UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Iago de Oliveira Silvestre

USING GEM5 SIMULATOR TO SUPPORT DESIGN SPACE EXPLORATION TARGETING ARM ARCHITECTURE

Florianópolis
2022

Iago de Oliveira Silvestre

USING GEM5 SIMULATOR TO SUPPORT DESIGN SPACE EXPLORATION TARGETING ARM ARCHITECTURE

Dissertação de Mestrado submetido ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do título de Mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Leandro Buss Becker, Dr. Eng.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Silvestre, Iago
USING GEM5 SIMULATOR TO SUPPORT DESIGN SPACE EXPLORATION TARGETING ARM ARCHITECTURE / Iago Silvestre ; orientador, Leandro Buss Becker, 2022.
52 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Sistemas Embarcados Computacionais. 3. Simulação de Sistema Completo. 4. Simulação gem5. 5. Veículos Aéreos não Tripulados. I. Buss Becker, Leandro. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Iago de Oliveira Silvestre

USING GEM5 SIMULATOR TO SUPPORT DESIGN SPACE EXPLORATION TARGETING ARM ARCHITECTURE

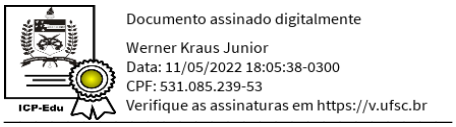
O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Antonio Carlos Beck Filho, Dr.
Instituição INF/UFRGS

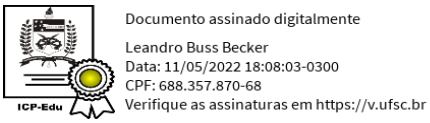
Prof. Giovani Gracioli, Dr.
Instituição PPGCC/UFSC

Prof. Carlos Barros Montez, Dr.
Instituição DAS/UFSC

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia de Automação e Sistemas.



Coordenação do Programa de Pós-Graduação



Prof. Leandro Buss Becker, Dr.
Orientador

Florianópolis, 2022.

To my family and friends

ACKNOWLEDGMENTS

I would like to acknowledge the Federal University of Santa Catarina and its faculty, who gave me the opportunity to study and work in my current research.

I am very grateful for all the support I had from colleagues and members of the ProVANT project, specifically from my advisor Leandro Buss Becker, who motivated and made several collaborations on my study.

I would also like to thank my parents, who supported me along the way.

Also my grandmother, for her shelter and care.

Finally I would like to thank my brother, who is always by my side and motivates me in the good and bad moments.

> "Coming back to where you started is not the same as never leaving. "
>
> (Terry Pratchett)

RESUMO

A análise de performance em sistemas embarcados é vital quando se trata de sistemas Ciber-físicos que tem como requisitos garantias de estabilidade. Esses sistemas geralmente operam tendo que respeitar limitações temporais impostas durante o design do sistema de controle responsável. Até recentemente, essa análise de performance era feita exclusivamente através da execução do código na plataforma de destino e fazendo medidas em cima desses testes. Porém essa análise de execução de código também pode ser feita através de ferramentas de simulação, que oferecem maior flexibilidade para os desenvolvedores configurarem a plataforma de acordo com os testes que se tem interesse. Apesar desse maior grau de liberdade, normalmente há preocupações se os dados fornecidos por esses ferramentas de simulação são precisos suficiente para serem usados para guiar a Exploração de Espaço de Design durante o desenvolvimento de um projeto. O objetivo principal dessa dissertação de mestrado é analisar a precisão da ferramenta gem5 quando utilizada com algoritmos de controle utilizados no projeto ProVANT, além de demonstrar como os dados fornecidos por essa ferramenta podem ser utilizados para guiar fases de desenvolvimento de um projeto, mais especificamente a escolha das configurações computacionais do sistema. Para fazer esse estudo foi utilizada a Raspberry Pi 3 B+ (com um processador Cortex-A53) como plataforma de referência, servindo de base para criar diferentes versões da simulação para análise. Os resultados obtidos demonstraram que o gem5 teve boa precisão no quesito temporal quando comparamos os resultados das simulações com testes em hardware físico, porém demonstraram variação significativa em dados relacionados aos acessos do ultimo nível de memória cache, que no caso da Raspberry é o nível 2. Além disso, durante a elaboração desses estudos, foi criada uma implementação do processo de simulação usando o gem5 através de computação em nuvem, usando a plataforma Google Colab. Essa implementação permite que usuários que não conhecem a ferramenta gem5 consigam fazer simulações em plataforma ARM de maneira simples, sem ter que aprender a ferramenta e removendo o tempo de *setup* para iniciar as simulações.

Palavras-chave: Simulação de Sistema Completo, gem5, Sistemas Computacionais Embarcados, VANT, DSE.

RESUMO EXPANDIDO

INTRODUÇÃO

Sistemas Embarcados tipicamente combinam um ou mais processadores, memórias e dispositivos de entrada/saída. Esses sistemas são comumente utilizados no contexto de Sistemas Ciber-físicos onde o software rodando no sistema embarcado controla o movimento do sistema mecânico acoplado. Tendo isso em vista, boa parte desses sistemas são aplicações de tempo real, já que precisam respeitar restrições temporais (*deadlines*) para assegurar a estabilidade do sistema (AMINIFAR, 2016).

Fica claro então a importância da análise dos algoritmos de controle utilizados em sistemas embarcado. Essa análise é valiosa para os desenvolvedores desse sistema já que permite a observação da performance temporal dos algoritmos e dos aspectos arquiteturais da plataforma que influenciam essa performance. Isso nos permite: (i) testar se o programa de controle consegue ser executado dentro da janela projetada (*deadline*) ; (ii) se não consegue respeitar as restrições temporais, ajuda a detectar onde os *bottlenecks* do sistema estão localizados. Pela perspectiva da plataforma embarcada, essa análise fornece um meio para o arquiteto do sistema conseguir selecionar a configuração de hardware mais apropriada. Além de permitir a iteração para explorar modificações em software e hardware (TASHIRO; OYAMADA, 2016a) que devem ser feitas para que os *deadlines* sejam respeitados e a estabilidade do sistema não fique comprometida (LATHI, 2005).

Dependendo do nível de detalhe do processo de simulação, a ferramenta pode ser utilizada para ajudar na Exploração de Espaço de Design do projeto (TASHIRO; OYAMADA, 2016b). Porém, apesar dos benefícios que essas ferramentas oferecem, um problema comum é a barreira inicial de aprender as peculiaridades de dada ferramenta e o tempo para estabelecer um processo de simulação ágil com reprodutibilidade. Por isso foi desenvolvida uma implementação do processo de simulação utilizando o gem5 através da ferramenta de computação em nuvem Google Colab, disponível em (SILVESTRE, 2021b). Por enquanto a implementação permite a simulação de programas de controle na arquitetura ARM, sendo que esses programas podem ser compilados com a biblioteca m5ops do gem5 para fornecer dados de regiões especificas do código sendo analisado. Para ajudar os usuários que não conhecem a ferramenta gem5 ou o ambiente do Colab foi feito um video guia (SILVESTRE, 2021a) que demonstra a utilização básica da plataforma.

Um ponto de discussão recorrente na literatura de sistemas embarcados é o problema de determinar WCET(*Worst Case Execution Time*) quando se utiliza a memória cache (LESAGE et al., 2010). De maneira geral a memória cache introduz problemas quando se deseja alcançar determinismo no nível de performance, por diversos motivos, como por exemplo a falta de controle dos dados que são armazenados nesta memória. Entretanto, não utilizar a memória cache pode ser inviável para programas de complexidade mais alta, neste caso algumas alternativas que aumentam a confiabilidade do tempo de execução usando

a memória cache podem ser tomadas. Por exemplo pode ser feito o particionamento da memória cache (BAI et al., 2016), outro caminho é a utilização de técnicas de escalonamento preemptivo (DESTELLE; DUFOUR, 2010). Uma alternativa que pode ser aplicada mais facilmente é o *Software Prefetching* que consiste em adaptar o programa com funções que instruem o processador a resgatar e inserir dados na memória cache logo antes que ela seja necessária, evitando tempos de espera em situações onde o dado não está na cache.

OBJETIVOS

O objetivo geral desta dissertação de mestrado é demonstrar se e como a ferramenta gem5 pode ser utilizada para guiar a Exploração de Espaço de Design de um projeto de um sistema de tempo real em sistemas embarcados.

Objetivos específicos

Aliados ao escopo principal, foram elencados alguns objetivos específicos:

- Verificar se a precisão da ferramenta gem5 no modo *Full System* é suficiente;
- Demonstrar como a ferramenta pode ser utilizada para guiar escolhas de configuração de hardware, como por exemplo CPU, utilizando os dados extraídos da simulação;
- Analisar impacto da memória cache nos algoritmos de controle e verificar alternativas para melhorar determinismo utilizando memória cache;
- Fornecer uma alternativa do processo de simulação usando gem5 utilizando computação em nuvem, tornando o acesso a ferramenta mais abrangente e simplificado.

METODOLOGIA

Como metodologia, foi estabelecida uma abordagem teórica/experimental, com uma etapa de pesquisa , seguida de desenvolvimento experimental, análise de dados e validação de resultados.

A etapa de pesquisa visou aprofundar a análise de performance através dos dados gerados pela ferramenta gem5 e procurar outros métodos de analisar a performance através de testes em hardware para poder fazer mais testes comparativos entre os dois.

O desenvolvimento experimental utilizou dos métodos adicionais encontrados na pesquisa para realizar os testes desejados.

A análise dos dados consistiu do estudo dos dados extraídos por testes em software e hardware. A comparação desses dados e o estudo do que eles representam para o arquiteto de um sistema embarcado é importante, ou seja como esses dados podem ajudar a guiar as escolhas da plataforma que irá executar o programa de controle.

Por fim, a validação dos resultados consistiu na busca e comparação com resultados já existentes em outros trabalhos.

RESULTADOS E DISCUSSÃO

A ferramenta da simulação gem5 se demonstrou capaz de avaliar e determinar características dos programas de controle que foram testados, especialmente na análise do comportamento temporal com múltiplas configurações de hardware. É uma ferramenta que pode ser utilizada para diversos fins, um muito notável é o de testar como diferentes configurações de hardware podem impactar a performance de um programa sobre teste, com intuito de possibilitar que os desenvolvedores façam ajustes para alcançar o nível de performance idealizado.

Em relação a implementação do Google Colab, ela fornece uma maneira mais simples para novos usuários executarem simulações, além de poder ser feita de maneira independente das configurações de hardware e Sistema Operacional instalado na máquina do usuário, requirindo apenas um *Web-browser*.

Os resultados obtidos demonstraram que o gem5 foi capaz de avaliar alternativas de design para projetos que utilizam a Arquitetura ARM. A grande maioria dos resultados obtidos em simulação apresentaram bons indicativos da performance do programa sendo testado, o que certamente é útil durante muitas fases da vida de um projeto, principalmente durante fases iniciais, onde esses dados podem ser utilizados para ajustar as configurações de hardware para atingir uma performance adequada.

CONSIDERAÇÕES FINAIS

As ferramentas de Simulação de Sistema Completo são muito importantes para possibilitar o projeto de Sistemas Embarcados de uma forma mais economicamente viável e com menos desperdício durante fases iniciais. Os resultados obtidos nessa dissertação de mestrado de demonstraram promissores em relação a utilização da ferramenta gem5 para análise de performance em projetos na arquitetura ARM. Na conclusão da dissertação, foram elencadas possíveis continuações para a pesquisa, incluindo a avaliação da utilização do gem5 em outras arquiteturas, como o x86, utilização de dados de sensores obtidos através de simulação do ambiente em Gazebo para aumentar a cobertura do programa de controle, entre outras.

ABSTRACT

Making the performance analysis of embedded systems is very important when dealing with Cyber-Physical Systems that require stability guarantees. They typically operate having to respect temporal constraints imposed during the design of the related control system. Until recently, performance analysis was done exclusively by executing the code on the target platform and making measures. Usually code execution/measuring can also be done on simulation software, which offers greater degree of flexibility for designers to configure the platform accordingly for the desired tests. However, there are doubts whether the results from such simulation software are reliable enough to guide Design alternatives during a project development. The main focus of this Master Thesis is to analyse the precision of the gem5 simulation tool for simulating control algorithms. It is envisioned that obtained simulation results can help designers selecting the embedded computing architecture. It is used as case study the control algorithms developed within the ProVANT project, which serve for controlling an Unmanned Aerial Vehicle (UAV). Such analysis is important given the following reasons: (i) allows to better understand the timing behavior of the algorithms; (ii) allows evaluating architectural issues related with the embedded platform, and (iii) allows determining how accurate the simulation software is. The development board Raspberry Pi 3 Model B+ (with Cortex-A53 processor) is used as reference platform and serves as basis for creating different simulated versions for the analysis. Obtained results showed that gem5 has enough timing precision when compared with results from the real hardware, at least with respect to execution time calculation, proving to be a useful tool to explore potential hardware and software alternatives when dealing with a project using the ARM architecture. Tests comparing real hardware with the gem5 simulation did show however some mismatches on data related to lowest-level cache accesses, which in the Raspberry Pi 3 B+ is level 2. An additional contribution of this work is the creation of a cloud computing environment at Google Colab, making significantly easier the use of gem5 simulator, even allowing its usage by non-experts.

Keywords:full-system simulation, gem5, embedded computing systems, UAV, DSE.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ALU	Arithmetic Logic Unit
CPU	Central Processing Unit
CISC	Complex Instruction Set Computer
CPS	Cyber-Physical System
DCache	Data Cache
DSE	Design Space Exploration
DDR	Double Data Rate
DRAM	Dynamic Random Acess Memory
UFMG	Federal University of Minas Gerais
UFSC	Federal University of Santa Catarina
FS	Full System
HDL	Hardware Description Language
HIL	Hardware-in-the-Loop
HPI	High Performance In-Order
ICache	Instruction Cache
ISA	Instruction Set Architecture
L1	Level 1 Cache
L2	Level 2 Cache
L3	Level 3 Cache
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
OS	Operating System
ProVANT	*Projeto de Veiculo Aereo Nao Tripulado*

RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROS	Robot Operating System
SSH	Secure Shell Protocol
SoC	System on Chip
UAV	Unmanned Aerial Vehicle
VHDL	VHSIC Hardware Description Language
VTOL	Vertical Take-Off and Landing
WCET	Worst-Case Execution Time

SYMBOLS

∞	Infinite
sup	Supremum
$\|\|\ \|\|$	Norm (Mathematics)
T	Period
A^T	Transpose Matrix of A
A^{-1}	Inverse Matrix of A
s	Complex plane of Laplace Transform
z	Complex plane of z-Transform
e	Euler
B	Bytes
KB	KiloByte
MB	MegaByte
GB	GigaByte

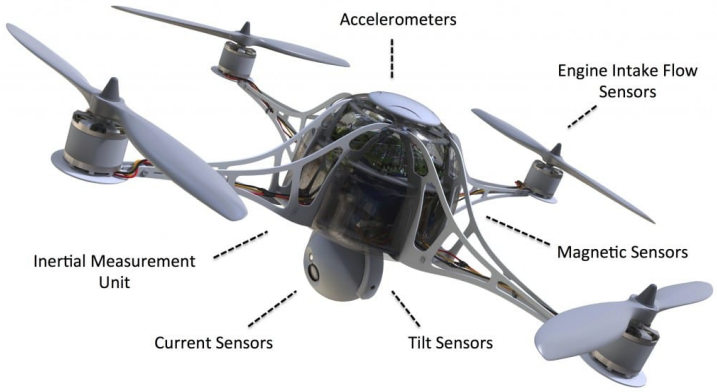
CONTENTS

1 INTRODUCTION

Typical embedded computing platforms combine one or more processors, memories, and input/output devices. They are typically used in the context of Cyber-Physical Systems (CPS), that is, the software running on the embedded platform controls the movements of the mechanical system that they are coupled with. For this reason many of them are classified as a real-time application, which need to respect timing constraints (e.g. deadlines) to ensure the system stability (AMINIFAR, 2016).

One of the many applications that use Embedded Systems are UAVs (Unmanned Aerial Vehicles).In this case the Embedded System is generally responsible for reading the sensor data, executing the control algorithm and controlling the UAV actuators, such as its rotors. UAVs can be controlled remotely by a human operator or can fly autonomously through a control program (RAFFO et al., 2010). In both cases, the use of an embedded control algorithm is required to ensure the flight stability (SILVANO, 2014), which is known as the low-level control of the aircraft. UAVs are considered a typical example of a CPS, where electric-mechanical devices are controlled through a computerized system which normally uses data provided from a sensor network, either external or distributed through its structure, as shown in Figure 1.

Fig. 1 – An UAV and its sensors.



Source:(WINKLER, 2016)

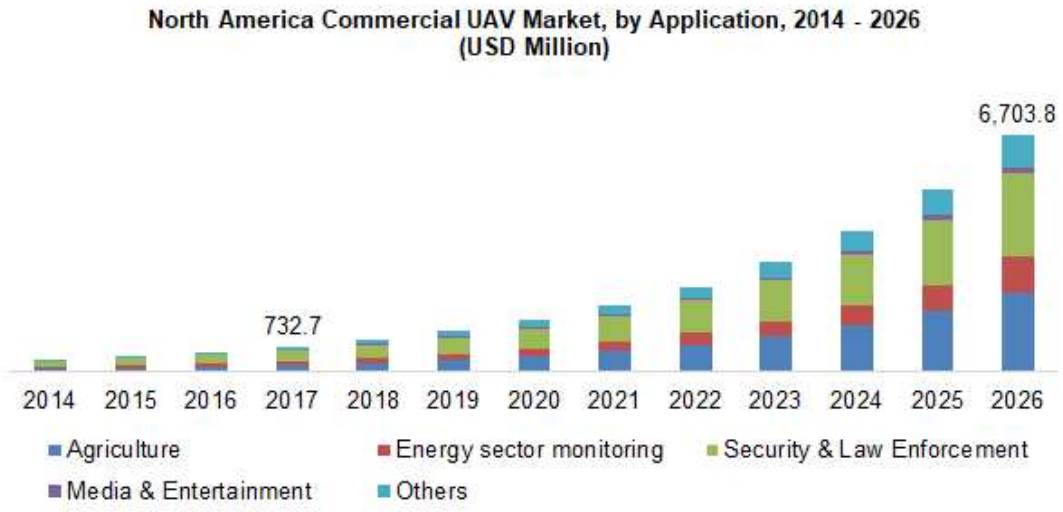
The utilization of UAVs is diverse and an area of constant research. In the military scope, UAVs have had great notoriety in the recent years for being deployed in Drone Strike missions, but are also heavily used for area reconnaissance and surveillance. Besides the military uses, it is possible to cite some areas where UAVs have had increasing interest and research such as:

- Search and rescue operations (RYAN; HEDRICK, 2005).
- Cargo Transport (REGO; RAFFO, 2016).

- Emergency medical care (JOHNSON et al., 2021).
- Agriculture or forests (SAARI et al., 2011).

With such diverse applications, the UAV market has increased significantly, as shown in Figure 2. With a constant demand for more efficient and competitive products it is worth investing in testing the products to evaluate the UAV stability. These types of tests are extremely important as they help improve current and future projects.

Fig. 2 – North America UAV Market Projection



Source: Adapted from ((RESEARCH, s.d.))

In the scope of UAV design, it must be highlighted the ProVANT initiative. ProVANT is an acronym for a collaborative project between the federal universities of Santa Catarina and Minas Gerais that started in 2012, with the UFSC team being a part of DAS (Department of Automation and Systems) under coordination of professor Leandro Buss Becker. The project main focus is related with researching the design and control of UAVs, allowing to promote advancements in these areas. There are many studies under development from this project and, with respect to the UFSC team, the most recent ones are:

- Communication between UAVS for rescue missions using Multi-agent systems
- Guidelines for a smart UAV design
- Utilization of ROS (QUIGLEY, 2009) (Robot Operating System) which can facilitate the development of embedded systems in cyber-phyiscal systems and improving portability between architectures
- Simulations using HIL (Hardware In the Loop) to analyze system stability in events where model checking could not be used due to the state explosion problem.

ProVANT also tackled an UAV design process. According to (GONÇALVES et al., 2016), the project of an UAV has three main phases: (i) Planning, (ii) Project Process, and (iii) Implementation. During the Project Process phase, after having the physical model of the UAV, it is possible to start the development of its control algorithm (DONADEL, 2015; MELLO, 2016). During this stage the development of the UAV embedded system also occurs. It will be in charge to manage all its tasks threads based on their priorities. Since the UAV is a critical system, this embedded system must be implemented in such manner to respect the system deadlines.

Analyzing the performance of the control algorithms developed to run on the UAV embedded system is another important task to be conducted within the Project Process phase. This analysis can be very useful during an UAV project development, providing a good indicator of the expected performance. Such analysis can be tackled in many different ways. One of the alternatives is to run these tests in real hardware or development boards such as BeagleBone, Raspberry Phi, and others. However, one disadvantage of this method is that its environment is restricted, not allowing one to modify hardware features, such as changing memory sizes or altering the CPU clock, number of cores, and the ISA (Instruction Set Architecture).

An alternative to such drawback is to use simulation tools. It happens that the majority of these tools are not precise enough in their calculations of execution time and frequently use methods of approximation to determine the simulation results, when the better solution would be a simulation with accuracy in the level of CPU cycles.

The ideal solution to execute these performance analyses would be a simulation tool with varied options to alter the simulation environment and offering simulation results with good precision when compared to real hardware tests.

This thesis presents the analyses of embedded control algorithms developed to control the flight stability of UAVs. Such analysis is valuable for embedded systems designers for allowing to observe the temporal performance of the algorithms and the platform architectural aspects that influence such performance. This allows us to: (i) test if the control program can be executed within its designed time window (deadline); and (ii) help to detect where the bottlenecks are in the control algorithm implementation when deadlines are not met. From the perspective of the embedded platform, it provide means for the embedded architect to better select the most appropriate target platform. These analyses should allow to iterate and explore modifications in software and hardware (TASHIRO; OYAMADA, 2016a) that need to be made so that deadlines are fulfilled and the system stability is not compromised (LATHI, 2005).

Simulation tools are often used during this analysis to speed up the process. However, these tools are limited in regarding the CPU architectures that they support, with some architectures, such as Intel x86, having significantly more simulation tools available than other architectures, such as ARM. The simulation tool that was chosen for the studies in

this thesis was the gem5 simulation tool, further discussions on system simulators and the gem5 tool are presented in sections 2.2 and 2.2.1. One of the main reasons for selecting *gem5* to be used along this work is the fact that it has a reasonably good precision when compared to performance measures of real hardware (BUTKO et al., 2012; AKRAM; SAWALHA, 2019).

Depending on the level of details in the simulation process, the simulation tool can be used to support Design Space Exploration (DSE) (TASHIRO; OYAMADA, 2016b). DSE refers to the analysis of the expected performance resulting from multiple combinations of designs, in order to determine which combinations are optimal. This activity can be executed manually, where the engineer chooses a set of designs based upon experience or observations from previous tests, or even automated, where a search algorithm will try to find the best suitable design combination for the project.

Despite the benefits of the simulation tools, a common problem is the initial barrier to learn the specifics of the chosen simulation software and time to set up the simulation process in a reproducible and agile manner. As an attempt to make the gem5 simulation more streamlined, a cloud computing environment was developed through the use of Google Colab. Such environment is let available for the scientific community in (SILVESTRE, 2021b). For the moment, this Colab implementation allows the simulation of ARM architecture programs, with the possibility of using m5ops library functions so that simulation data is collected only from specific regions from the program being analyzed. To help new users who do not know how to use gem5 or Google Colab, a simple youtube guide was made in (SILVESTRE, 2021a) which demonstrates the basic functionality of the current implementation.

In the literature it is normally discussed the problems of using cache when trying to achieve performance determinism in a computer program. A couple of different approaches can be used to try to improve program performance determinism and a more detailed discussion of this topic is presented in subsection 5.4, where some tests were made, both in hardware and software, regarding the usage of cache memory for the control programs.

1.1 OBJECTIVES

The main goal of this Master Thesis is to evaluate up to which extent the gem5 simulation tool can be utilized to support the design space exploration of real time embedded systems using an ARM-based CPU architecture.

Allied with the main goal, specific objectives are defined, as listed below:

- Verify if gem5 has sufficient precision in Full System mode.
- Demonstrate how the gem5 tool can be used to guide choices in hardware configuration, for instance the CPU, using data extracted from the simulations.

- Analyze the impact of cache memory usage in control algorithms and search alternatives to improve performance determinism in such regard.
- Offer an alternative to the gem5 simulation process using cloud computing, improving accessibility to the gem5 tool by simplifying its usage more simplified.

1.2 OBTAINED PUBLICATIONS

During the course of this Master in Automation and Systems Engineering, two papers were published at the Brazilian Symposium on Computing Systems Engineering (SBESC). The first one was in 2020 at the Work-In-Progress Track (SILVESTRE; BECKER, 2020) and was entitled *Performance Analysis of Embedded Control Algorithms used in UAVs*. This work consisted of various simulations on the gem5 simulation platform with different hardware configurations and some comparisons with real hardware tests, with emphasis on the gem5 simulation results.

The second paper was published in the main track of SBESC 2021 and was entitled *Using gem5 Simulator to Support Design Space Exploration Targeting ARM Architecture* (SILVESTRE et al., 2021). This paper consisted of an improved and updated version of the previous one, with further discussions on how the simulation data can be used to help guide hardware design choices and a detailed comparison of the results found in simulation and in real hardware tests. This work also included the implementation of the gem5 simulation process through cloud computing using the Google Colab platform.

1.3 THESIS STRUCTURE

The reminder parts of this Master thesis are organized in the following manner:

The second chapter has the objective of going over some technical concepts that are present in this thesis and briefly explaining them.

The third chapter discusses some related works in regarding gem5 precision and its usage in design space exploration.

The fourth chapter explains how the simulation environment was configured and how it can be altered to better emulate real hardware specifications.

The fifth chapter discusses the many experimental results that were obtained from gem5 simulation tool and real hardware tests from the Raspberry Pi 3B+.

The sixth and final chapter goes over considerations about the experimental results and explores possible future works that are still available.

2 THEORETICAL FOUNDATION

It is important to establish some basic notions on the themes that are discussed on this master thesis. The current chapter will present brief discussions regarding three main topics: Computer Architecture, Control Strategies, and System Simulators.

2.1 CPU ARQUITECTURE

In this thesis, some topics related to CPU design, Control theory and Full System Simulation will be explored, a brief explanation is then necessary to familiarize readers to the topics that will be discussed.

2.1.1 Instruction Pipeline

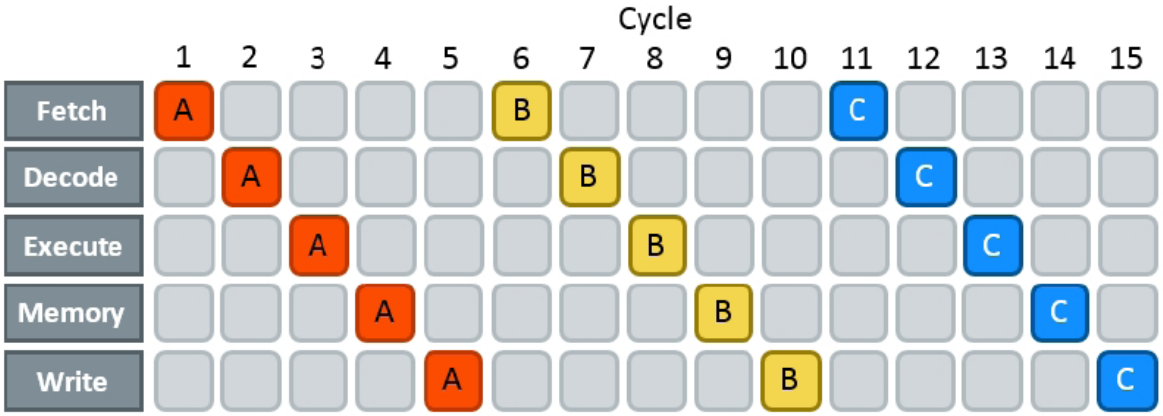
The pipeling method to order CPU instruction executing (ZARGHAM, 1996) started to gain popularity in the 1970 and 1980 decades, it is present in the great majority of modern CPUs and basically consists of a strategy to parallelize instructions executed by the same CPU core. The processing of each instruction by the CPU is not instantaneous, the number of stages vary depending on implementation, however they can be conceptually grouped in 5 steps :

1. *Fetch* : Collect the instruction from computer memory
2. *Decode* : Decode the instruction, seeing what is needed to calculate in order for its execution, for example the data from the result of a previously executed instruction
3. *Execute* : Execute the instruction
4. *Memory* : Access the data memory
5. *Write* : Write the instruction result in memory

Without the implementation of a pipeline, each instruction is processed exclusively from begin to end, as shown in figure 3 where it is assumed that each step would take 1 CPU cycle to execute. This is a very ineffective way of processing instructions, since each step normally uses different hardware parts of the CPU, for this reason modern CPUs implement pipelines to increase efficiency, as shown in figure 4.

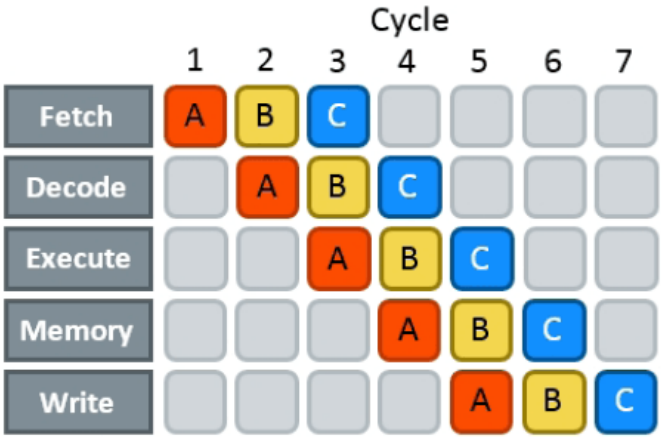
Instruction throughput vastly increases with a pipeline implementation, however it is important to note that average instruction execution time is slightly impacted negatively due to the overhead from the added pipeline process. Furthermore, an instruction pipeline usually has a more notable impact in processors with a reduced instruction set architecture, such as RISC-V or ARM, since these have more basic instructions that take less time to execute. The solution taken to increase instruction throughput in CISC (Complex Instruction Set Computer), such as x86, was to separate each instruction of the processor

Fig. 3 – Implementation without pipeline.



Source: ((SHINSEL))

Fig. 4 – Implementation with pipeline.



Source: ((SHINSEL))

in multiple micro-operations (μops), with the split of each instruction being realized in the decode stage of the pipeline.

2.1.2 CPU Branch Predictor

Another discussed topic related to CPU architecture is the Branch Predictor (LEE, s.d.), a digital circuit which tries to calculate the branch that a conditional instruction will take before it is in fact taken. The objective of the Branch Predictor is to increase the flow of instructions on the CPU pipeline and a good implementation is important for a modern processor performance. Without the implementation of a Branch Predictor, the CPU has to wait until the conditional instruction has passed through the execute stage of the pipeline so that the next instruction to be executed can go to the fetch stage.

The branch predictor tries to calculate which branch is the most likely to be taken and starts to process the instructions of this speculated branch, when the conditional instruction is executed, two situations can occur:

1. The branch predictor made a good calculation and the predicted branch was correct, the CPU continues the execution of this branch and the impact on performance is positive.

2. The branch predictor made a bad calculation and the predicted branch was incorrect, the processing done is discarded and the impact on performance is negative

The first time a conditional instruction is executed there is not much information to help the branch predictor choose the most likely branch, however as the computer program advances the branch predictor stores these past decisions and uses this to make future predictions. The level of complexity of the branch predictor varies based on its implementation, some of them for example are capable of recognizing patterns such as a branch that is taken every third repetition of a loop.

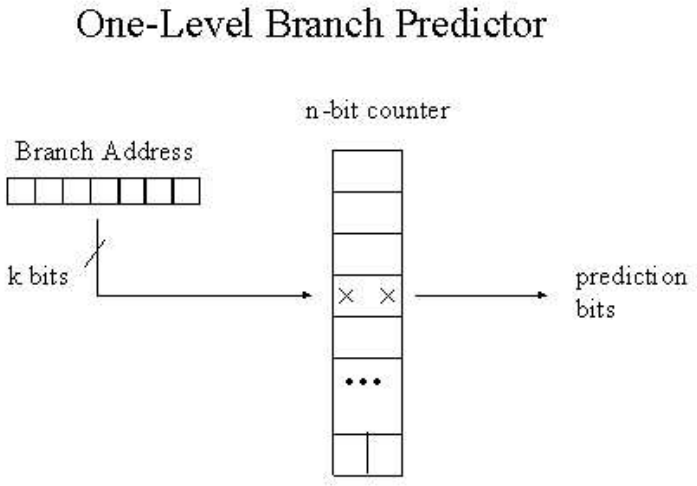
The most basic implementation of a Branch Predictor is found in the MIPS-x microprocessor, where all the branches are accepted as good prediction. With the technological improvements in processors, more complex implementations were needed and they can be split in two groups:

- Static Branch Prediction - The predictor uses only information obtained before the program was executed.

- Dynamic Branch Prediction - The predictor also uses data obtained in program execution time.

A simple model of a dynamic branch predictor is the One-Level Branch Predictor shown in figure 5, k bits of the branch address from a conditional jump are stored in the predictor and every time this branch is taken its counter is incremented with the maximum value on n, and when it is not taken its counter is decreased with the minimum value of 0. The predictor then takes the decision whether to continue on a branch by checking its counter value, if its less than 2^{n-1} the branch is not taken, otherwise the branch is taken.

Fig. 5 – One-Level Branch Predictor Model.



Source: ((LEE))

2.1.3 In Order and Out of Order Execution

Another topic related to CPU design is the method that the processor execute its instructions, where there are two general approaches:

1. In Order Execution - The processor searches, executes and finishes instructions in the order generated by the compiler. In this type of execution if an instruction takes longer to be executed and occupies the execute stage of the CPU pipeline for a long period o time, the remaining instructions have to wait. In this type of execution the instructions are statically scheduled.

2. Out of Order Execution - The processor searches the instruction in the order generated by the compiler, however it executes them in order governed by availability of instruction input data and execution unit. This type of execution tries to avoid CPU idle cycles while waiting for previous instructions to complete and in the meantime executes instructions that are able to run independently.

Out of Order execution generally allows higher performance out of the processor since it reduces CPU idle times that occur during the execution of a computer program. This comes with some disadvantages, probably the worst of them is the fact that this type of execution makes it very hard to calculate the WCET (Worst-Case Execution Time) and for this reason these types of processors are generally not recommended for systems with real time constraints.

2.1.4 Cache Determinism

In literature it is common to find discussions about the performance determinism problems that are introduced when cache memory is being used (LLORCA-CEJUDO; FRANDON, 2011), this happens for many reasons, one that can be mentioned is not having control in which data is stored or released from the cache during program execution. One of the problems introduced by the use of cache memory is that it makes calculating the WCET for a determined program a very difficult task (LESAGE et al., 2010).

However, when the complexity of a program increases, it can be unfeasible to not use cache as it benefits to performance can be necessary. There are quite a few alternatives to reach sufficient reliability in execution time while also using cache memory, such as using cache partitioning algorithms (BAI et al., 2016) and using advanced preemptive scheduling (DESTELLE; DUFOUR, 2010). A comprehensive survey of cache management techniques is presented in (GRACIOLI et al., 2015), providing a comparison of multiple techniques from the first studies of the field to more modern solutions. The main techniques therefore are:

- ScratchPad Memory
- Processing In Memory
- Software Prefetching

Both Scratchpad and Processing In Memory are emerging technologies that try to tackle different problems related to memory. Scratchpad memory can be considered similar to L1 cache with explicit instructions to move data to and from main memory, allowing some degree of control of the storage on this fast memory. Processing in memory refers to the integration of a processor with Random Access Memory (RAM) on a single chip, this is done to reduce latency related to memory bottlenecks. The main disadvantage of these two technologies is that they require designated hardware parts to be implemented, which makes it difficult to experiment their benefits on projects that are on their final design phases.

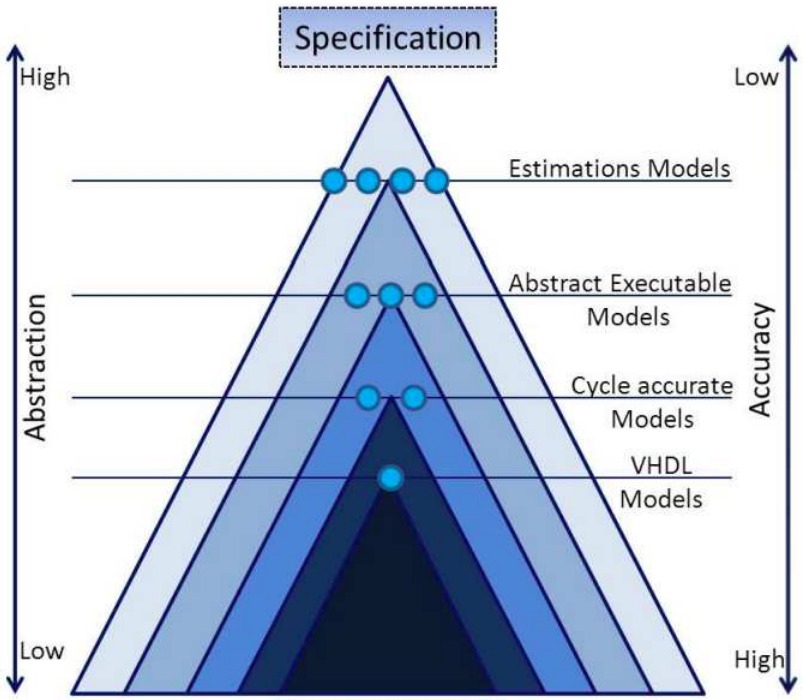
Software Prefetching however can be implemented on a much wider variety of programs, since it basically consists modifying the program by describing what data should be prefetched and put on the cache memory, so that when the data is necessary time wont be lost fetching the data from slower memories.

2.2 SYSTEM SIMULATORS

In the Computational Systems field, the analysis of the performance of an embedded system is a problem with constant research. The methods used vary a lot however one approach has been heavily explored in the recent years, this approach consists of using

executable models of hardware components to analyze performance during the design phase of the embedded system. The level of complexity from these models can vary, with the most complex models having the least abstraction from the simulated system, as shown in figure 6.

Fig. 6 – Abstraction and precision of executable models.



Source: (BUTKO et al., 2012)

The executable models that use HDL (Hardware Description Language), such as VHDL or Verilog, have a low level of abstraction and for this, if well implemented, can model with high precision a Computational System (PEDRONI, 2010). However these low levels of abstraction also brings a big disadvantage to these models since the modelling process ends up taking a long period of time and since these types of simulations are normally done during a project design phase a more agile process would be better suited. For this factor the majority of System simulation tools uses executable models with higher levels of abstraction.

A concept that is present in good part of modern simulation tools is the Full-System Simulation, where the software in some manner tries to emulate physical hardware so that the system under test operates more realistically when compared to real hardware tests, this also makes so that the simulated software does not need to be altered between simulation and real hardware execution. A Full System Simulator allows the user to simulate operating systems, devices drivers, kernels, and middleware. For this reason, one of its common use is to detect system failures before the hardware design phase is

completed. It can also can be used to test the performance on different configurations of hardware to guide later design phases.

Generally, the precision of a computer architecture simulator can be separated in three different levels:

- **Functional level** - The simulation provides performance results similar to real hardware tests, however there is no precision in total number of instructions executed or total CPU cycles
- **Instruction level** - Simulation data is similar to real hardware tests and simulation has an accurate idea of total instructions executed by the CPU
- **CPU Cycles level** - Information obtained from simulation provides an accurate number of instructions executed by the CPU and total CPU cycles, while still providing performance results similar to real hardware tests.

Another point that simulation tools differ from each other are the supported architectures and licence form. In general these simulation tools offer different types of models for CPU, cache and RAM memory and other systems, these models differ from each other mainly on level of complexity and type of system being simulated such as different CPU models for in order and out of order execution.

2.2.1 gem5 Simulation Tool

The present work makes use of the gem5 simulation platform (BINKERT et al., 2011). It is a system simulator that can be used for multiple means such as testing performance and behaviour of a target system. The simulation tool originated from the merge of two existing simulation tools:

- M5 (University of Michigan)
- GEMS (University of Wisconsin-Madison)

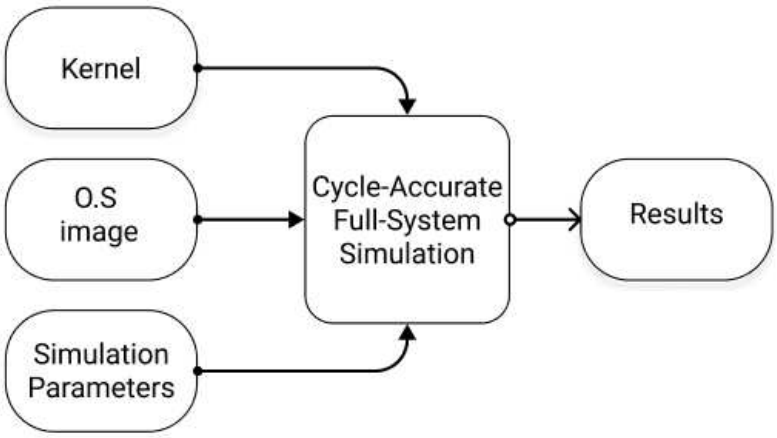
Combining the infrastructure and CPU models from M5 and the memory protocols and cache coherency models from GEMS, gem5 is classified as a modular discrete event-driven simulator with CPU cycles level of precision, this means that:

- Its components can be rearranged, parameterized or replaced easily to configure the desired simulation environment.
- It simulates the passing of time as a series of discrete events.
- It is considered a simulator platform that lets you use as many of its available components to configure a simulation system.

It is an Open Source software, which means its source code is available online for users to make personal adjustments, for this reason it is a simulation tool that is in constant evolution, receiving updates on the master branch as the community develops and tests them, either to add new functionalities or to facilitate existing ones.

The simulation tool currently is written in C++ for its model constructors and uses Python for scripts that are used to pass parameters for its constructors. The gem5 primarily has two simulation modes, the first one called Syscall Emulation mode (SE mode), where the simulation is limited to the execution of user space programs and to use system services developed by the tool, trading lower accuracy for higher simulation speed. The second mode is called Full System mode (FS mode) and it allows to simulate a complete system with its components and operating system (see Figure 7 for an overview of its data flow).

Fig. 7 – Overview of gem5 Full System mode.



Source: Author

Gem5 supports a high variety of ISA (Instruction Set Architecture) such as ARM, RISC-V, x86, Alpha, MIPS, Power, and SPARC. This allows the study of the impact on performance by using either reduced or complex instructions set architectures (AKRAM, 2017). Another important feature of gem5 is the facility to create and restore checkpoints along the simulation. This is important because it allows creating a checkpoint after the kernel and operating system initialization that, depending on the complexity of the adopted hardware models and on the host machine hardware, can take a few hours to complete - and this checkpoint can be used to skip such initialization on further simulations. Another possibility is to use checkpoints to swap CPU models between parts of the program being tested, since this is not possible to be done during simulation runtime. Thereby one can start with a simplified CPU model and further change to a more complex one during a specific segment of the code.

The gem5 simulator offers various CPU and memory system models that differ on the level of accuracy. For instance, the more complex CPU models include the simulation of the CPU branch predictor and instruction pipelines (ZARGHAM, 1996), both common features of a modern CPU architecture.

The precision of simulation tools is the main point of many studies and can be evaluated in many ways, one of them is through the simulation of benchmarks and comparing the variance between simulation and real hardware tests. The result from some of these benchmark tests can be seen in table 1.

Table 1 – *Mismatch* of Benchmarks executed on the gem5 tool

		SPLASH-2									
		Barnes	*FMM*	*Ocean*		*Radiosity*	*Water-Spatial*	*FFT*	*LU*		*Radix*
				contiguous	*non contiguous*				*contiguous*	*non contiguous*	
Execution Time (ms)	*Real System*	206.5	18.65	15.84	369.0	7473.5	448.8	1.984	397.2	7.996	777.9
	GEM5	1957.7	18.39	16.36	364.5	8559.7	521.1	1.895	427.3	8.227	638.3
Mismatch		5.08%	1.39%	3.26%	1.23%	14.53%	16.12%	4.49%	7.58%	2.89%	17.94%

Source: Adapted from (BUTKO et al., 2012)

Another important feature of gem5 simulation platform is M5ops, a set of opcodes that can be added to the program/code under test in order to provide very useful information. For instance, it allows the user to determine the part of the program code where the simulation tool should collect the performance data. This allowed the performance analysis of the control loops that are executed in the embedded system after the initialization, as shown in Listing 2.1.

```
1  int main(){
2  ...
3  hinfinity *control = new hinfinity();
4  control->config();
5  while(k<100){
6     m5_reset_stats(0,0);
7     out=control->execute();
8     m5_dump_stats(0,0);
9     k++;
10 }
11 return 0;
12 ...
```

Listing 2.1 – M5ops being used to isolate the program portion under analysis

It is important to note that to make use of the m5ops library the user needs to make some slight adjustments on the program source code and compiling process.

1. Compile the library for target ISA.
2. Insert m5op.h into the program folder and include it as a header file in the source code.
3. Statically link the m5op.S library when cross-compiling to target architecture.

Considering that this work tests used the ARM architecture, to execute the first step it is necessary to go into the /gem5/util/m5 path, rename Makefile.arm to Makefile and compile the library through the make command on a terminal on that path. With the library compiled and m5op header file included in the program under test folder and source code, all that is left is to link the library on the cross-compiling process, as seen on listing 2.2.

```
CC = arm-linux-gnueabi-g++ ...
CFLAGS = -static ${GEM5_PATH}/util/m5/m5op_arm.S ...

TARGET = vant3adap
all: $(TARGET)
$(TARGET): $(TARGET).cpp
	$(CC) $(CFLAGS) -o $(TARGET) $(TARGET).cpp
clean:
	$(RM) $(TARGET)
```

Listing 2.2 – Example Makefile

Lastly, gem5 also offers a couple different simulation binaries for the users to choose based on what they need for their research purpose, as presented on Table 2.

Table 2 – Available build targets of gem5

Binary name	Optimizations	Run time debugging support	Profiling support
gem5.debug		X	
gem5.opt	X	X	
gem5.fast	X		
gem5.prof	X		X
gem5.perf	X		X

Once a gem5 simulation is finished, it is possible to extract results by analyzing the generated stats files. These files keep a detailed log of every parameter of the hardware components used on the simulation and require some filtering by the user to extract a comprehensible overview of the test performance.

2.3 CONTROL STRATEGIES

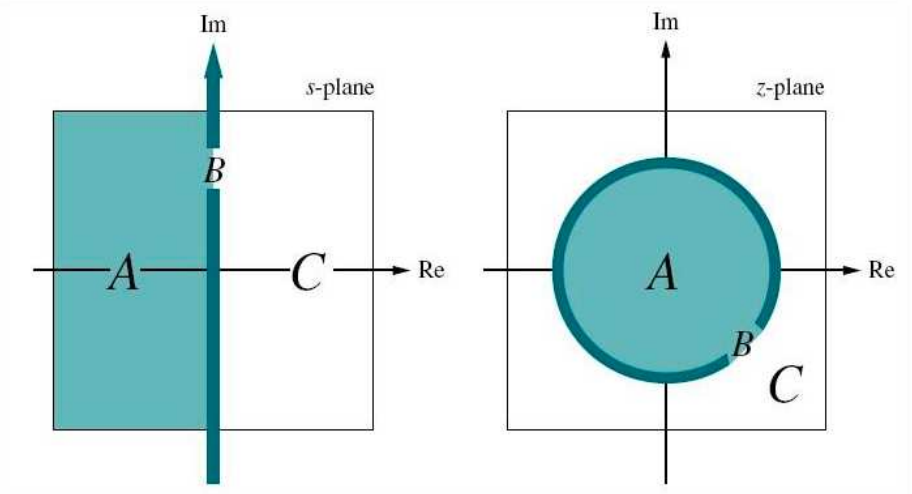
Explaining the algebra behind the analyzed control programs is not one of the objectives of this master thesis, however it is important to present some basic concepts to understand what distinguishes one from another.

2.3.1 System stability

Whenever we discuss the stability of a LTID (Linear Time Invariant Discrete) system, it is common knowledge that :

1. The system is absolutely stable if and only if all its transfer function H[z] poles are located inside the unit circle.
2. The system is unstable if one or two of the following conditions are true: (i) at least one pole is outside the unit circle, (ii) there are repeated poles on top of the unit circle
3. The system is marginally stable if there are not any poles outside of the unit circle and there is only simple poles on top of the unit circle.

Fig. 8 – Regions of s and z plane.



Source: (VILLANUEVA, 2019)

Since the H[z] transfer function depends on the sampling period (VILLANUEVA, 2019) as $z = e^{Ts}$, it is a logic conclusion that if for any reason the processor is unable to execute the control program inside this designed control window, the stability of the system itself is going to be compromised as the resulting poles will not be the ones planned.

2.3.2 Linear Quadratic Regulator

One of the most common strategies for optimal control is the *Linear Quadratic Regulator* (DONADEL et al., 2014). Its objective is to control a dynamic system described by set of differential equations, with minimal cost. The tuning of the control is done using an algorithm that minimizes the cost function that uses criterion provided by the engineer, this cost function generally consists of some form of sum of the difference of system variables, such as height or temperature of the process, from the desired values. The algorithm then finds the adjustment necessary to minimize these undesirable variances and normally this tuning is an iterative process, where the engineer through simulation tries to find an optimal control that is in line with the main objectives of the project. In general the LQR control method consists of trying to find a control with state feedback that minimizes a cost function designed by the responsible engineer.
For a Continuous Time Linear system described by 2.1:

$$\dot{x} = Ax + Bu \tag{2.1}$$

with a cost function defined by 2.2

$$J = \int_0^\infty \left(x^T Q x + u^T R u + 2x^T N u\right) dt \tag{2.2}$$

the control that will minimize the cost function through state feedback is 2.3:

$$u = -Kx \tag{2.3}$$

with the K gain calculated as 2.4:

$$K = R^{-1}(B^T P + N^T) \tag{2.4}$$

and the P matrix can be found solving the Riccati Equation 2.5:

$$\mathcal{A}^T P + P\mathcal{A} - PBR^{-1}B^T P + \mathcal{Q} = 0 \tag{2.5}$$

with 2.6:

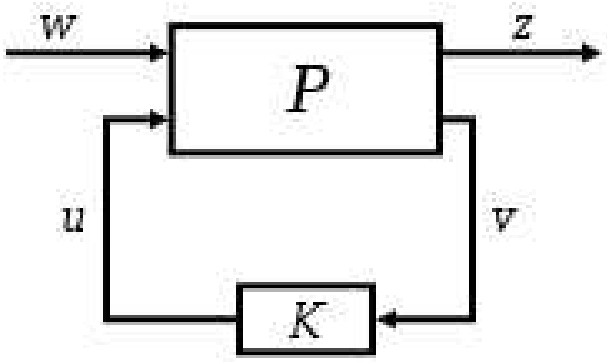
$$\mathcal{A} = A - BR^{-1}N^T \qquad \mathcal{Q} = Q - NR^{-1}N^T \tag{2.6}$$

2.3.3 H2/H∞

The control algorithm H∞ (MELLO, 2016) is used to synthesize a controller that reaches stability with ensured performance. To utilize this method, the control designer has to express the control problem as an mathematical optimization problem to find the control that solves this optimization problem. One of its disadvantages is the need for good mathematical foundations for the method to be applied correctly and a good model of the system being controlled. As advantages over more classic control techniques is

the fact that the H∞ control method is applicable in multivariable systems with coupled control variables. The denomination H∞ comes from the mathematical space where the optimization is done called Hardy space. This control method is represented in figure 9

Fig. 9 – Process model for the H∞ control method



Source: Wikipedia

The system can be represented by 2.7 and 2.8

$$\begin{bmatrix} z \\ v \end{bmatrix} = \mathbf{P}(s) \begin{bmatrix} w \\ u \end{bmatrix} = \begin{bmatrix} P_{11}(s) & P_{12}(s) \\ P_{21}(s) & P_{22}(s) \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix} \tag{2.7}$$

$$u = \mathbf{K}(s)\, v \tag{2.8}$$

and the relation of z and w can be expressed by 2.9

$$z = F_\ell(\mathbf{P}, \mathbf{K})\, w \tag{2.9}$$

doing the linear fractional transformation, F_l can be expressed by 2.10

$$F_\ell(\mathbf{P}, \mathbf{K}) = P_{11} + P_{12}\, \mathbf{K}\, (I - P_{22}\, \mathbf{K})^{-1}\, P_{21} \tag{2.10}$$

and the objective of a H∞ method is to find the controller K that minimizes the infinity norm 2.11

$$||F_\ell(\mathbf{P}, \mathbf{K})||_\infty = \sup_\omega \bar{\sigma}(F_\ell(\mathbf{P}, \mathbf{K})(j\omega)) \tag{2.11}$$

and in similar form to a control with H_2 design.

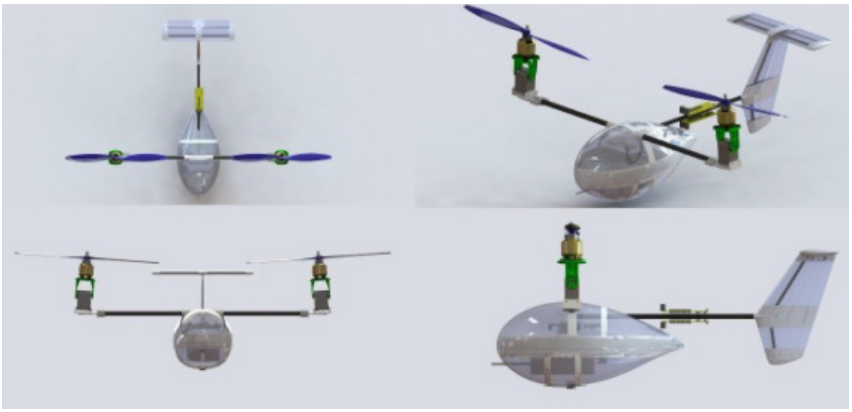
2.3.4 Adaptive Mixing

The *Adaptive Mixing* control strategy analyzed in this thesis was introduced in a work published in the 2016 IEEE 19th International Conference on Intelligent Transportation Systems by members of the ProVANT project (CARDOSO et al., 2016).

Basically it consists of a controller designed specifically for a ProVANT tilt-rotor UAV presented in figure 10. The dynamic model is obtained using the Euler-Lagrange

formulation considering the aerodynamic forces and torques exerted on the horizontal and vertical stabilizers, and fuselage. Based on the linearized dynamic models, a mixed H2/H∞ robust controller is designed for each operation point and an adaptive mixing scheme is used to perform a smooth gain-scheduling between them.

Fig. 10 – ProVANT Tilt-rotor UAV 3.0



Source: Adapted from (CARDOSO et al., 2020)

The robust adaptive mixing controller was corroborated by numerical experiments conducted in the ProVANT simulator developed using the Gazebo and ROS platforms. The tilt-rotor UAV control program had a trajectory comprising stretches with VTOL, forward acceleration, circular path and deceleration, subjected to time-varying wind disturbances. The results showed a successful trajectory tracking obtained with the proposed control strategy.

3 STATE-OF-THE-ART ANALYSIS

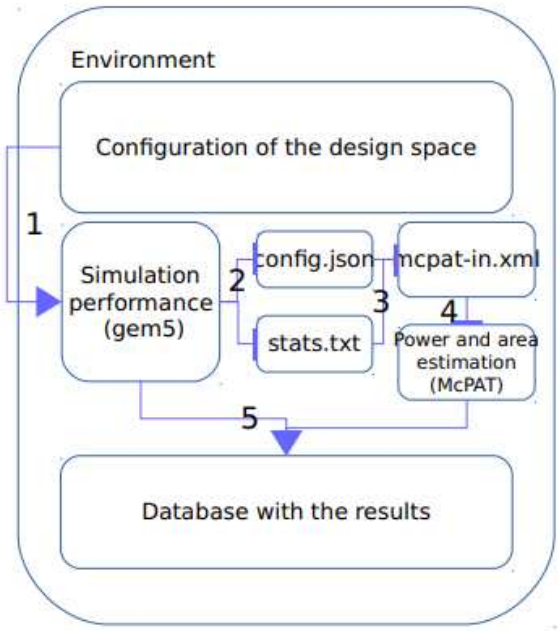
This chapter presents some related works that also used gem5 for design space exploration. Moreover, it discusses some publications related with gem5 precision and capabilities.

3.1 DESIGN SPACE EXPLORATION WITH GEM5

An important reason for selecting the gem5 simulation tool is its extensive options to configure the hardware configuration of the system under test. This can be of great use for design phases of a project where the final hardware configuration is not yet decided, in which gem5 simulation results can be used to help guide hardware choices and help design space exploration.

In (TASHIRO; OYAMADA, 2016b), gem5 is used with McPAT to estimate performance, energy, and other characteristics of different solutions in a multi-core Architecture. In this work a design space with 17496 architectures was explored, the simulation environment is shown in figure 11. The application consisted of a matrix multiplication and the design space was separated in two, where one simulated the application in a single core architecture and the other in a quadcore. The total time to simulate this design space

Fig. 11 – Simulation Environment of gem5 being used with Mcpat



Source: Adapted from (TASHIRO; OYAMADA, 2016b)

was approximately 9 days. This implementation, as their authors acknowledge, is very exhaustive and slow, which makes its use very restrict. However it still holds relevance since it shows gem5 great capability to evaluate multiple hardware configurations, regarding quality and performance.

In (SAM et al., 2019) a new simulator called SMAUG is proposed, this simulator uses a modified version of the gem5 simulator called gem5-Aladdin for its simulation of custom Systems on Chip (SoC). SMAUG is a framework that allows simulation of Deep Learning models on custom SoCs with a variety of hardware accelerators. SMAUG is designed to enable Deep Neural Network researchers to evaluate different accelerator and SoC, helping in the process of making hardware and software design choices. In this work it is described how the SMAUG simulator can be used to optimize performance of a wide range of Deep Neural Networks to achieve significant speedup by optimizing some aspects such as SoC-accelerator interfaces.

The gem5-Aladdin simulator used in (SAM et al., 2019) was presented in (SHAO et al., 2016), where it is discussed that the co-design of the accelerator micro-architecture with the system in which it belongs is critical. It is also discussed that data movement and coherence management for accelerators are significant in total accelerator runtime but are often unaccounted, which results in misleading performance predictions. The gem5-aladdin simulator was developed to explore the design space of accelerator-system co-design, it is an SoC simulator that captures dynamic interactions between accelerators and the SoC platform. To validate this simulator, the authors conducted real hardware tests and found the simulator results within a 6% margin against real hardware.

It is clear that the use of gem5 for exploring hardware and software design choices is an active area of research. Some studies focus on using the simulation tool to explore a wide range of architectures and how they correlate with the expected performance on the target software (TASHIRO; OYAMADA, 2016b), others focus more on analyzing software performance by using or improving gem5 device models (SAM et al., 2019) (SHAO et al., 2016). However, there seems to be a lack of studies on the area of comprehensively going through a Full-System simulation result, exploring how it can be useful in helping a system developer to make hardware and software choices with overall performance and resource utilization in mind.

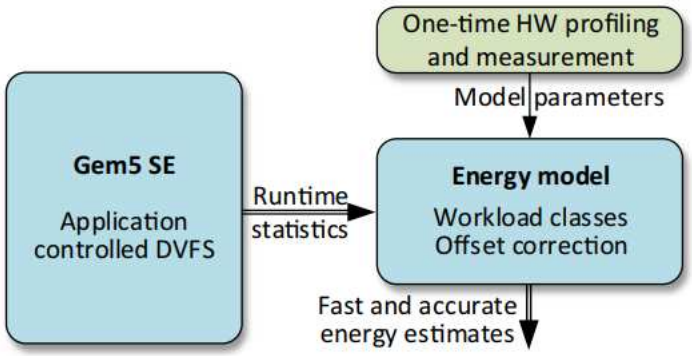
3.2 GEM5 PRECISION

One of the reasons why gem5 was the simulation tool chosen to run the tests of this thesis was the good amount of published works mentioning its good precision of simulation data when compared with real hardware data.

In (BUTKO et al., 2012), various benchmarks are tested, which vary on application such as scientific workloads, media applications and memory bandwidth benchmarks. The results show a mismatch between 1.4% to 18%, depending mostly on memory traffic. The conclusions mentioned that the worst case of mismatches happened due to an overly simple model of the external DDR memory in GEM5 used at the time that did not model DRAM behaviour adequately.

The precision of gem5 tool is also discussed in (YASSIN et al., 2021), where its syscall emulation mode is used with a energy model to calculate the Dynamic voltage and frequency scaling to obtain the total energy consumption, represented in figure 12. Using the user level alterations described in the work, and some systematic offset error corrections, the resulting deviation of energy consumption was at most 4%.

Fig. 12 – Environment of gem5 used with energy models



Source: Adapted from (YASSIN et al., 2021)

Furthermore, in (AKRAM; SAWALHA, 2019) it is presented a validation of gem5 simulator for x86 architectures. In such work the gem5 models were configured as close as possible to match the target Core-i7 Haswell microarchitecture configurations, and some of the existing gem5 code was modified. By fixing some sources of error that were present it was possible to validate the gem5 simulator, reaching results with less than 6% mean error rate for different control, memory, dependency, and execution benchmarks.

It is important to mention that simulation tools can have greatly varying precision for different systems, it can have good reliability for some and be inconsistent for others. In this regard it is very important to first verify whether the simulation tool can reliably be used to analyse the target system, by comparing real hardware tests with data from the simulation.

Clearly gem5 is used in many studies that explore, among other points, the precision of a Full-System simulator. Some studies focus on using the simulator in combination with other tools to deduce important information such as energy consumption (YASSIN et al., 2021). Meanwhile, other studies focus on analyzing the simulation data precision when compared to real hardware tests, making use of varied benchmarks that stress different parts of the system hardware. This study also has as one of its objectives verifying gem5 simulation data precision, but instead of using available benchmarks it uses control programs developed by the ProVANT project. This allows the study of gem5 being used to analyze different control programs characteristics, such as resource utilization, overall performance and possible bottlenecks detection.

3.3 REAL-TIME SYSTEMS

Another possible area of research of gem5, related to its models precision and capacity to be modified, is its use analyze real-time systems.

In (FARSHCHI et al., 2017) gem5 is used to verify a deterministic memory model proposed by its authors. This memory abstraction was developed to better express memory criticality, contributing to improve efficiency and predictability of real-time computing on multicore architectures. As a result, by using this deterministic memory, the system used 49% less cache space, on average, than the conventional way-based cache partitioning method.

In (LIMA, 2019) it is presented a supporting mechanism for using Processing in Memory (PIM) in the gem5 simulator. It also includes a methodology for prototyping PIM accelerators. The main purpose of this proposed simulator, based on gem5, is to contribute to the PIM research field by allowing computer architects to model and analyze a full environment.

Another possibility is to use gem5 to simulate modifications in the system architecture, with the purpose of achieving better performance determinism, as shown in (FARSHCHI et al., 2017) and (LIMA, 2019). This Master thesis focuses, however, on the analysis of whether gem5 can determine if a target hardware is appropriate for the implementation of a time-constrained control algorithm, as it is the case in the ProVANT UAV.

Analysing all the mentioned works, it becomes clear that gem5 is actively used on the real-time computing research field.

3.4 SUMMARY

In this state-of-the-art analysis it was possible to notice a certain lack of studies that comprehensively explore gem5 simulation results, and how they can be used to help a system designer to make hardware choices based on them. Furthermore, most of them make the use of benchmarks, which can be great for reproducibility of the tests and testing certain parts of the simulator hardware models, since these benchmarks tend to stress test one or a few parts of the system hardware. But with benchmarks, these studies lack in exploring the capabilities of gem5 being used to point out possible shortcomings in the developed code and how it could be improved to obtain better temporal performance or resource utilization.

With that said, the main motivations of this Master Thesis are to further explore these aforementioned points: (i) comprehensively explain how gem5 simulation results can be used to help a system designer make hardware choices; (ii) explore gem5 capabilities of pointing out code shortcomings and possible improvements to increase program temporal performance and resource utilization.

4 SIMULATION ENVIRONMENT SETUP

It is important to go over a few details regarding the setup of the simulation environment, how it differs from a default Full System gem5 simulation and some choices that were made from the available models on the simulation tool. This is important for the following reasons: (i) allows the replication of the tests executed on this work; (ii) explains the reasons certain models were chosen to better represent the target hardware configuration. This discussion will go over the following points:

- The simulated control programs and how they differ from one another.
- The simulated hardware configurations and the reasons that they were chosen.
- What gem5 models and configurations were used for the executed tests .
- What changes were done on gem5 models to better suit the simulated hardware.

4.1 SYSTEM UNDER TEST

To make the performance analyses envisioned in this investigation, three C++ control programs developed within the ProVANT project were selected. They were designed to control UAV prototypes like the one shown in Figure 13, and cover a different spectrum of control techniques with varying complexity. The first algorithm uses a common Linear Quadratic Regulator (LQR) strategy, the second a mix of H_2 and $H\infty$ strategies with a feedforward control (DONADEL, 2015) to control a scenario where the UAV is carrying a load, and finally the last one uses uses an approach called adaptive mixing control (CARDOSO et al., 2016). An important characteristic is that these algorithms were designed to be executed within a 12 milliseconds window, otherwise the UAV may not stabilise. Table 3 compares source code information extracted from the control algorithms, which are available for download in a github repository (SILVESTRE, 2021c).

Table 3 – Comparison of LQR and H2/H∞ source codes

	LQR	H2/H∞	AdapMix
Characters in source code	21 597	2 410 134	84 530
Compiled Binary Size	2.56 MB	9.27 MB	10.9 MB

Regarding their source code structure, they share a lot in common, like the use of a C++ class structure. The Eigen library is used by all of them for solving matrix calculations and both use the Robot Operating System (ROS) libraries to package sensor data and communication info. One should notice the huge header file of 2.3 MB only present in the H2/H∞ program, needed by its Feedforward implementation.

Fig. 13 – VANT prototype from the ProVANT project.

Source:((MACRO-UFMG))

They differ, however, in some important aspects. The first difference relates with their matrix sizes, as in LQR the Expanded State Vector has 20 states, in the adaptive mixing 18 and in H2/H∞ it has 24. This suggests more complex matrix calculations for the H2/H∞. They also distinguish on the Feedforward implementation. In LQR the disturbance is dealt as a constant value and the Feeforward can be performed as a simple addition before the output. In H2/H∞, however, since it is designed to control a scenario where the UAV has to carry an attached load, the Feedforward implementation has to calculate the disturbance with a set of complex matrix operations in every iteration of the control loop. There is no feedforward action present in the adaptive mixing control algorithm.

Regarding the Operating System image and Kernel, the tests executed in this paper used a compact aarch64 OS image file and a Linux kernel. These were chosen to be similar to the Raspberry Pi default ambient and can be found in the gem5 resources (GEM5, 2021) , however they can be built and modified freely to test different configurations, such as a simplified OS with with the intent of reducing background processes.

Regarding the compiling process for these algorithms, it is important to mention that:

- arm-linux-gnueabi-g++ was used to cross-compile the programs to the ARM architecture.
- To allow the execution of hardware floating-point instructions the compiler flag *-mfloat-abi=softfp* was used.
- To optimize the code, the *-O2* compiler flag was used. It was chosen instead of *-O3* because it was important to optimize the code but maintain it as small as possible (*-O3* typically generates larger machine code).

4.2 HARDWARE SETUP

For the initial test setup it was decided to configure the simulation by cloning the specifications of the Raspberry Pi 3 Model B+ (see Table 4), a very common ARM architecture development board (see Figure 14). This configuration was chosen in order to compare obtained results with the results of tests executed on the real hardware, and since current ProVANT control programs were designed for an ARM processor architecture, so using the Raspberry platform was a natural option.

Table 4 – Raspberry Pi 3 B+ Specs - baseline setup

CPU	4 Core Cortex-A53 (ARMv8) 1.4Ghz
L1 I Cache	16 kB
L1 D Cache	16 kB
L2 Cache	512 kB
RAM	1GB LPDDR2 SDRAM

Fig. 14 – Raspberry Pi 3 Model B+.

Source: Raspberry Pi

Besides this baseline configuration, some other hardware setups should be mentioned. The first one consists of the tests where the cache memory was removed. For such tests L2 cache was completely removed and L1 cache was reduced to 32 B for instructions and 32 B for data. L1 Cache could not be completely removed due to a limitation of the gem5 current memory models, even so 32 B approximately reduces the cache to 0,2% of its actual size. The second setup that can be mentioned consisted of the tests where

a simulation of the improvement in performance a Cortex-A72 CPU architecture would bring, for these tests we used Raspberry Pi 4 as the hardware specifications (see Table 5). Although a similar compromise had to be taken here since due to a current tool limitation cache values have to be in the power of two, for this reason the L1 Instruction cache was reduced from 48KB to 32KB.

Table 5 – Raspberry Pi 4 Specs - alternative setup

CPU	4 Core Cortex-A72 (ARMv8) 1.5Ghz
L1 I Cache	48 KB*
L1 D Cache	32 KB
L2 Cache	1 MB
RAM	1-4 GB LPDDR4-3200 SDRAM

* reduced to 32KB due to gem5 tool limitation

It is important to note that besides these basic memory sizes and clock differences, there are other significant changes in CPU architecture such as pipeline depth and type of execution, a more detailed comparison is shown in Table 6.

Table 6 – CPU Architectures Comparison

Architecture Comparison	Cortex-A53	Cortex-A72
Decode	2-wide	3-wide
Pipeline depth	8	15
Type of Execution	In-Order	Out-of-Order
L1 Cache (Instr + Data)	8–64 KB + 8–64 KB	48 KB+ 32 KB
L2 Cache	128 KB - 2 MB	0.5 - 4 MB

4.3 GEM5 SETUP

Besides the hardware configurations and software setup, it is important to mention some choices and adaptations made on the gem5 simulation tool to run the analyses performed on this work.

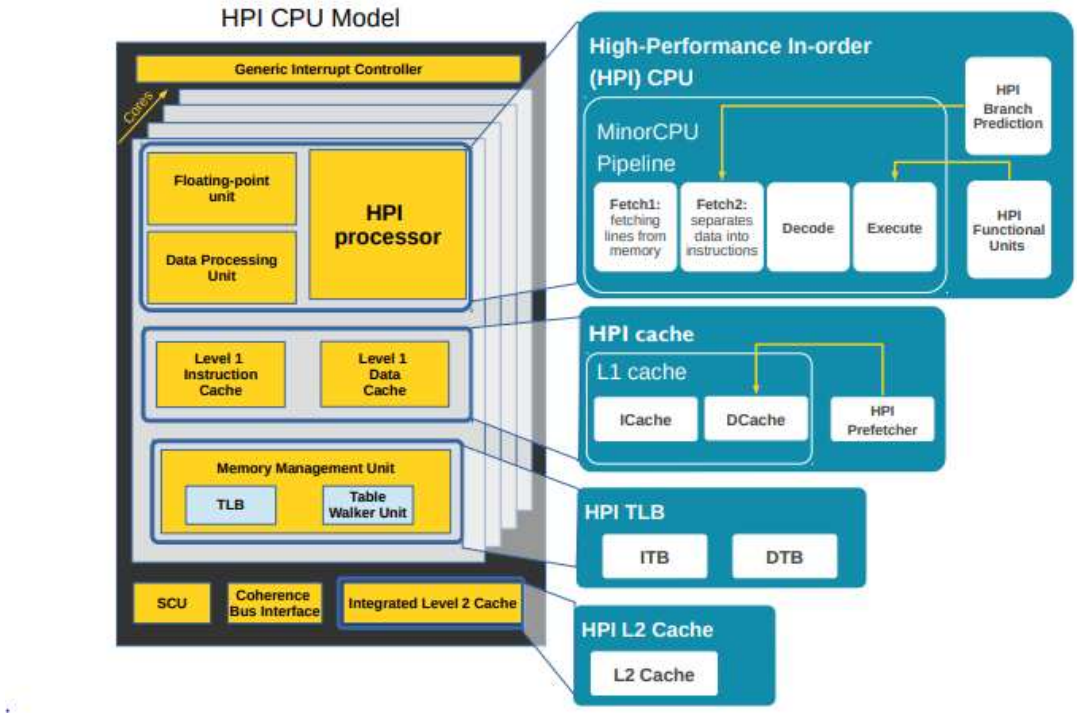
The simulation analyses conducted on this thesis were performed using the following gem5 models, binaries, and parameters:

- ISA: *ARM*
- CPU Model: *HPI* and *O3*
- Memory Model: *Classic*

- gem5 simulation binary: *gem5.fast*
- gem5 memory type: LPDDR2

The HPI CPU model, represented in figure 15, was used for the Raspberry Pi 3 B+ performance tests and the O3 model for the Raspberry Pi 4 tests. This modification was needed in order to simulate the different types of CPU executions, in order and out of order.

Fig. 15 – HPI CPU Model.



Source: (TOUSI; ZHU, 2017)

The simulations were conducted on a Ubuntu 18.04 OS with a Dell G3 3590 notebook as the host machine, which has a 9300h Intel quad-core CPU with a clock speed of up to 4.1 GHz, and 8 GB of 2666 MHz DDR4 RAM.

4.3.1 Improving gem5 CPU models with real hardware info

One of the advantages gem5 offers is that its hardware models are open to be modified by the user, this allows for further fine tuning to the target simulated CPU by adapting the model values to those that are verified through benchmarks. For example, for the tests executed in this thesis, the cache latency was modified to match the Cortex A53 values found in benchmarks (7-CPU, s.d.), by modifying the HPI.py file in gem5 folder, as demonstrated in listing 4.1. The O3 CPU model was also modified to match the pipeline described in Table 6.

```
class HPI_DCache(Cache):
    data_latency = 1
    tag_latency = 1
    response_latency = 1
...
class HPI_L2(Cache):
    data_latency = 13
    tag_latency = 13
    response_latency = 5
```

Listing 4.1 – Cache Latency values in gem5 CPU model python file

4.4 GOOGLE COLAB IMPLEMENTATION

As an attempt to make the gem5 simulation process more streamlined, a Google Colab environment was developed where so that the user can use cloud computing to experiment gem5. The environment can be accessed in (SILVESTRE, 2021b). So far it allows the user to simulate ARM architecture compiled binaries on a Full System simulation mode on a Linux based machine.

The process to use gem5 through Colab basically consists of:

- Installing the gem5 dependencies and the simulation tool, in this step the precompiled files are also downloaded to cut the initial setup required to run gem5
- Import a prebuilt ARM compiled binary from Google Drive
- Choose simulation Parameters, such as CPU clock speed, number of cores and cache sizes.
- Run the simulation block
- Download the stats.txt file from the /gem5/m5out path to analyze program simulation performance

A basic guide to use the Google Colab gem5 implementation to simulate ARM binaries was prepared in the form of video, which can be accessed in (SILVESTRE, 2021a).

For more familiarized gem5 users, there are options to create their own checkpoints and run the control program under test directly from the simulation terminal, these options however require the user to access the Google Colab host machine through Secure Shell protocol (SSH), besides a basic understanding of the gem5 simulation process.

Some approaches can be used to connect to the Google Colab machine through SSH, the one chosen for our implementation uses the Ngrok service. After successfully running

the SSH into Colab cell two results will be generated, one is the command that can be used through a Linux terminal or windows command prompt to connect using SSH into the Colab host machine.

After accessing the Colab host machine through SSH, the user can start the simulation using checkpoint 1, which restores the simulation to an idle state in terminal waiting for commands. Shortly after the simulation cell starts some messages are printed in the cell to report the simulation status, between these messages there is one that reveals the port used to accept connections to the simulation terminal, which by default is 3456:

system.terminal: Listening for connections on port 3456

After confirming the terminal port, the user can go to the terminal which accessed the Colab host machine through SSH and run the *telnet localhost 3456* command. This will connect to the simulated Linux O.S terminal and the user can execute their desired tests through this terminal.

Once the user has accessed the terminal of the simulated Linux system, many actions become available, a few that can be listed are:

- Build the workloads image file, which contains the imported compiled binary under the name of SUT, through the command *sudo mount /dev/sdb /mnt/p2*.
- Browse through the simulated Linux system
- Create their own checkpoints with the *m5 checkpoint* command through the terminal or called inside the compiled program.
- Read a custom .rcS script through the command m5 readfile |sh.
- Explore the m5op functions, which are listed by typing into the terminal the *m5* command.

The current implementation has space to be expanded, for example in different ISA support such as x86 and further improving the capabilities for more advanced users to import their own gem5 models and files in an easier way. Still it offers a great way for inexperienced gem5 users to run ARM simulations in a Linux based system in a few minutes and offers a way for more advanced users to manually make their own tests by accessing the Colab host machine through SSH.

5 EXPERIMENTAL RESULTS

The simulation process for each algorithm consisted of the execution of 100 control loops, with data from the UAV sensors considered the same static integer value for all loops, then the data from each control loop as well as the overall program performance was analyzed. For the H2/H∞ and AdapMix algorithms some additional tests on the Raspberry Pi board were made to measure gem5 precision on some CPU data such as total instructions count.

5.1 LQR ALGORITHM RESULTS

An important measurement for control programs is the execution time of each control loop. This is used to determine whether the control program is able to execute its algorithm inside the designed control window of time. The simulation results for the first 100 control loops executed in the LQR algorithm are presented in Table 7.

Table 7 – CPU Data from LQR program

LQR CPU Data	min	max	max*	avg
Execution time	1 us	29 us	2 us	1.5 us
Idle Cycles	2.04%	22.55%	21.72%	2.45%
Cycles Per Instruction	1.53	2.35	2.14	1.58
Instructions Executed	1 352	17 000	1 531	1 501
Operations Executed (including micro ops)	1 829	19 060	2 026	2 003

* Disregarding the first control loop

From these results it is clear that the LQR algorithm was able to be executed in quick manner with a worst case execution time of 29 microseconds on the first control loop, still well inside the designed control window of 12 milliseconds. It is also interesting to analyze Memory utilization during the control program execution, this information is present in gem5 results and is presented in Table 8.

Table 8 – Memory Data from LQR program

LQR Memory Data	min	max	max*	avg
Memory BUS Utilization	0%	3.93%	2.56%	0.09%
RAM READ	0 KB	3.6 KB	0.25 KB	0.04 KB
RAM WRITE	0 KB	1 KB	0 KB	0.01 KB

* Disregarding the first control loop

As suspected the LQR was a program that barely used the Memory BUS of the Raspberry Pi, with the worst case being the first control loop with reads and writes of a

few kilobytes. During the rest of the control loops it was common to have zero DRAM BUS utilization, suggesting that Raspberry cache was sufficient for this control program data profile. Another useful information available in gem5 results is the types of CPU instructions executed during the control program, this data is available in Table 9.

Table 9 – Instruction Count from LQR program

Instruction Type	Total Count	Percentage
IntAlu	107 770	50.13%
IntMult	281	0.13%
FloatAdd	3000	1.40%
FloatMult	2100	0.98%
FloatMultAcc	8400	3.91%
MemRead	55024	25.59%
MemWrite	38111	17.73%

For the LQR algorithm there were very few Floating Point instructions executed, with the big majority being executed in the ALU (Arithmetic Logic Unit) and memory instructions. This information combined with the temporal performance can be useful for the process of choosing the CPU for a project. In the case of the Linear Quadratic Regulator the floating point instructions were a small percentage of the overall instructions, this coupled with the fast performance showed in Table 7 suggests that a CPU with hardware floating point is not necessary for this control implementation.

To check this alternative, some additional tests were executed through simulation, with floating point operations being solved by library calls. Obtained results showed that the average execution time for this control program increased to 48 microseconds, with a worst case execution time of 82 microseconds. Overall, despite not leading to deadline miss, not using the hardware devoted for floating point computation caused a significant increase in the execution time.

5.1.1 Comparison with Raspberry Pi results

Tests performed on the Raspberry Pi board confirmed the longer execution time for the first control loop. Figure 16 presents the execution times of the first 10 control loops.

For the LQR control program, the executions times between gem5 simulation and real hardware had a mismatch of approximately 2% for the initial 10 control loops. Overall, gem5 was accurate for this LQR implementation and provided data with sufficient precision to help understand the control program performance.

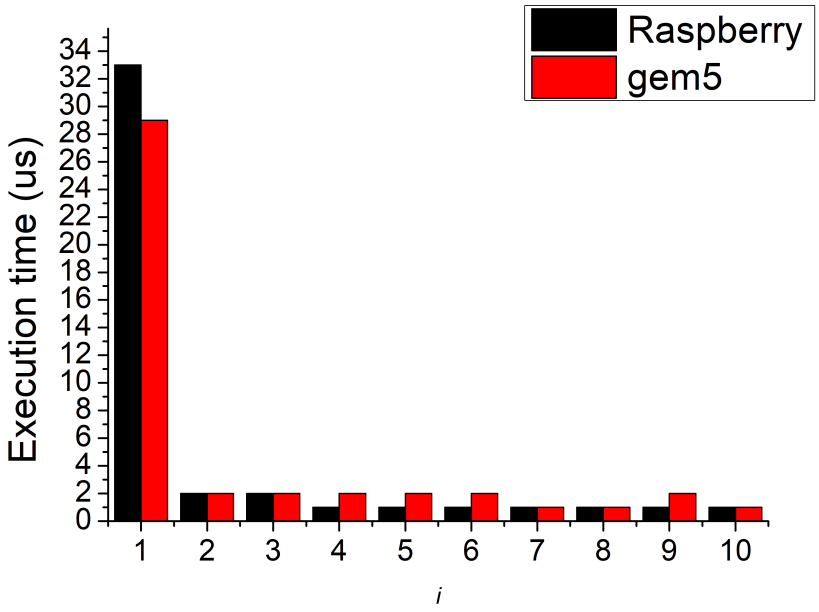


Fig. 16 – Execution time of simulation and physical hardware for the LQR.

5.2 H2/H∞ ALGORITHM RESULTS

Similarly to the LQR results, the first tests consisted of an analysis of each control loop and later an overview of the control program execution as a whole. The first 100 control loops execution times are presented in Table 10 along with some other CPU data.

Table 10 – CPU Data from H2/H∞ program

H2/H∞ CPU Data	min	max	max*	avg
Execution time	2.15 ms	6.57 ms	2.30 ms	2.25 ms
Idle Cycles	49.3%	76.7%	76.7%	76.3%
Cycles Per Instruction	3.44	7.06	7.06	6.95
Instructions Executed	442 908	2 669 085	457 667	467 169
Operations Executed (including micro ops)	451 162	3 097 272	469 583	484 690

* Disregarding the first control loop

This control algorithm showed very different results, with an average execution time of around 2 milliseconds and a worst case of 6.57milliseconds for the first control loop execution, dangerously close to the 12 milliseconds designed control window when it is considered that this simulation only tested the control thread performance with no other processes being performed, which will surely not be the case in an UAV embedded system. Another worrying result is the high percentage of CPU idle cycles with an average of approximately 75%, this indicates that the CPU is most likely waiting for data to conclude

its instructions for a good part of the control program execution.

Table 11 – Memory Data from H2/H∞ program

H2/H∞ Memory Data	min	max	max*	avg
Memory BUS Utilization	16.69%	23.10%	23.10%	22.83%
RAM READ	1.84 MB	2.38 MB	1.93 MB	1.87 MB
RAM WRITE	0.20 MB	2.08 MB	0.21 MB	0.22 MB

* Disregarding the first control loop

The simulation results for memory utilization are shown in Table 11. Here it is clear that the memory BUS utilization was high during every control loop executed, with an average of approximately 23% and reads/writes averages reaching the megabyte scale, this is worrying since RAM is significantly slower and this high usage could be causing the high amount of CPU idle cycles mentioned before.

In regarding the control program executions, gem5 simulation results contains the types of instructions that were executed during its course, as presented in Table 12.

Table 12 – Instruction Count from H2/H∞ program

Instruction Type	Total Count	Percentage
IntAlu	24 630 915	50.92%
IntMult	105 433	0.22%
FloatAdd	4 052 400	8.38%
FloatMult	5 532 600	11.44%
FloatMultAcc	916 400	1.89%
MemRead	9 229 763	19.08%
MemWrite	3 681 940	7.61%

For this algorithm there was a significantly higher amount of floating point operations when compared to LQR results, most likely due to the feedforward implementation of this program, where multiple matrices are calculated every control loop. This coupled with the relatively poor temporal results showed in Table 10 suggests that hardware floating point calculations are probably necessary for this control program to perform adequately on the Raspberry Pi 3 B+ CPU.

5.2.1 Comparison with Raspberry Pi results

Similarly to the procedures for the LQR analysis, some tests were made on the Raspberry Pi board to measure the control loops execution time mismatch between simulation and real hardware, shown in figure 17.

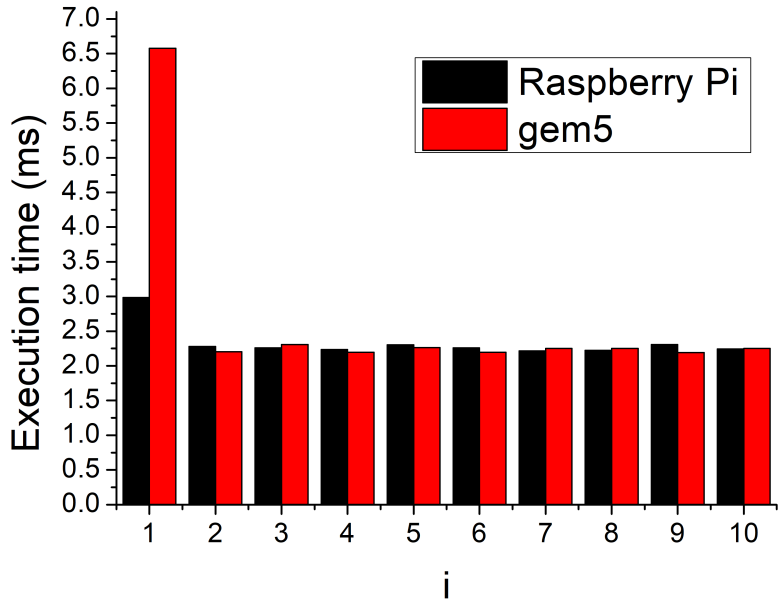


Fig. 17 – Execution time of simulation and physical hardware for the H2/H∞ .

These results were used to further check whether gem5 was accurate when used to measure execution time, since the control program is significantly more complex than the LQR. Other than a big mismatch on the first execution time of 117%, gem5 was accurate during the rest of the control loops for the H2/H∞ control program. For these 10 initial control loops the mismatch between hardware and simulation was of 4.8%.

Additionally some tests were made on the Raspberry Pi, using the perf tool available for Linux distributions, and the results compared to the ones obtained in gem5 simulation, shown in Table 13.

Table 13 – Perf and gem5 comparison for the H2/H∞ program

gem5 and perf Comparison	Perf	gem5
Execution time	2.268 ms	2.274 ms (+0.2%)
CPU Cycles	3 165 522	3 185 265(+0,6%)
CPU Instructions	458 636	470 062 (+2.5%)
L1 dcache Loads	124 060	114 127 (-8%)
L1 dcache Misses	8071	14 919 (+85%)
L2 cache loads	50 784	40 837 (-20%)
L2 cache Misses	30 390	30 843 (+1.5%)

The results from perf, running on the Raspberry Pi, and gem5, running on the host machine, offered similar results for program performance overall, with an exception of the level 1 data cache misses, which had a big mismatch of around 85%.

5.3 ADAPTIVE MIXING ALGORITHM RESULTS

The last control program tested was an Adaptive Mixing algorithm, the results obtained from the execution of the first 100 control loops are available in Table 14.

Table 14 – CPU Data from Adaptive Mixing program

AdapMix CPU Data	min	max	max*	avg
Execution time	159 us	463 us	298 us	172 us
Idle Cycles	9.42%	31.99%	30.42 %	9.70%
Cycles Per Instruction	1.66	2.42	2.36	1.68
Instructions Executed	134 120	266 482	177 262	135 898
Operations Executed (including micro ops)	167 519	324 880	220 991	169 654

* Disregarding the first control loop

It is clear that in regard to execution time, this control algorithm was kind of a middle term between the LQR and H2/H∞. The average execution time was of 172 microseconds, well withing the 12 milliseconds designed control window and it also had a relatively small percentage of CPU idle cycles, with an average of around 10%. The memory utilization for this control program is shown in Table 15.

Table 15 – Memory Data from Adaptive Mixing program

AdapMix Memory Data	min	max	max*	avg
Memory BUS Utilization	0%	10.24%	4.90%	0.15%
RAM READ	0 B	84.28 KB	54.25 KB	1.43 KB
RAM WRITE	0 B	113.25 KB	6.70 KB	1.21 KB

* Disregarding the first control loop

As suspected, the memory BUS utilization was low for the adaptive mixing algorithm, with reads and writes averages on the kilobytes scale. Similarly to the LQR control program the worst case was during the first control loop and during the rest of the control loops it was common to have zero DRAM BUS utilization, suggesting that Raspberry cache was sufficient for this control program data profile inside the basic control loop.

On a program overview analysis, it is interesting to check what types of CPU instructions were executed during the simulation of this control program, this data is presented in Table 16.

For this control program there were few floating point instructions, mostly it consisted of ALU and memory instructions. This data coupled with the relatively good temporal performance showed in Table 14 suggests that perhaps hardware floating point calculations

Table 16 – Instruction Count from AdapMix program

Instruction Type	Total Count	Percentage
IntAlu	8 501 492	48.58%
IntMult	146 011	0.83%
FloatAdd	39500	0.23%
FloatMult	83 800	0.48%
FloatMultAcc	1 560 600	8.92%
MemRead	4 726 114	27.01%
MemWrite	2 092 529	11.96%

are not a necessity for this control program to execute well inside it's 12 milliseconds designed control window.

5.3.1 Comparison with Raspberry Pi results

Similarly to the procedures for the LQR and H2/H∞ analysis, some tests on the Raspberry Pi board were made to measure the control loops execution time mismatch between simulation and real hardware, shown in figure 18.

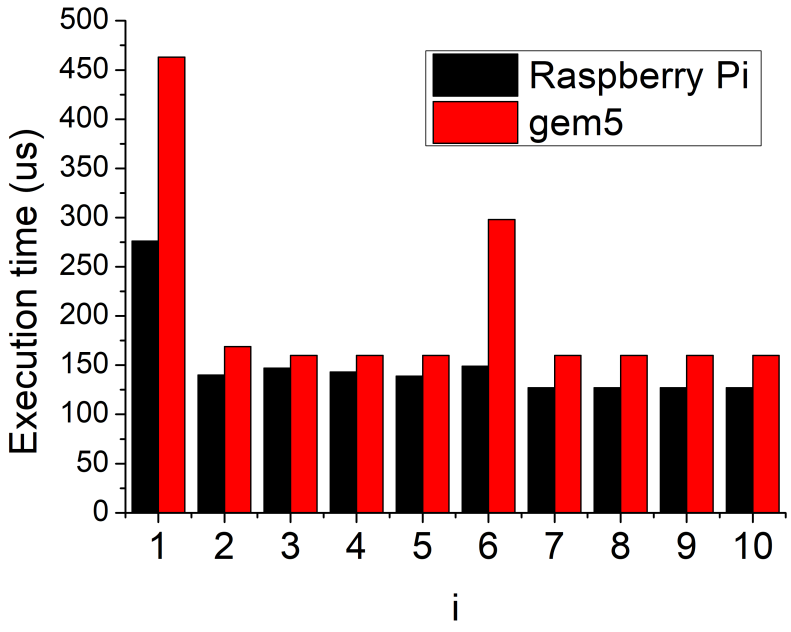


Fig. 18 – Execution time of simulation and physical hardware for the AdapMix .

For this adaptive mixing algorithm there was a higher mismatch between execution time of gem5 simulation and of the Raspberry Pi board. For the two control programs tested before only the first control loop had a big mismatch, but for this control program the first and sixth control loops had significantly high mismatches. For these 10 initial control loops the mismatch between simulation and hardware was of 36%.

Additionally some tests on the Raspberry Pi were made, using the perf tool available for Linux distributions, and compared to the results obtained in gem5 simulation, shown in Table 17.

Table 17 – Perf and gem5 comparison for the AdapMix program

gem5 and perf Comparison	Perf	gem5
Execution time	153.3 us	172.5 us (+12%)
CPU Cycles	213 282	241 587 (+13%)
CPU Instructions	121 526	140 277 (+15%)
L1 dcache Loads	57 856	65 321 (+13%)
L1 dcache Misses	1 225	2 432 (+98%)
L2 cache loads	4 645	2 846 (-40%)
L2 cache Misses	53	58 (+10%)

Most of the data extracted from perf running on the Raspberry Pi was similar to the results found in gem5 simulation, with the exclusion of level 1 data cache misses and level 2 cache loads.

5.4 SOFTWARE PREFETCHING EXPERIMENTS

One of the problems from the use of cache memory that is commonly discussed in literature is the fact that it is not possible to achieve determinism, regarding performance, in the execution of a computer program since there is no control in which data is stored or released from the cache during execution. However simply not using the cache memory most of the times is not an option, given the increasing complexity of the processed algorithms (like control algorithms). The performance impact due the lack of the cache memory on execution time was than verified.

Therefore some tests on the gem5 simulator were made, so that the system cache was reduced to the minimum amount possible in gem5, which is 64 Bytes total for level 1 cache (32 Bytes for Instruction Cache and 32 Bytes for Data Cache) with no level 2 cache. The algorithms that performed worse in the plain system (H2/H∞ and AdaptiveMixing) were tested. Thereby 100 control loops were executed to verify the impact of virtually removing the cache memory, and the results are shown in Table 18.

The main goal behind these tests was to emulate a situation where the control program did not use the cache memory. Thereby it would cope with claims from the Real-Time Systems literature, that the cache memory should not be used to favor timing determinism. So we provided a performance estimation related with the impact caused by not using the cache memory.

The results obtained from gem5 simulation showed a major impact on performance when the cache memory was virtually removed. Besides the very noticeable impact on

Table 18 – gem5 Performance Comparison with no cache*

H2/H∞		
	Default Raspberry 3 Cache	32B+32B L1
Exec Time	227.4 ms	842.13 ms (+270%)
Idle Cycles	75.73%	86.41%
RAM READ	187.55 MB	666.60 MB
RAM WRITE	22.18 MB	39.54 MB
AdapMix		
Exec Time	17.25 ms	360.9 ms (+1992%)
Idle Cycles	9.70%	88.93%
RAM READ	0.3091 MB	258.99 MB
RAM WRITE	0.3349 MB	11.94 MB

* - 64 Bytes level 1 cache due to gem5 limitations

execution time which increased 270% for the H2/H∞ and approximately 2000% for the AdaptiveMixing, it is simple to notice an increase to CPU idle cycles and major impact on the DRAM memory usage, particularly for the AdaptiveMixing control program.

To increase a program performance determinism while still using a cache memory, multiple solutions can be pursued, some of them are listed below:

- ScratchPad Memory
- Processing In Memory
- Software Prefetching

Software Prefetching was implemented in the control algorithms through a builtin prefetch function in the C++ compiler called *__builtin_prefetch*. This function can have 3 inputs but only requires one, first it needs the address of the variable being prefetched, then if the programmer wants to specify the prefetch function there are two additional inputs. The second input of the function is meant to specify whether the prefetch call was meant to prepare the prefetch for a read or write, 0 is used for read and is the default value and 1 is used for writes. The third input is used to determine what to do with the data after it is accessed in the cache, this input accepts the following values:

- 0: None, the data can be removed from the cache after the access.
- 1: Low, L3 cache, leave the data in the L3 cache level after the access.
- 2: Moderate, L2 cache, leave the data in L2 and L3 cache levels after the access.
- 3 (default): High, L1 cache, leave the data in the L1, L2, and L3 cache levels after the access.

For these tests prefetching was applied to the matrices and vectors slightly before they were needed for calculations for the control algorithm, as exemplified in listing 5.1.

```
1 __builtin_prefetch (&Lin, 0, 1);
2 __builtin_prefetch (&M, 0, 1);
3 ...
4 Linplus = (Lin.transpose()*Lin).inverse()*Lin.transpose();
5 uref = Linplus*(M*qrefddot + (C + Lfr)*qrefdot + G);
6 ...
```

Listing 5.1 – C++ built in function being used to prefetch matrices for future operations

Several variations of different placements of the prefetch functions in the code and different values for the third input were tested but did not show any significant impact in program performance, the tests were conducted on the gem5 simulation plaftform and on the Raspberry Pi 3 B+. To verify that the built in prefetch function was working correctly some tests were executed using a binary search algorithm with and without the prefetch function, these tests were executed on gem Raspberry Pi board and showed that the function was working correctly.

We could conclude that this implementation of Software Prefetching did not offer impactful improvements in performance or memory usage for the control programs tested. The control programs with software prefetching alternatives are also available in this project repository.

5.5 CORTEX-A72 SIMULATION

Besides these tests, some additional simulations were performed using a faster CPU as the hardware baseline, more specifically the Cortex-A72, which is used in the recently launched Raspberry Pi 4. Its specifications and main differences in comparison with the Cortex-A53 were presented in Table 5 and 6 in section 4.2, but mainly consists of a different type of CPU execution coupled with an expanded pipeline, a slight increase in CPU frequency and a bigger cache. With the change of the CPU type of execution on Cortex-A72 , and since the HPI CPU model used for previous tests was meant to simulate an In-Order processor, the tests on this section were conducted with the O3 CPU model.

5.5.1 LQR Analysis

For the LQR control program, as was expected, the difference in performance was minimal, as the program already performed very well both in resource utilization and execution time.

Table 19 – gem5 LQR Simulation Overview in Cortex-A72

LQR Overview	min	max	max*	avg
Execution Time	1 us (-0%)	25 us (-16%)	2 us (-0%)	1.3 us (-15%)
Idle Cycles	1.82% (-13%)	27.75% (+22%)	27.75% (+28%)	2.34% (-5%)
CPI	1.54 (+1%)	2.22 (-6%)	2.22 (+4%)	1.56 (-1%)
RAM Read	0 B (-0%)	3.7 KB (+3%)	0.38 KB (+52%)	0.04 KB (-0%)
RAM Write	0 B (-0%)	1.1 KB (+10%)	1.1 KB (+1.1KB)	0.01 KB (-0%)

* Disregarding the first control loop

The slight increase in DRAM utilization could be caused by the change of the CPU type of execution, considering that Out of Order CPUs normally uses a Reorder Buffer (ROB) that tracks the state of instructions in the pipeline.

5.5.2 H2/H∞ Analysis

The H2/H∞ control program showed great performance improvement when simulated on the Cortex-A72 architecture, with an average execution time improvement of 29%. This makes the control program fit even better into the designed 12 milliseconds control window when we take into consideration that the control thread will compete for hardware resources with the other Embedded System threads.

Table 20 – gem5 H2/H∞ Simulation Overview in Cortex-A72

H2/H∞ Overview	min	max	max*	avg
Execution Time	1.70 ms (-26%)	4.99 ms (-32%)	1.78 ms (-29%)	1.75 ms (-29%)
Idle Cycles	39.04% (-26%)	72.30% (-6%)	72.30% (-6%)	71.90% (-5%)
CPI	2.79 (-23%)	5.84 (-21%)	5.84 (-21%)	5.75 (-21%)
RAM Read	1.70 MB (-8%)	1.93 MB (-23%)	1.81 MB (-6%)	1.73 MB (-8%)
RAM Write	0.15 MB (-33%)	2.13 MB (-2%)	0.18MB (-16%)	0.18 MB (-22%)

* Disregarding the first control loop

Interestingly, the H2/H∞ control program showed a small reduction in DRAM usage when simulated in the Cortex-A72 architecture, this is likely due to the increase in cache capacity on the Raspberry Pi 4 CPU when compared to its older model.

5.5.3 Adaptive Mixing Analysis

As expected, the Adaptive Mixing control program also showed good performance improvement when simulated on the Cortex-A72 architecture but less than when compared to the H2/H∞ algorithm.

Interestingly this control program also showed an increase in DRAM data writes, which again could be caused by the change in CPU type of execution and the Reorder Buffer introduction.

Table 21 – gem5 Adaptive Mixing Simulation Overview in Cortex-A72

AdapMix Overview	min	max	max*	avg
Execution Time	144 us (-10%)	391 us (-16%)	189 us (-57%)	148 us (-14%)
Idle Cycles	7.6% (-24%)	23.53% (-36%)	14.4% (-111%)	8.0% (-21%)
CPI	1.61 (-3%)	2.11 (-15%)	2.03 (-16%)	1.62 (-4%)
RAM Read	0 KB (-0%)	63.7 KB (-32%)	24.6 KB (-120%)	1.2 KB (-16%)
RAM Write	0 KB (-0%)	123.2 KB (+8%)	9.68 KB (+44%)	1.4 KB (+16%)

* Disregarding the first control loop

5.6 MULTI-THREAD EXPERIMENTS

Additional tests were performed with the intent of checking what could negatively impact the CPU performance in the case the control algorithm had to be executed concurrently with other tasks, and how the default Linux Native POSIX Thread Library would handle a multi-thread control program. These tests consisted of a program where the same CPU core was assigned to two tasks, executing 100 control loops of the H2/H∞ algorithm and during this process also executing the Adaptive Mixing algorithm. This was implemented using *phtreads* and the CPU_ZERO and CPU_SET macros available in the <sched.h> header. There was no different priorities or special scheduling policy used on these tests, by default Linux uses the Native POSIX Thread Library implementation, which considers threads as light-weight processes.

With no special scheduler being used, the default scheduler used in Linux since 2007 for tasks of the SCHED_NORMAL class is the Completely Fair Scheduler (CFS). The CFS scheduler implementation is based on per-CPU run queues, whose nodes are time-ordered schedulable entities that are kept sorted by a kind of self-balancing binary search tree.

First these tests were executed by means of gem5 simulations. Obtained results showed was that both control algorithms performed relatively worse when they had to share the same CPU core, more noticeably the worst case execution time for both tasks was significantly longer than the performance obtained from the tests executed in 5.2 and 5.3, where the respective control program had exclusive access to the CPU. An overview of the gem5 simulation data is shown in Table 22.

Table 22 – gem5 Multitask simulation overview

H2/H∞			
	min	max	avg
Completion Time	2.19 ms (+2%)	14.45 ms (+120%)	4.62 ms (+105%)
Idle Cycles	25.9%	76.2%	53.7%
CPI	2.05	6.91	4.36
AdapMix			
Completion Time	160 us (+1%)	4.37 ms (+844%)	368 us (+114%)
Idle Cycles	9.2%	74.3%	13.9%
CPI	1.66	6.37	1.90

It is clear that when the control programs had to share the CPU with another task, the performance was negatively impacted. Both the H2/H∞ and Adaptive Mixing tasks had big performance losses when they were executed together in the multi thread program, having their average completion time increased over 100%. The worst case execution time for both control tasks also increased significantly, with the H2/H∞ surpassing the 12 milliseconds designed control window and the AdapMix having a worst case of 4.37 milliseconds, over 9 times longer when compared to the control program where it had exclusive access to the CPU.

This multi-thread control program was also tested on the Raspberry board, to verify this negative impact in completion time. In these tests the H2/H∞ thread had an average completion time of 4.55 milliseconds, with an increase of 102% from the data shown in section 5.2, where the H2/H∞ was the only task executed. The adaptive mixing also showed a negative impact in performance in the Raspberry tests, with an average completion time of 269 microseconds, a 56% higher value compared to the tests executed in 5.3.

To investigate these significant increases in the control programs threads execution times, the quantity of context switches were measured on the Raspberry Pi board with the Linux *getrusage* function - this required the inclusion of the $< sys/resource.h >$ header file. The *getrusage* function returns the resource usage measures for a target which is decided by the function first input, which can be the following alternatives:

- RUSAGE_SELF - Returns resource usage statistics for the calling process, containing the sum of all threads in the process.
- RUSAGE_CHILDREN - Return resource usage statistics for all children of the calling process.
- RUSAGE_THREAD - Return resource usage statistics for the calling thread.

Since the main focus of these simulations was to analyze the increasing in the average execution times of the control threads and the overall program resource utilization, RUSAGE_THREAD and RUSAGE_SELF were the targets used for the executed tests. The *getrusage* function returns as the result a specific structure called *rusage*, which contains some useful information about the resource utilization of the function target, some of the most relevant are:

- ru_utime - Total amount of time spent executing in user mode, expressed in a timeval structure.
- ru_nvcsw - The number of times a voluntary context switch occurred, when the process gives up the processor before its time slice was completed. This can happen when the process is waiting for data availability.

- ru_nivcsw - The number of times an involuntary context switch occurred, when the process time slice expires or another process with higher priority becomes runnable.

Obtained results showed a pattern that whenever an involuntary context switch was registered, the control thread execution time had huge negative impacts on performance (increased considerably). For the tests that consisted of the H2/H∞ and AdaptiveMix control programs running in separate threads, the H2/H∞ had an average completion time of 2.4 milliseconds for control loops with no involuntary context switch, however when there was a switch registered the average completion time increased to 12.9 milliseconds, surpassing the designed control window of 12 milliseconds.

To verify whether the multi-thread control program was being preempted by an operating system process, it is important to check the structure *ru_utime* value. The results showed very similar results between the control program CPU execution time and the sum of the two threads completion time, meaning that the worst case execution times were not generated by preemptions outside of the control program. One possibility is that the poorly optimized H2/H∞ feedforward could be causing these worst completion times, since there is big amount of matrices data to be transferred during the context switches.

It is important to investigate what caused this big difference in performance, verified both on simulation and on the Raspberry Pi board. An important statistic to check in the gem5 simulation is memory usage, shown in Table 23. It clearly shows a significant increase in DRAM usage. For the Adaptive Mixing program, the higher usage of DRAM is most likely caused by context switching between the H2/H∞ control thread.

Table 23 – gem5 Multitask memory overview

H2/H∞			
	min	max	avg
RAM Read	1.88 MB (+2%)	3.05 MB (+28%)	2.01 MB (+7%)
RAM Write	0.21 MB (+5%)	2.21 MB (+6%)	0.27 MB (+23%)
AdapMix			
RAM Read	0 MB (+0%)	3.56 MB (+4 224%)	0.15 MB (+10 641%)
RAM Write	0 MB (+0%)	1.09 MB (+886%)	0.03 MB (+2 442%)

When the simulation data for the adaptive mixing thread was analyzed, it was clear that the control loops with poor performance were the ones with higher use of the DRAM memory, as seen in figure 19 with the normalized Execution Time and DRAM Read data. Control loops with 0 DRAM Read bytes were removed, since they all had similar good performance, for better graph visibility. It is clear that the worst execution times were accompanied by higher use of the DRAM.

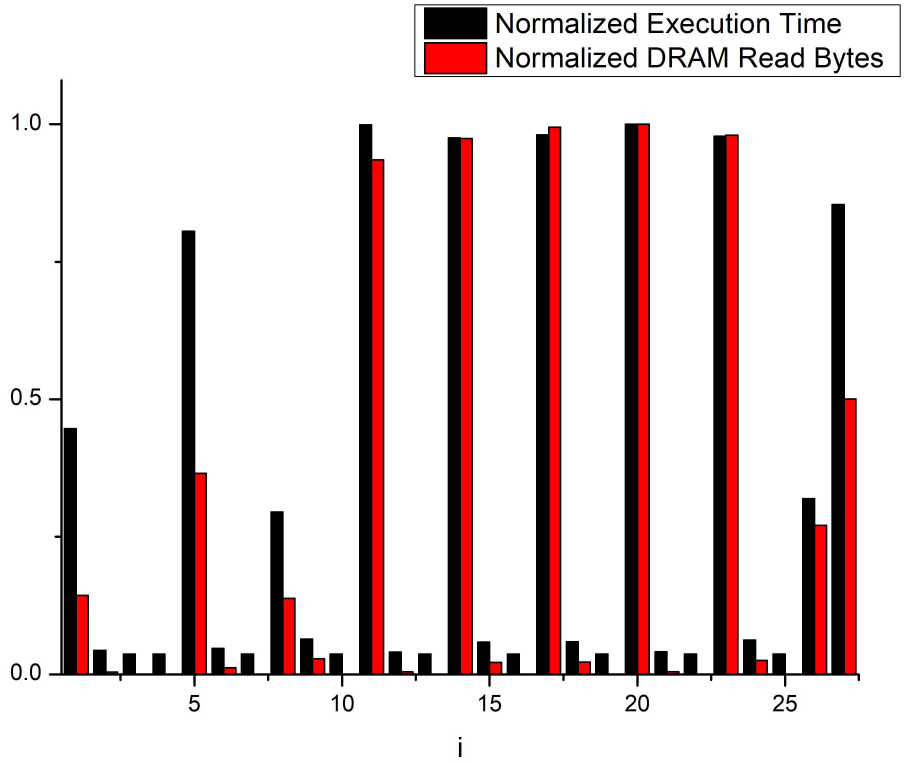


Fig. 19 – Adaptive Mixing Normalized Execution Time and DRAM Read Bytes

6 CONCLUSIONS AND FUTURE WORKS

The gem5 simulator showed to be a very valuable asset to test and determine programs characteristics, especially in regarding analyzing their time behavior on specific hardware configurations. This tool can be used for multiple ends but one that is very noticeable is the ability to test how the hardware configuration can impact a given program, so that designers can make the proper tuning to achieve the envisioned performance levels.

Regarding the Google Colab implementation, it can provide new users a fairly simpler way to execute simulations, so that they can perform these simulations in any machine, regardless the Operating System, simply using a Web-browser. It also speeds up the simulation process significantly through the use of checkpoints, although this method could also be used in a host machine in case the user knows how to create and restore it appropriately.

Given the obtained results, it is also possible to conclude that gem5 was able to evaluate design alternatives for projects using the ARM architecture. Most of the simulation results presented good indicatives of the programs' performance and are certainly of interest during many phases of a project life, mostly during earlier stages, when it can help to adjust the hardware configuration needed to execute the embedded system code in a timely manner.

In section 5.6 it was possible to verify yet another viable use of the gem5 simulator, in this section we explored its use for testing the Linux Native POSIX Thread Library handling of a multi-thread control program, the results showed that gem5 could be used to verify its default scheduling policy impact on performance with the tested control programs.

Overall, it was possible to observe that the Raspberry Pi was able to execute all three control programs well under the 12 milliseconds deadline established when they had exclusive access to its resources. However, as shown in the tests executed in 5.6, where another control program was competing for its resources, deadline misses were observed, both in simulations and in the real hardware tests. Such results do not mean, necessarily, that the Raspberry Pi is not proper for executing this type of control algorithms. One possible conclusion is that these results could also mean that the control programs should be improved to better suit a high concurrency scenario. For instance, since the algorithms were developed with the Gazebo simulation in mind, some matrices were directly imported from MATLAB and, in a more optimized implementation, they could be simplified.

6.1 FUTURE WORKS DIRECTIONS

There are numerous studies that could be explored and expanded with the contents that were discussed in this master thesis, some of these are listed below :

- Evaluate the impact of changing the ISA on the performance results of the developed

control programs.

- Explore the use of the gem5-gpu (POWER et al., 2015). Nowadays it is not too expensive to include GPUs on embedded platforms and there are some studies on how to better use the combination of CPU and GPU to improve performance.
- Explore other results that can be generated through the gem5 simulation, such as the CPU pipeline visualization through the use of the Konata tool.
- Explore the use of customized OS to reduce the amount of background processes and to better isolate the performance of the program-under-interest. One possibility is to remove the user space programs from the OS and create a custom disk image.
- Explore the use of gem5 to determine system scalability through simulation of other processes competing for CPU against the control programs.
- Further tests regarding the handling of a multi-thread control program, exploring threads priorities and different scheduling policies.
- Improve the Google Colab implementation, through additional ISA precompiled gem5 binaries and different CPU models checkpoints for example.
- Further explore the studies of verifying if the gem5 simulation results are reliable (GUTIERREZ et al., 2014) when compared to results of tests ran on physical hardware, most likely trying different instruction architectures and different styles of programs such as memory intensive and CPU intensive programs. It would also be interesting to use other available more precise memory models in the gem5 simulation tool and check the outcome on simulation results.
- Use sensor data from the Gazebo (KOENIG; HOWARD, 2004) simulation environment to diversify UAV states and with that increase coverage of the control program, resulting in a more comprehensive performance study .

BIBLIOGRAPHY

7-CPU. Cortex A53 Benchmark. Acessado em fevereiro 2022 em https://www.7-cpu.com/cpu/Cortex-A53.html.

AKRAM, A. **A Study on the Impact of Instruction Set Architectures on Processor's Performance**. 2017. Diss. (Mestrado) – Western Michigan University.

AKRAM, A.; SAWALHA, L. Validation of the gem5 Simulator for x86 Architectures. In: p. 53–58.

AMINIFAR, A. **Analysis, Design, and Optimization of Embedded Control Systems**. Linköping University, 2016.

BAI, L.; WANG, J.; HUANG, B. The Effect of Shared L2 Cache on Determinism in Airborne Embedded System. In: 2016 12th International Conference on Computational Intelligence and Security (CIS). 2016. P. 697–700.

BINKERT, N. et al. The gem5 simulator. **ACM SIGARCH Computer Architecture News**, Association for Computing Machinery (ACM), v. 39, n. 2, p. 1–7, mai. 2011.

BUTKO, A.; GARIBOTTI, R.; OST, L.; SASSATELLI, G. Accuracy evaluation of GEM5 simulator system. In: 7TH International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). IEEE, jul. 2012.

CARDOSO, D. N.; ESTEBAN, S.; RAFFO, G. V. A new robust adaptive mixing control for trajectory tracking with improved forward flight of a tilt-rotor UAV. **ISA transactions**, 2020.

CARDOSO, D. N.; RAFFO, G. V.; ESTEBAN, S. A robust adaptive mixing control for improved forward flight of a tilt-rotor UAV. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC). 2016. P. 1432–1437.

DESTELLE, M.; DUFOUR, J.-L. Deterministic scheduling reconciles cache with preemption for WCET estimation. In: ERTS2 2010, EMBEDDED REAL TIME SOFTWARE & SYSTEMS. Toulouse, France, mai. 2010. Disponível em: <https://hal.archives-ouvertes.fr/hal-02267646>.

DONADEL, R. **Modeling and Control of a Tiltrotor Unmanned Aerial Vehicle for Path Tracking**. 2015. Diss. (Mestrado) – Federal University of Santa Catarina.

DONADEL, R.; ALMEIDA NETO, M. M. de; RAFFO, G. V.; BECKER, L. PATH TRACKING CONTROL OF A SMALL SCALE TILTROTOR UNMANNED AERIAL VEHICLE. In:

FARSHCHI, F.; VALSAN, P.; MANCUSO, R.; YUN, H. Deterministic Memory Abstraction and Supporting Cache Architecture for Real-Time Systems, jul. 2017.

GEM5. **gem5 Linux image**. Ago. 2021. Disponível em: <http://dist.gem5.org/dist/current/arm/aarch-system-20180409.tar.xz>.

GONÇALVES, F. S.; RAFFO, G. V.; BECKER, L. B. Managing CPS Complexity: Design Method for Unmanned Aerial Vehicles. **IFAC-PapersOnLine**, v. 49, n. 32, p. 141–146, 2016. Cyber-Physical Human-Systems CPHS 2016. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405896316328762>.

GRACIOLI, G.; ALHAMMAD, A.; MANCUSO, R.; FRÖHLICH, A. A.; PELLIZZONI, R. A Survey on Cache Management Mechanisms for Real-Time Embedded Systems. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 48, n. 2, nov. 2015. Disponível em: <https://doi.org/10.1145/2830555>.

GUTIERREZ, A.; PUSDESRIS, J.; DRESLINSKI, R. G.; MUDGE, T.; SUDANTHI, C.; EMMONS, C. D.; HAYENGA, M.; PAVER, N. Sources of error in full-system simulation. In: 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 2014. P. 13–22.

JOHNSON, A.; CUNNINGHAM, C.; ARNOLD, E.; ROSAMOND, W.; ZÈGRE-HEMSEY, J. Impact of Using Drones in Emergency Medicine: What Does the Future Hold? **Open Access Emergency Medicine**, Volume 13, p. 487–498, nov. 2021.

KOENIG, N.; HOWARD, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). 2004. v. 3, 2149–2154 vol.3.

LATHI, B. P. Linear systems and signals. In: Oxford University Press, 2005. 9 - Time-Domain Analysis of Discrete-Time Systems.

LEE, B. Dynamic Branch Prediction. Acessado em janeiro 2020 em http://web.engr.oregonstate.edu/ benl/Projects/branch_pred/.

LESAGE, B.; HARDY, D.; PUAUT, I. Shared Data Caches Conflicts Reduction for WCET Computation in Multi-Core Architectures. **18th International Conference on Real-Time and Network Systems**, nov. 2010.

LIMA, J. P. C. de. PIM-gem5 : a system simulator for Processing-in-Memory design space exploration. In:

LLORCA-CEJUDO, R.; FRANDON, O. Cache-Induced Execution Time Variability of a Satellite On-Board Software in a LEON-2 Microprocessor. In: OUWEHAND, L. (Ed.). **DASIA 2011 - Data Systems In Aerospace**. Ago. 2011. v. 694. (ESA Special Publication), p. 54.

MACRO-UFMG. Mechathronics, Control, and Robots - Universidade Federal de Minas Gerais. Acessado em janeiro 2022 em http://macro.ppgee.ufmg.br/robots.

MELLO, L. S. MODELAGEM E CONTROLE DE CORPO COMPLETO DE UM MANIPULADOR AEREO. Acessado em janeiro 2020 em https://adorno.eng.ufmg.br/students/modelagem-e-controle-de.pdf, 2016.

PEDRONI, V. **Circuit Design and Simulation with VHDL**. MIT Press Ltd, 17 set. 2010. 632 p.

POWER, J.; HESTNESS, J.; ORR, M. S.; HILL, M. D.; WOOD, D. A. gem5-gpu: A Heterogeneous CPU-GPU Simulator. **IEEE Computer Architecture Letters**, v. 14, n. 1, p. 34–36, 2015.

QUIGLEY, M. ROS: an open-source Robot Operating System. In: ICRA 2009. 2009.

RAFFO, G. V.; ORTEGA, M. G.; RUBIO, F. R. An integral predictive/nonlinear H∞ control structure for a quadrotor helicopter. **Automatica**, Elsevier BV, v. 46, n. 1, p. 29–39, jan. 2010.

REGO, B. S.; RAFFO, G. V. Suspended load path tracking control based on zonotopic state estimation using a tilt-rotor UAV. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC). 2016. P. 1445–1451.

RESEARCH, P. M. Commercial UAV Market Analysis. Acessado em janeiro 2022 em https://www.unmannedairspace.info/latest-news-and-information/commercial-uav-market-to-reach-usd15-62-billion-by-2026-polaris-market-research/.

RYAN, A.; HEDRICK, J. A mode-switching path planner for UAV-assisted search and rescue. In: PROCEEDINGS of the 44th IEEE Conference on Decision and Control. 2005. P. 1471–1476.

SAARI, H.; PELLIKKA, I.; PESONEN, L.; TUOMINEN, S.; HEIKKILÄ, J.; HOLMLUND, C.; MÄKYNEN, J.; OJALA, K.; ANTILA, T. Unmanned Aerial Vehicle (UAV) operated spectral camera system for forest and agriculture applications. **Proc SPIE**, v. 8174, out. 2011.

SAM; XI; YAO, Y.; BHARDWAJ, K.; WHATMOUGH, P.; WEI, G.-Y.; BROOKS, D. **SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads**. Dez. 2019.

SHAO, Y. S.; XI, S. L.; SRINIVASAN, V.; WEI, G.-Y.; BROOKS, D. Co-designing accelerators and SoC interfaces using gem5-Aladdin. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). 2016. P. 1–12.

SHINSEL, A. Understanding the Instruction Pipeline. Acessado em janeiro 2020 em https://techdecoded.intel.io/resources/understanding-the-instruction-pipeline.

SILVANO, F. **PROJETO DA ARQUITETURA DE SOFTWARE EMBARCADO DE UM VEíCULO AEREO NãO TRIPULADO**. 2014. Diss. (Mestrado) – Federal University of Santa Catarina.

SILVESTRE, I.; BECKER, L. Performance Analysis of Embedded Control Algorithms used in UAVs. In: ANAIS Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais. Evento Online: SBC, 2020. P. 180–185. Disponível em: <https://sol.sbc.org.br/index.php/sbesc_estendido/article/view/13110>.

SILVESTRE, I.; FILHO, A. B.; BECKER, L. Using gem5 Simulator to Support Design Space Exploration Targeting ARM Architecture. In: ANAIS do XI Simpósio Brasileiro de Engenharia de Sistemas Computacionais. Evento Online: SBC, 2021. P. 40–47. Disponível em: <https://sol.sbc.org.br/index.php/sbesc/article/view/18468>.

SILVESTRE, I. O. **gem5 Colab Video Guide**. Ago. 2021a. Disponível em: <https://www.youtube.com/channel/UCuMtLsiR-Amuw9AizYQLCnw>.

SILVESTRE, I. O. **gem5 Prototype in Colab**. Ago. 2021b. Disponível em: <https://colab.research.google.com/github/iagosilvestre/gem5Colab/blob/master/SBESC_gem5_Prototype.ipynb>.

SILVESTRE, I. O. **ProVANT Control Programs**. Ago. 2021c. Disponível em: <https://github.com/iagosilvestre/ProVANT-Control-Test-gem5>.

TASHIRO, R.; OYAMADA, M. S. An Environment for Design Space Exploration Using gem5-McPAT. In: 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE, nov. 2016a.

TASHIRO, R.; OYAMADA, M. S. An Environment for Design Space Exploration Using gem5-McPAT. In: 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC). 2016b. P. 220–225.

TOUSI, A.; ZHU, C. Arm Research Starter Kit: System Modeling using gem5. Acessado em janeiro 2020 em
https://developer.arm.com/solutions/research/research-enablement-kits, 2017.

VILLANUEVA, J. M. M. Estabilidade de Sistemas Discretos. Acessado em janeiro 2020 em http://www.cear.ufpb.br/juan/wp-content/uploads/2019/06/Aula-4.-Estabilidade-e-Transformações-1.pdf,
2019.

WINKLER, C. SENSOR SOLUTIONS PLAY CRITICAL ROLES IN ENABLING INNOVATION IN DRONES. Acessado em janeiro 2020 em
http://www.memsic.com/about-memsic/publications.cfm, 2016.

YASSIN, Y.; JAHRE, M.; KJELDSBERG, P.; AUNET, S.; CATTHOOR, F. Fast and Accurate Edge Computing Energy Modeling and DVFS Implementation in GEM5 Using System Call Emulation Mode. **Journal of Signal Processing Systems**, v. 93, jan. 2021.

ZARGHAM, M. R. **Computer Architecture**. Prentice Hall, 1996.