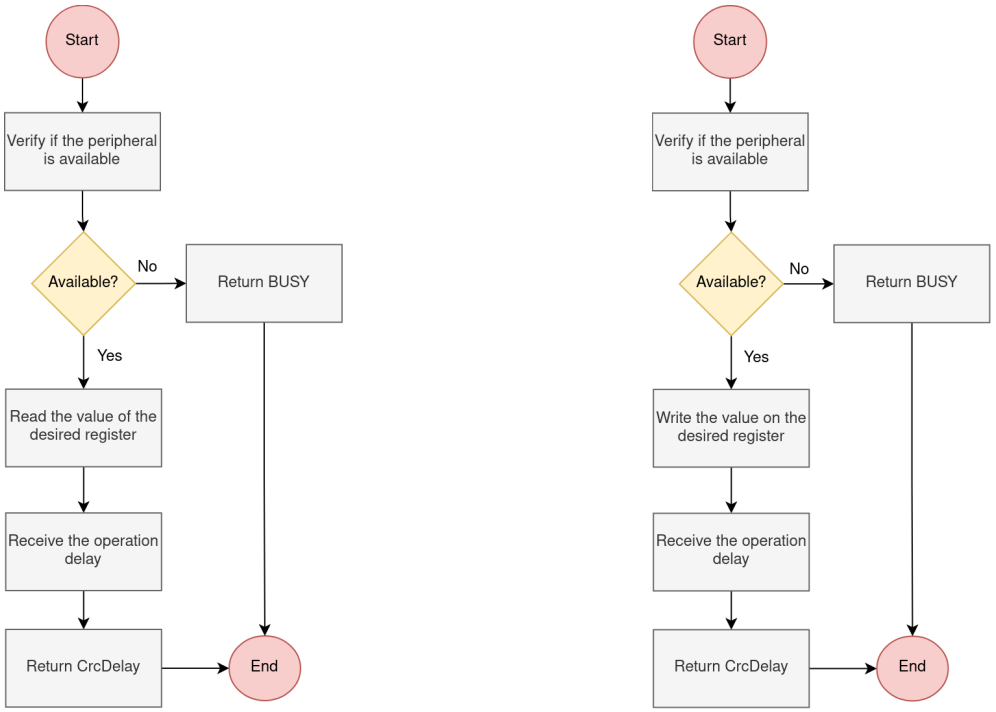
It is possible to notice that the Gem5-specific peripherals memory region is not fully occupied. Since the understudy device is not available on this board, its implementation will be done in this memory region. In accordance with the remaining devices, the memory from 0x10030000 to 0x1003FFFF will be reserved for SystemC CRC. Subsequently, it must be recognized as a Gem5 peripheral to enable communication with it. To achieve this, it should be defined and added to the list of off-chip devices in the board's configuration. In addition, it is also required to create a Page Table Entry (PTE) for the device, which can be done by following the code on 4.1.

```
LDR   r1,= DEVICE_ADDR   // CRC address -> 0x10030000
LSR   r1, r1, #20        // Find which 1MB block it is in
LSL   r2, r1, #2         // Gives offset into the page tables
LSL   r1, r1, #20        // Put back in address format
LDR   r3, =L1_DEVICE     // Descriptor template
ORR   r1, r1, r3         // Combine address and template
STR   r1, [r0, r2]       // Store table entry
```

Code 4.1: Template for a PTE

For this case of study, it will be needed two peripherals, the previously mentioned one and the UART. The UART will be used to communicate with the user by the terminal. As referred, this is already implemented, hence it only requires its initialization and configuration. However, it's important to note that even after this process, the CRC is only recognized as a device of the VExpress_gem5 board and its implementation still need to be completed.

To implement a device in Gem5, a few aspects are mandatory. First of all, the peripheral's Python interface. In this document, it is described the type, where implementation is, and, optionally, some parameters to be customized. In this context, these can be the port number, for the remote connection, or the action time delay. In second place, is the implementation itself. Gem5 has implemented a device template, that is present in the class *BasicPioDevice*. By inheriting this, the default configurations are placed nevertheless, read, and write functions still need to be defined, as these change from device to device. The figures 4.6a and 4.6b present its implementation. Further, other aspects must be present, like the CRC registers, the socket_ID, and the remote port functions(listen, accept, and detach). The remote connection is done before the simulation starts, meaning that it does not continue until a connection is made. As soon as it happens, an init function is called to state the initialization, with the device's information. It is required due to the possibility of various devices in SystemC. The last point is the *Sconscript* file, which is required by the compiler to understand the available SimObjects. Here should be discriminated the SimObject, that is, the Python file, the SimObject's source file, and the debug flags, if presented.



(a) CRC read flowchart **(b)** CRC write flowchart

Figure 4.6: Redefinition of *BasicPioDevice* functions

4.1.3 TLM wrapper

Although Gem5 and SystemC are two frameworks that use C++, they cannot communicate directly with each other. Gem5 is unable to interpret the TLM commands, just as SystemC doesn't understand read and write operations to the registers. For these reasons, it is necessary a wrapper to enable communication between the two platforms. This will be available in both, and every transaction must pass through it otherwise, the channel could be corrupted.

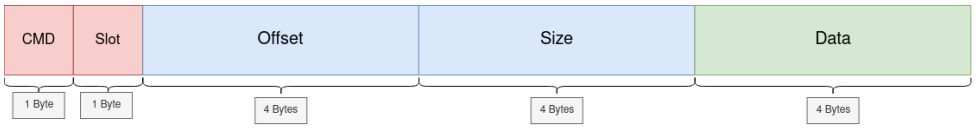


Figure 4.7: TLM wrapper payload definition

The main part of the wrapper is the payload definition. It provides coherent communication between platforms, keeps the transactions easy to understand, and improves efficiency. For this case, its definition can be found in the prior image. Each operation must have a command, a slot, reason why both have the color red. The remote port IPC reads and writes eight bits at a time. Consequently, even if the command list does not use all the bits, a byte is allocated for their size. The list of commands can be found on Table 4.2. The slot is used as an ID. Regarding the Figure 4.1, each peripheral has a remote connection associated hence, with this, is possible to decode the port where the transaction should be done. Offset and size are marked as blue since they are mandatory for both read and write operations. Data is with

green color because only is necessary for the write action. Four bytes are demanded for these parameters, due to the 32-bit processor present in the VExpress_gem5 board.

Table 4.2: TLM wrapper commands

Bits	Command
00	TLM_INIT
01	TLM_CLOSE
10	TLM_READ
11	TLM_WRITE

Every transaction is kept up with a response, TLM_ACK (0) for success, or TLM_NACK (1) for error. In the case of reading, is added the bytes of the selected memory region. The next figures demonstrate how the wrapper is implemented in both frameworks.

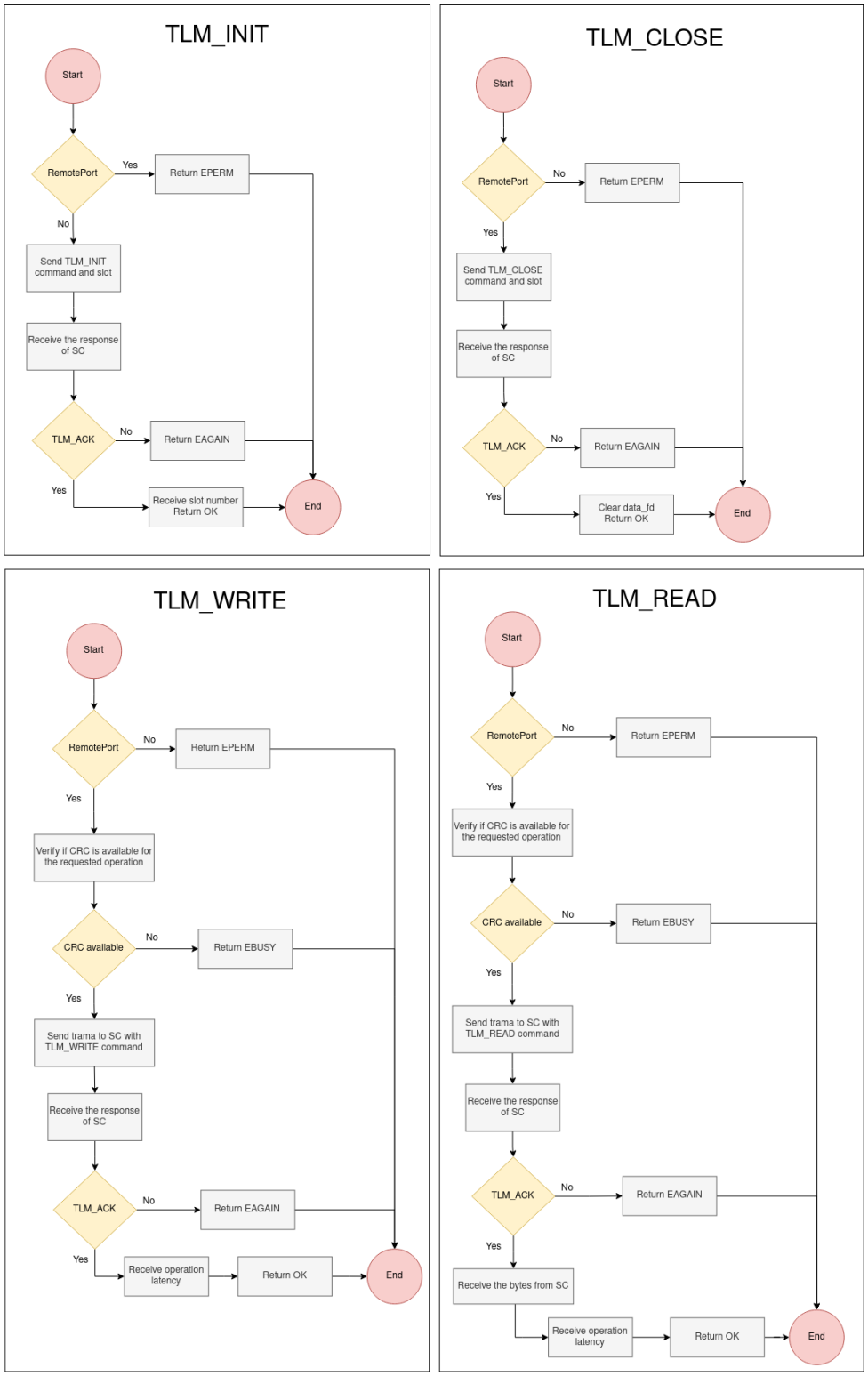


Figure 4.8: TLM wrapper in Gem5

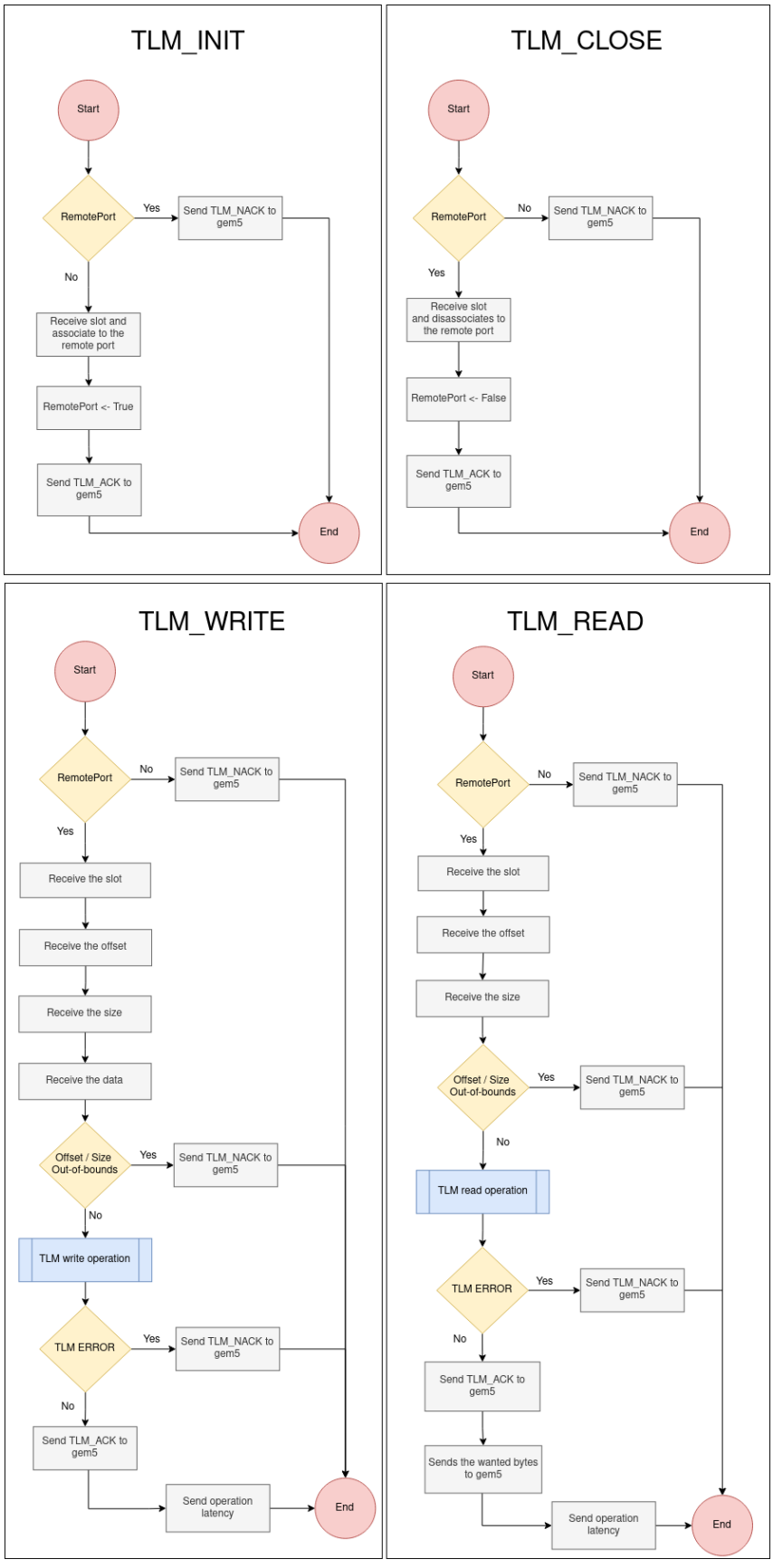


Figure 4.9: TLM wrapper in SystemC

4.1.4 Application interface

4.2 Application simulation using Gem5

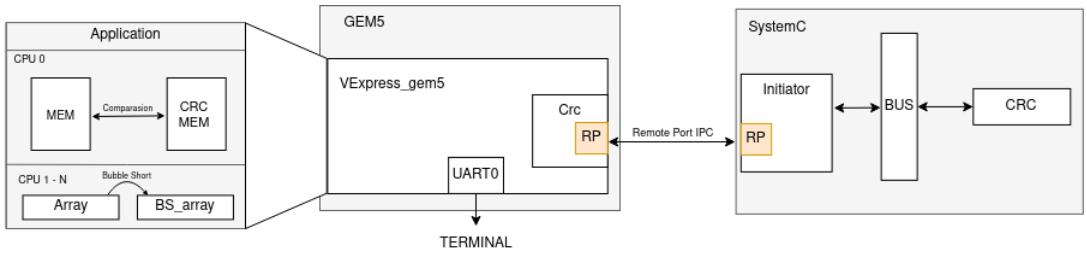


Figure 4.10: Co-simulation design

4.2.1 CRC peripheral validation

From the application point of view, the system will execute 2 distinct jobs. CPU0 will be responsible for performing the memory integrity checks, while the remaining ones will execute a bubble sort algorithm.

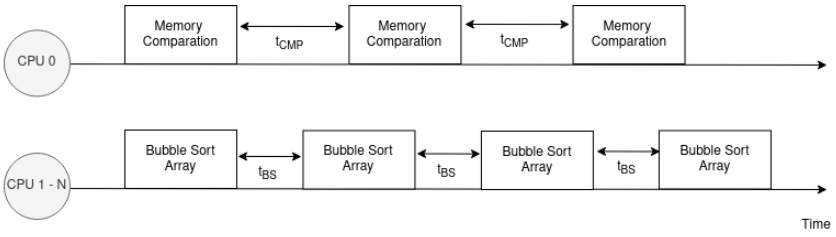


Figure 4.11: Application execution timely diagram

4.3 Memory integrity

4.3.1 Fault Modeling

4.4 Results

5 | Case Of Study

Final ADALINE tests running in Niko's computer

Wrinting also

Co-simulation topic is almost done, CRC pheripherial is done and functional

6 | Conclusions

7 | Future Work

References

[1] R. P. A. A. F. A. N. C. O. O. Sugiono Sugiono Andi S. Putra, "New concept of product design by involving emotional factors using eeg: A case study of xomputer mouse design," *ACTA NEUROPSYCHOLOGICA*, vol. 19, no. 1, pp. 63–80, 2021.

[2] D. N. J. A. K. P. K. S. P. I. S. A. S. V. Mani Azimi Naveen Cherukuri, "Tera-scale computing," *Intel Technology Journal*, vol. 11, no. 3, pp. 173–184, 2007.

[3] N. Zurstraßen, C.-C. Jose, J. M. Joseph, R. Leupers, X. Xinghua, and L. Yichao, "Par-gem5: Parallelizing gem5 s atomic mode," English, in *2023 Design, Automation and Test in Europe Conference (DATE)*, 2023.

[4] Z. M. Research, *Virtual training and simulation market size, growth, trends, share*.

[5] S. Robinson, *Simulation: The Practice of Model Development and Use*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004, ISBN: 0470847727.

[6] M. Verkuyl, N. Dubois, S. Goldsworthy, T. Merwin, T. Willet, and T. Job, *Virtual Simulation: An Educator's Toolkit*. 2022.

[7] J. Banks, "Introduction to simulation," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, 1999, pp. 7–13.

[8] J. Banks, J. Carson, B. Nelson, and D. Nicol, *Discrete-Event System Simulation*, English, 5th ed. Prentice Hall, 2010, ISBN: 0136062121.

[9] M. Barr, *Programming Embedded Systems in C and C++* (O'Reilly Series). O'Reilly, 1999, ISBN: 9781565923546.

[10] R. Camposano and J. Wilberg, "Embedded system design," *Design Automation for Embedded Systems*, vol. 1, pp. 5–50, 1996.

[11] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005, ISBN: 0596005903.

[12] X. Zhai, F. Bensaali, and K. McDonald-Maier, "Automatic number plate recognition on fpga," in *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, 2013, pp. 325–328. DOI: 10.1109/ICECS.2013.6815420.

[13] J. Cong and J. Peck, "On acceleration of the check tautology logic synthesis algorithm using an fpga-based reconfigurable coprocessor," in *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186)*, 1997, pp. 246–247. DOI: 10.1109/FPGA.1997.624629.

[14] J. Cong and J. Peck, "On acceleration of the check tautology logic synthesis algorithm using an fpga-based reconfigurable coprocessor," in *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186)*, 1997, pp. 246–247. DOI: 10.1109/FPGA.1997.624629.

[15] M. Silva, D. Cerdeira, S. Pinto, and T. Gomes, "Operating systems for internet of things low-end devices: Analysis and benchmarking," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 375–10 383, 2019.

[16] IEC, *Iec, 192-01-22 dependability*.

[17] A. Tanenbaum and H. Bos, *Modern Operating Systems, Global Edition*. Pearson Education, 2015, ISBN: 9781292061955.

[18] R. Barry, *Mastering the freertos real time kernel. real time engineers ltd*, 2016.

[19] A. A. Adenowo and B. A. Adenowo, "Software engineering methodologies: A review of the waterfall model and object-oriented approach," *International Journal of Scientific & Engineering Research*, vol. 4, no. 7, pp. 427–434, 2013.

[20] W Royce, "Winston," *Proceedings, Managing the Development of Large Software Systems, IEEE WESCON*, 1970.

[21] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc," *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, 2012.

[22] G. B. Regulwar, P. Deshmukh, R. Tugnayat, P. Jawandhiya, and V. Gulhane, "Variations in v model for software development," *International Journal of Advanced Research in Computer Science*, vol. 1, no. 2, pp. 134–135, 2010.

[23] S. Mathur and S. Malik, "Advancements in the v-model," *International Journal of Computer Applications*, vol. 1, no. 12, pp. 29–34, 2010.

[24] W. Van Casteren, "The waterfall model and the agile methodologies: A comparison by project characteristics," *Research Gate*, vol. 2, pp. 1–6, 2017.

[25] B. Boehm, "A survey of agile development methodologies," *Laurie Williams*, vol. 45, p. 119, 2007.

[26] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[27] A. Alshamrani and A. Bahattab, "A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model," *International Journal of Computer Science Issues (IJCSI)*, vol. 12, no. 1, p. 106, 2015.

[28] B. Boehm, "A spiral model of software development and enhancement," *ACM SIGSOFT Software engineering notes*, vol. 11, no. 4, pp. 14–24, 1986.

[29] B. Liu, H. Zhang, and S. Zhu, "An incremental v-model process for automotive development," in *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, 2016, pp. 225–232.

[30] W. M. Zabołotny, "Development of embedded pc and fpga based systems with virtual hardware," in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2012*, SPIE, vol. 8454, 2012, pp. 259–265.

[31] *Ghdl main/home page*.

[32] S. Williams and M. Baxter, "Icarus verilog: Open-source verilog more than a year later," *Linux Journal*, vol. 2002, no. 99, p. 3, 2002.

[33] F. Bellard, "Qemu, a fast and portable dynamic translator.," in *USENIX annual technical conference, FREENIX Track*, Califor-nia, USA, vol. 41, 2005, p. 46.

[34] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011, ISSN: 0163-5964. DOI: 10.1145/2024716.2024718.

[35] O. Bringmann *et al.*, "The next generation of virtual prototyping: Ultra-fast yet accurate simulation of hw/sw systems," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2015, pp. 1698–1707.

[36] L. Jünger, C. Bianco, K. Niederholtmeyer, D. Petras, and R. Leupers, "Optimizing temporal decoupling using event relevance," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 331–337.

[37] A. Varga, "Discrete event simulation system," in *Proc. of the European Simulation Multiconference (ESM'2001)*, 2001, pp. 1–7.

[38] E. Babulak and M. Wang, "Discrete event simulation," *Aitor Goti (Hg.): Discrete Event Simulations. Rijeka, Kroatien: Sciyo*, p. 1, 2010.

[39] W Boughton and O Droop, "Continuous simulation for design flood estimation—a review," *Environmental Modelling & Software*, vol. 18, no. 4, pp. 309–318, 2003.

[40] F. Henning, L. Kärger, D. Dörr, F. J. Schirmaier, J. Seuffert, and A. Bernath, "Fast processing and continuous simulation of automotive structural composite components," *Composites Science and Technology*, vol. 171, pp. 261–279, 2019.

[41] M. Helal, *A hybrid system dynamics-discrete event simulation approach to simulating the manufacturing enterprise*. University of Central Florida, 2008.

[42] T. Q. P. Developers, *Qemu's documentation*.

[43] C. Schumacher, R. Leupers, D. Petras, and A. Hoffmann, "Parsc: Synchronous parallel systemc simulation on multi-core host architectures," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis*, 2010, pp. 241–246.

[44] "Ieee standard for standard systemc language reference manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, 2012. DOI: 10.1109/IEEESTD.2012.6134619.

[45] A. Yoga and S. Nagarakatte, "Parallelism-centric what-if and differential analyses," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 485–501.

[46] T. Zhou, *Sequential and parallel discrete event simulation on computer communication networks*. Western Michigan University, 1992.

[47] N. R. Tallent, L. Adhianto, and J. M. Mellor-Crummey, "Scalable identification of load imbalance in parallel executions using call path profiles," in *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11. DOI: 10.1109/SC.2010.47.

[48] L. Rose, B. Homer, and D. Johnson, "Detecting application load imbalance on high end massively parallel systems," Aug. 2007, pp. 150–159, ISBN: 978-3-540-74465-8. DOI: 10.1007/978-3-540-74466-5_17.

[49] K.-Y. Chen, J. M. Chang, and T.-W. Hou, "Multithreading in java: Performance and scalability on multicore systems," *IEEE Transactions on Computers*, vol. 60, no. 11, pp. 1521–1534, 2010.

[50] K. Du Bois, J. B. Sartor, S. Eyerman, and L. Eeckhout, "Bottle graphs: Visualizing scalability bottlenecks in multi-threaded applications," *ACM SIGPLAN Notices*, vol. 48, no. 10, pp. 355–372, 2013.

[51] S. Eyerman, K. Du Bois, and L. Eeckhout, "Speedup stacks: Identifying scaling bottlenecks in multi-threaded applications," Apr. 2012. DOI: 10.1109/ISPASS.2012.6189221.

[52] R. Ciesla, "Bits, sample rates, and other fundamentals of digital audio," in *Sound and Music for Games: The Basics of Digital Audio for Video Games*, Springer, 2022, pp. 1–24.

[53] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[54] G. Busnot, T. Sassolas, N. Ventroux, and M. Moy, "Standard-compliant parallel systemc simulation of loosely-timed transaction level models," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2020, pp. 363–368.

[55] M. Jung, F. Schnicke, M. Damm, T. Kuhn, and N. Wehn, "Speculative temporal decoupling using fork ()," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 1721–1726.

[56] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, "Dist-gem5: Distributed simulation of computer clusters," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2017, pp. 153–162.

[57] J. H. Weinstock, R. Leupers, G. Ascheid, D. Petras, and A. Hoffmann, "Systemc-link: Parallel systemc simulation using time-decoupled segments," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2016, pp. 493–498.

[58] J. Lowe-Power *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[59] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006. DOI: 10.1109/MM.2006.82.

[60] M. M. K. Martin *et al.*, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," vol. 33, no. 4, pp. 92–99, 2005, ISSN: 0163-5964. DOI: 10.1145/1105734.1105747.

[61] D.-I. G. Hempel and I. J. Castrillon, "Simulation of risc-v based systems in gem5,"

[62] W. Jakob, J. Rhinelander, and D. Moldovan, "Pybind11–seamless operability between c++ 11 and python," *URL https://github.com/pybind/pybind11*, 2022.

[63] S. Knight, "Scons design and implementation," in *Tenth int'l python conf*, 2002.

[64] N. Zurstraßen, R. Brandhofer, J. C. Cascante, J. M. Joseph, N. Bosbach, and R. Leupers, *Beyond the Quantum of Temporally-Decoupled Simulations*,

[65] G. Glaser, G. Nitsche, and E. Hennig, "Temporal decoupling with error-bounded predictive quantum control," in *2015 Forum on Specification and Design Languages (FDL)*, 2015, pp. 1–6.

[66] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985.

[67] F. Morales and J. L. Bismarck, *Evaluating gem5 and qemu virtual platforms for arm multicore architectures*, 2016.

[68] A. Herrera, *"running trusted firmware-a on gem5*, 2020.

[69] T. Wieman, B. Bhattacharya, T. Jeremiassen, C. Schroder, and B. Vanthournout, "An overview of open systemc initiative standards development," *IEEE Design & Test of Computers*, vol. 29, no. 2, pp. 14–22, 2012.

[70] A. Akram and L. Sawalha, "A comparison of x86 computer architecture simulators," 2016.

[71] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2sim: A simulation framework for cpu-gpu computing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, 2012, pp. 335–344.

[72] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[73] M. T. Yourst, "Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator," in *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, IEEE, 2007, pp. 23–34.

[74] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

[75] A. Patel, F. Afram, and K. Ghose, "Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors," in *1st International Qemu Users' Forum*, Citeseer, 2011, pp. 29–30.

[76] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: Insights of machine learning and deep learning algorithms," *Machine Learning with Applications*, vol. 6, p. 100 164, 2021.

[77] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[78] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[79] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.

[80] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Univ Ca Stanford Electronics Labs, Tech. Rep., 1960.

[81] B. Widrow and S. D. Stearns, "Adaptive signal processing prentice-hall," *Englewood Cliffs, NJ*, p. 52, 1985.

[82] B. Widrow and M. A. Lehr, "Perceptrons, adalines, and backpropagation," *Arbib*, vol. 4, pp. 719–724, 1995.

[83] K. Mitchell-Wallace, M. Jones, J. Hillier, and M. Foote, *Natural catastrophe risk management and modelling: A practitioner's guide*. John Wiley & Sons, 2017.

[84] L. von Rueden, S. Mayer, R. Sifa, C. Bauckhage, and J. Garcke, "Combining machine learning and simulation to a hybrid modelling approach: Current and future directions," in *Advances in Intelligent Data Analysis XVIII: 18th International Symposium on Intelligent Data Analysis, IDA 2020, Konstanz, Germany, April 27–29, 2020, Proceedings 18*, Springer, 2020, pp. 548–560.

[85] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM review*, vol. 57, no. 4, pp. 483–531, 2015.

[86] E. Tsymbalov, S. Makarychev, A. Shapeev, and M. Panov, "Deeper connections between neural networks and gaussian processes speed-up active learning," *arXiv preprint arXiv:1902.10350*, 2019.

[87] F. Noé, A. Tkatchenko, K.-R. Müller, and C. Clementi, "Machine learning for molecular simulation," *Annual review of physical chemistry*, vol. 71, pp. 361–390, 2020.

[88] K. Albertsson *et al.*, "Machine learning in high energy physics community white paper," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1085, 2018, p. 022 008.

[89] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: State of the art," *arXiv preprint arXiv:1702.00686*, 2017.

[90] Xilinx, *Xilinx/libsystemctlm-soc: Systemc/tlm-2.0 co-simulation framework*.

[91] C. Menard, J. Castrillon, M. Jung, and N. Wehn, "System simulation with gem5 and systemc: The keystone for full interoperability," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, IEEE, 2017, pp. 62–69.

[92] D. Bailey *et al.*, "The nas parallel benchmarks rnr-94-007," *NASA Advanced Supercomputing Division, Tech. Rep*, 1994.

[93] S. Haykin, "Linear prediction," *Adaptive filter theory*, pp. 562–588, 1996.

[94] M. Stella, D. Begusic, and M. Russo, "Adaptive noise cancellation based on neural network," in *2006 International Conference on Software in Telecommunications and Computer Networks*, 2006, pp. 306–309. DOI: 10.1109/SOFTCOM.2006.329765.

[95] *Rm0385 reference manual -based 32-bit mcus*.

[96] G Xilinx and S Guide, "Zynq-7000 all programmable soc technical reference manual (ug585)," Tech. rep., Xilinx, 2014, https://www. xilinx. com/support/documentation ..., Tech. Rep., 2014.

[97] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proceedings of the IRE*, vol. 49, no. 1, pp. 228–235, 1961.

[98] C. Borrelli, "Ieee 802.3 cyclic redundancy check," *application note: Virtex Series and Virtex-II Family, XAPP209 (v1. 0)*, 2001.