PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

ESTEVAN LINCK LARA

# A HARDWARE-BASED APPROACH TO GUARANTEE CRITICAL TASK SCHEDULABILITY IN TDMA BUS ACCESS OF MULTI-CORE ARCHITECTURE

Porto Alegre
2020

PÓS-GRADUAÇÃO - *STRICTO SENSU*


Pontifícia Universidade Católica
do Rio Grande do Sul

**PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL**
**SCHOOL OF TECHNOLOGY**
**POST GRADUATE PROGRAM IN ELECTRICAL ENGINEERING**

# A HARDWARE-BASED APPROACH TO GUARANTEE CRITICAL TASK SCHEDULABILITY IN TDMA BUS ACCESS OF MULTI-CORE ARCHITECTURE

**ESTEVAN LINCK LARA**

Thesis presented as partial requirement for obtaining the degree of Master in Electrical Engineering at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Dr. Fabian Luis Vargas

**Porto Alegre**
**2020**

# Ficha Catalográfica

L318h Lara, Estevan Linck

A Hardware-Based Approach to Guarantee Critical Task Schedulability in Tdma Bus Access of Multi-Core Architecture / Estevan Linck Lara. – 2020.
88.
Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Elétrica, PUCRS.

Orientador: Prof. Dr. Fabian Luis Vargas.

1. Multi-core processor. 2. Critical application. 3. High-performance embedded system. 4. Critical task schedulability. 5. Worst-Case Execution Time. I. Vargas, Fabian Luis. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

Pontifícia Universidade Católica do Rio Grande do Sul
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA - PPGEE

# A HARDWARE-BASED APPROACH TO GUARANTEE CRITICAL TASK SCHEDULABILITY IN TDMA BUS ACCESS OF MULTICORE ARCHITECTURE

**CANDIDATO: ESTEVAN LINCK LARA**

Esta Dissertação de Mestrado foi julgada para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica do Rio Grande do Sul.

**DR. FABIAN LUIS VARGAS - ORIENTADOR**

**BANCA EXAMINADORA**

**DR. JARBAS ARYEL NUNES DA SILVEIRA - PPGETI - UFC**

**DR. TIAGO ROBERTO BALEN - PGMICRO - UFRGS**

**DR. CÉSAR AUGUSTO MISSIO MARCON - PPGEE - PUCRS**

PUCRS
Av. Ipiranga, 6681 - Predio 32 - Sala 507 | CEP 90619-900 | Porto Alegre, RS - Brasil
Fone: (51) 3320-3540 | E-mail: engenharia.pg.eletrica@pucrs.br | www.pucrs.br/politecnica

I dedicate my dissertation work to my advisor Fabian Luis Vargas, who taught me everything that was needed and patiently lit my way in this research.

"If knowledge can create problems, it is not through ignorance that we can solve them."

(Isaac Asimov)

## ACKNOWLEDGMENTS

Agradeço a todos que participaram dessa etapa de minha vida, aos amigos e colegas de laboratório que entre uma xícara ou outra de café compartilhávamos risadas. Agradeço também aos meus pais que me incentivaram e estiveram me apoiando, mesmo longe mantiveram-se presente. Principalmente ao meu orientador Fabian Luis Vargas, que mostrou-se ser um cara admirável e me instruiu da maneira correta e me iluminou nos momentos necessários, se manteve presente e interessado no tema. Gostaria também de agradecer ao colega Roger Curtinaz Goerl que me ensinou muitas coisas e sempre esteve disposto a me ajudar quando encontrava algum problema.

**RESUMO**

O uso de processadores multi-core em sistemas embarcados de uso geral em tempo real sofreu um grande aumento nos últimos anos. Infelizmente, aplicações críticas como as dedicadas ao setor aeroespacial e automotivo, por exemplo, não estão se beneficiando do alto desempenho desse tipo de processador. Existem obstáculos nessas plataformas, um deles é a imprevisibilidade e o outro é a falta de garantias sobre as propriedades em tempo real do software nesses sistemas. O barramento de memória compartilhada é a principal fonte de imprevisibilidade de temporização devido à contenção de acesso entre os núcleos. Nessas aplicações, quando uma tarefa crítica é executada, todas as outras tarefas comuns (não críticas) são temporariamente pausadas até a conclusão da tarefa crítica. O presente trabalho trata de criar uma alternativa para esse tipo de ação. A seguir, propõe-se uma abordagem com o objetivo de garantir a execução completa de uma tarefa crítica, ou seja, impedir que a execução simultânea de tarefas com diferentes criticalidades induza a violação do tempo de execução de tarefas críticas em uma plataforma multicore. Assume-se que o sistema esteja executando simultaneamente a tarefa crítica no núcleo crítico e qualquer número de tarefas (não críticas) nos núcleos restantes do sistema, com um perfil misto de criticidade. A abordagem proposta é baseada na implementação de um watchdog (denominado de Controlador de Acesso de Barramento Compartilhado, ou SBAC, do termo em inglês: "Shared-Bus Access Controller"), instanciado no barramento de endereço que conecta o processador à memória do sistema. Assim, ao monitorar o barramento de endereços da memória do sistema multi-core, o sistema proposto é capaz de prever o evento de violação do tempo de execução da tarefa crítica. A abordagem proposta pressupõe que os núcleos acessem o barramento de memória na política TDMA (Time-Division Multiplexing Access), que aloca intervalos de tempo iguais para cada um dos núcleos, permitindo o acesso à memória em diferentes intervalos de tempos. Para ajustar a execução crítica da tarefa sem ter que tomar uma medida que interrompa temporariamente os outros núcleos de executarem seus processos por um longo período. Elaborou-se modelos de acessos que garantem a execução de tarefas críticas sem violação de deadline, ao mesmo tempo que reduzem o tempo de ócio dos demais núcleos do processador. Com o

objetivo de validar esta abordagem, o watchdog foi implementado em linguagem VHDL e conectado a uma versão de dois núcleos do processador soft-core LEON 3. Todo o sistema foi simulado no Simulador ModelSim e prototipado em um FPGA comercial. Experimentos práticos demonstram que a técnica proposta é extremamente eficaz para garantir a execução correta da tarefa crítica em uma plataforma multi-core com criticalidade mista.

**Palavras Chave:** Processadores multi-core, Aplicações críticas, Sistemas embarcados de alto desempenho, Escalonamento de tarefas críticas, Pior caso de tempo de execução, Violação de tempo máximo de execução (deadline)..

# A HARDWARE-BASED APPROACH TO GUARANTEE CRITICAL TASK SCHEDULABILITY IN TDMA BUS ACCESS OF MULTI-CORE ARCHITECTURE

## ABSTRACT

The use of multi-core processors in general-purpose real-time embedded systems has experienced a considerable increase in recent years. Unfortunately, critical applications such as those devoted to aerospace and automotive, for instance, are not benefiting from the high-performance of this type of processor. The major obstacle: we may not predict and provide any guarantee on real-time properties of software running on such platforms. Shared memory bus is the main source of timing unpredictability due to access contention among cores. In these applications, when a critical task is executed, all other ordinary (non-critical) tasks are temporarily paused until completing the critical task. The present work aims to counteract with this problem. Hereafter, we propose an approach aiming to guarantee the complete execution of a critical task, i.e., preventing critical task execution from violating deadline while running in a multi-core platform. We assume that the system is simultaneously running the critical task in the Critical Core, and any number of (non-critical) tasks in the remaining cores of the system, with a mixed-criticality profile. The proposed approach is based on the implementation of a watchdog (namely, Shared Bus-Access Controller: SBAC) that is connected to the address bus between the processor and the system memory. Though, by monitoring the memory address bus of the system, the watchdog is able to predict the event of critical task execution violation. The approach assumes that bus access is based on the Time-Division Multiplexing Access (TDMA) policy, which allocates time slices for each of the cores to access the system bus. In this context, a time slice is of fixed size.To fit critical task execution into the deadline, a different number of time slices

is allocated to each task running in the system. To validate the approach, the watchdog was implemented in VHDL language and connected to a dual-core version of the LEON 3 soft-core processor. The whole system was simulated on ModelSim and prototyped into a commercial FPGA. Practical experiments have shown that the proposed technique effectively guarantees a smooth execution of a critical task simultaneously with other non-critical tasks on a multi-core platform.

**Keywords:** Multi-core processor, Critical application, High-performance embedded system, Critical task schedulability, Worst-Case Execution Time (WCET), Deadline violation.

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

AADL – Architectural Analysis and Design Language

AHB – Advanced High Bus

ALF – Artist Flow Analyses

AMBA – Advanced Micro Controller Bus Architecture

APB – Advanced Peripheral Bus

ARM – Advanced RISC Machine

ASB – Advanced System Bus

ASIC – Application Specific Integrated Circuit

BCET – Best-case Execution Time

CAN – controller Area Network

CDMA – Code Division Multiple Access

CC – Critical Core

CEP – Critical Execution Point

CFA – Control-Flow Analysis

CFG – Control Flow Graph

CPU – Central Processing Unit

CRTES – Critical Real-Time Embedded Systems

CT – Critical Task

DSU – Debug Support Unit

EDF – Earliest Deadline First

ESA – European Space Agency

FCFS – First-Come First-Served

FDMA – Frequency Division Multiple Access

GPGGA – Global Positioning System Fix Data

II-P – Infrastructure Intelectual Property

IPET – Implicit Path Enumeration

MCS – Mixed Critical Systems

MIPS – Microprocessor without interlocked pipeline stages

NC – Non-critical Core

NMEA – National Marine Electronics Association

NOC – Network-on-Chip

OCBP – Own-Criticality Based Priority

QOS – Quality-of-Service

PC – Program Counter

PT – Period Transformation

PTA – Probabilistic Timing Analysis

PWCET – probabilistic worst-case execution time

RPS – Referent Points

RTA – Response-Time Analysis

PT – Period Transformation

SBAC – Shared Bus-Access Controller

SMC – Static Mixed-Criticality

STTM – Single Time Table per Mode

SWEET – SWEdish Execution Time tool

TDMA – Time Division Multiple Access

TT – Time-Triggered

TTS – TDMA Time Slice

VHDL – VHSIC Hardware Description Language

VHSIC – VLSI for ASIC

VLSI – Very-Large-Scale Integration

WCET – Worst Case Execution Time

# CONTENTS

# 1. INTRODUCTION

First of all, it is general knowledge that the number of processors in the machines and system tend to increase in number. As the amount of cores increases, so does the complexity, for some components of the system can only be accessed by one core at a time. The waiting time for other cores can cause a disaster if the core does not finish the task before the deadline comes. This approach shows a method assuring the critical task execution before the deadline comes, preventing a disaster. One of the main obstacles to solve this problem is to predict if there will be time to execute critical tasks. Otherwise, it is necessary to know if the system could operate normally or a different way of operation should be triggered. In critical embedded systems, such as avionics systems, the way to guarantee the critical tasks is to execute in the stand-alone mode when that task is to be executed. Unfortunately, this method prevents the usage of the full performance of these systems, wasting its features. With this in mind, this approach has the intent to increase the performance of multi-core systems. In more detail, the hardware proposed is designed in such a way that it divides the permission to access the memory when identifying a critical task that can not be finished at the deadline time. Instead, the approach is based on using a dedicated Shared Bus-Access Controller (SBAC), which works in conjunction with the bus controller to allow core access to the memory bus. Based on real-time monitoring of the memory address bus by one side and on the static calculation (assuming a single-core) on the other side, the SBAC can predict if the critical task will be concluded before reaching the deadline. If the time allocated to the Critical Core is not enough at that instant, the SBAC uses memory bus time slots from the other cores and enlarges the time slot allocated to the execution of the Critical Core at the same amount. This process is performed under the control of the SBAC in conjunction with the bus access controller. In the end, therefore, implements the bus access policy determined by the SBAC. This approach's final goal is to ensure the maximum bus access time shared to all processor cores, under the condition that the Critical Core will not miss the deadline. The main features and advantages of the proposed approach, when compared to existing techniques, are listed below:

(a) The fact of concentrated critical tasks only on one core ensures a particular facility for analyzing multi-core systems. It allows the division of access to memory bus access.

(b) As opposed to some existing approaches, the proposed approach does not require studying other less important tasks than critical tasks. The SBAC is maintained according to specific code structure and timing characteristics of the critical task.

(c) This approach can be applied to any processor since some required items are supplied. The adaptation after the validation of this hardware can be tested in other multiprocessor systems different from those chosen in this work.

(d) Another aspect of this approach can be visualized after the conclusion regarding overhead in the occupation area.

Many codes were used in this work, and a repository on GitHub was created to share them. This work's main codes are provided at https://github.com/Estevanll?tab=repositories, yet the IP cores for embedded processors based on the SPARC architecture tested in this work are provided by Cobham Gailser at https://www.gaisler.com/index.php/downloads/leongrlib. To recreate this work, it is necessary some modifications and replacements of codes provided in the informed repository.

## 1.1 Objective

The purpose of this work is divided into two parts, which are listed below:

- Develop an Intellectual Infrastructure Property (II-P) called WCET/Deadline Controller to prevent critical task deadline violation in a mixed-criticality multi-core system;
- Develop a TDMA bus access approach, in which the II-P will be selected to take the place of the arbitrary vector.

## 1.2 Contributions of this work

The following items list the main contributions of this work:

(a) It minimizes the computational complexity imposed by multi-core static timing analysis: it needs only to analyze interactions between pipeline and cache models of a single-core when

executing a given critical task. All inter-core conflicts generated during the execution of the critical and the various less-critical tasks caused by their non-uniform and concurrent memory bus accesses are not taken into account by the SBAC.

(b) From the above (a) statement, it is also concluded that the proposed approach does not need development and it is not based on timing analysis models of multi-core processors. Therefore, the approach is not based on the multi-core processor model's faithfulness to guarantee a precise workload timing prediction.

(c) In contrast to the existing approaches, the proposed approach does not require any knowledge about implementing the less-critical workloads or even the number of workloads that will run concurrently with the critical task. Then, the SBAC monitors the critical task execution online and automatically switches the bus usage from the "Shared Mode" to the "Stand-alone" mode to guarantee the maximum possible processor performance with workload schedulability. If the number of less-critical task changes, there is no need to re-compute the timing analysis process for the critical task to guarantee workload schedulability. This condition is ensured because the SBAC can switch from the "Shared Mode" to the "Stand-alone mode" automatically, no matter the number of less-critical tasks is running in parallel with the critical one since the number of cores is fixed and defined by the system architecture.

(d) This approach can be applied to any processor, considering that the designer can collect two signals from the processor ("Program Counter" and "Annul"). The last signal indicates if the current instruction in the pipeline was executed or not.

(e) Given that the approach can be applied to any processor, it allows a large spectrum of real-time operating systems to be used. Thus, traditional and well-established real-time operating systems for critical applications such as VxWorks, LynxOS, Integrity, or RTEMS and their advanced versions compliant with ARINC-653 (an avionics standard for safe, partitioned systems)[40] could also be considered in the whole system design.

# 2. BACKGROUND

In general, it is known that a hard real-time system needs to satisfy the constraint timing derived from all systems they control. According to this necessity, upper bounds on the execution times are required to visualize these constraints. However, in general, these constraints are very complicated to obtain, and other methods should be used to permit the usage of applications in need of this. Whenever systems interact with people, there is a risk involved for these people, and these systems could be called Critical Real-Time Embedded Systems (CRTES). CRTES needs studies about timing analysis because it is one of many ways to increase these systems. Up until now, the answers of these systems are essential for the total system, these answers should occur in a fraction of time before the deadline, but if, in this case, the response occurs delayed, it could be disastrous. [45] [34]

## 2.1 Real-Time Systems

Today most of the processors in typical applications are not part of what we usually called computers, so many examples of this affirmation arise in our minds when we hear that. It is a consensus that computing systems like these examples are denominated Embedded System.[34] Embedded Systems that must fulfill timing constraints are called real-time systems due to some of the tasks that implement must meet deadlines. In case the task has been finalized in the expected time, correct results have been produced. Part of the design of these systems is responsible for the timing analysis of critical tasks.

Furthermore, an incredible amount of tools and software have been developed for this purpose.[34] According to the literature, Real-time systems are commonly divided into the following three categories according to the consequences of a miss-match. The worst case of these categories is hard real-time systems. The mismatch of deadline carries the system a state of full failure, taking the chance of undesirable consequences like accidents and economic crashes, among others. A primary example of the effects of delay to active some task such as a system is the control unit that triggers inflating airbags during a crash, resulting in unwanted consequences. Quantitatively, the probability of a hazard occurring in that system should be less than $(10^{-9})$. [45] [21]

Fig. 2.1 depicts another classification, and this classification contains three definitions of systems: soft real-time system, firm real-time system, and Hard real-time system. The first allows the use of results after the deadline, but the Quality-of-Service (QoS) is degraded over time. Although the second system can miss some deadlines, furthermore does not cause any catastrophic consequence. But in this case, the result produced after the deadline is discarded anyway. The third, called a hard real-time system, shows the usefulness curve representing the state of usefulness or unwanted consequence. After the deadline, any answer would have undesirable effects like an accident or some severe repercussions. In Fig. 2.1, it is observable the use of tasks concerning time described previously. [45] [34]

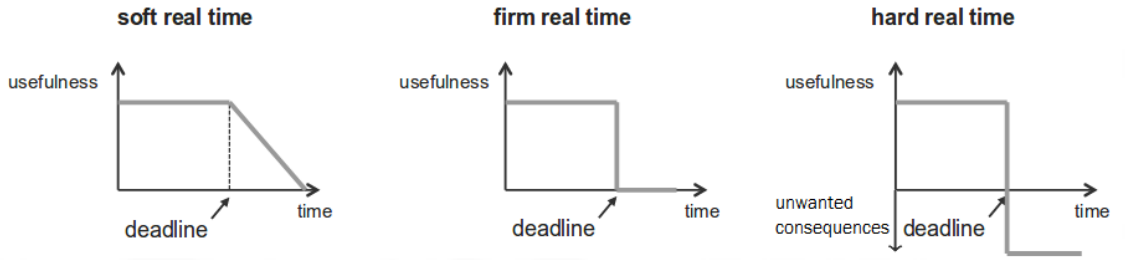


Figure 2.1 – Graphic of Usefulness by Time.
Source:Adapted of Rochange, page 13 (2014)

Therefore, it is notable for the necessity of ways to estimate the upper bounds about these systems, specifically for hard real-time systems. For this reason, advanced hardware features have been created for systems architectures that minimize the probability of failure. [14]

### 2.1.1 Upper Bounds Analysis

The worst-case execution time (WCET) and the best-case execution time (BCET) are crucial for analyzing timing constraints. These points are the opposite extreme points that a system has, but in most cases, the state space is too large to explore and find the exact value of these points. For critical systems, the most crucial issue between these two is WCET, by this mean the BCET will not be studied. In other words, the WCET refers to the maximum execution time of a particular program executing on a specific processor. These days, determining time bounds for systems is to measure the maximal observed execution time and overestimate the result. In doing so, this method is not safe for hard real-time systems, and this method is called dynamic timing analysis. [41] [45]

There were two main criteria for evaluating methods for timing analysis and their precision and safety. According to Willhelm, the literature about timing analysis has created confusion by not mentioning the WCET in significant cases, as it is estimated WCET and not thoroughly the exact value. It is necessary to make a distinction between the terminologies and take care of this misnomer.[45] According to this analysis, the WCET is typically the estimated worst-case execution time or sometimes probabilistic worst-case execution time (pWCET) when it used Probabilistic Timing Analysis (PTA). [14] Fig. 2.2 shows the significant timing bounds about execution time. Also, is noticeable the problem of the worst-case execution time by the fact the real worst-case timing is allocated after the observable. The vertical axis is allocated the distribution of the number of times that the task finishes, and in the horizontal axis is the time to execute the task.

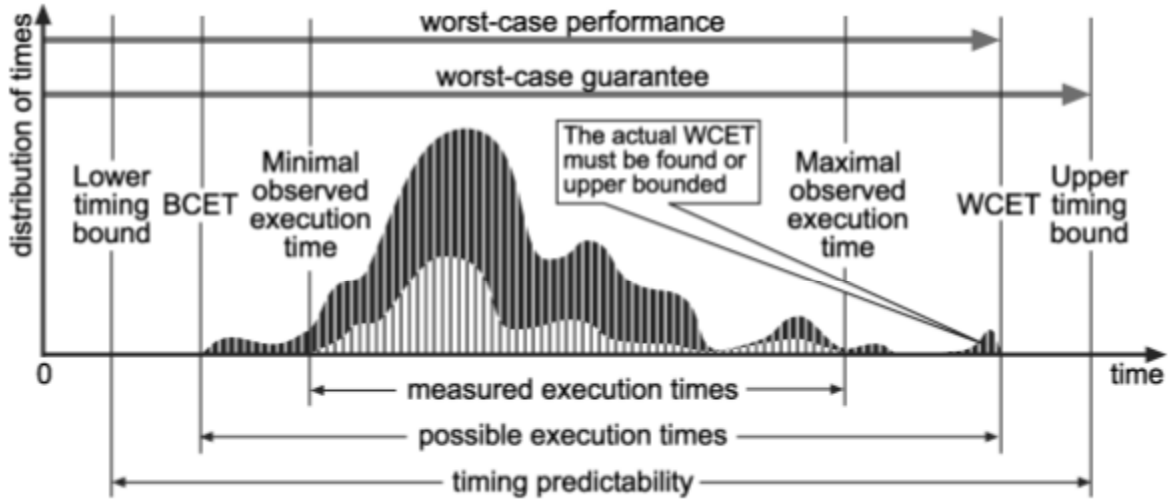


Figure 2.2 – The Worst-Case Execution-Time Problem.
Source: Wilhelm et al., page 3 (2008)

For a prior estimation of WCET is necessary certain predictability; otherwise, it is impossible to affirm with precision when the execution will finish. Thus, in many cases, it is common to increase a percent over the time estimation to guarantee the curve out of observable. [45]

#### 2.1.1.1 Static Timing Analysis

Static Methods do not rely on executing code on real hardware or a simulator. It is calculated through constraints of the system or approaches to estimate the flow operating and the adequate time to achieve it. To estimate the calculation of the WCET is necessary to combine measurements of basic blocks that represent an execution of instructions without branches. According to the literature, there

are three main classes of methods combining analytically determined or measured times to end-to-end using CFG, and they are structure-based, path-based, and techniques using implicit-path-enumeration (IPET).[45] In Fig. 2.3 it is presented these three techniques.

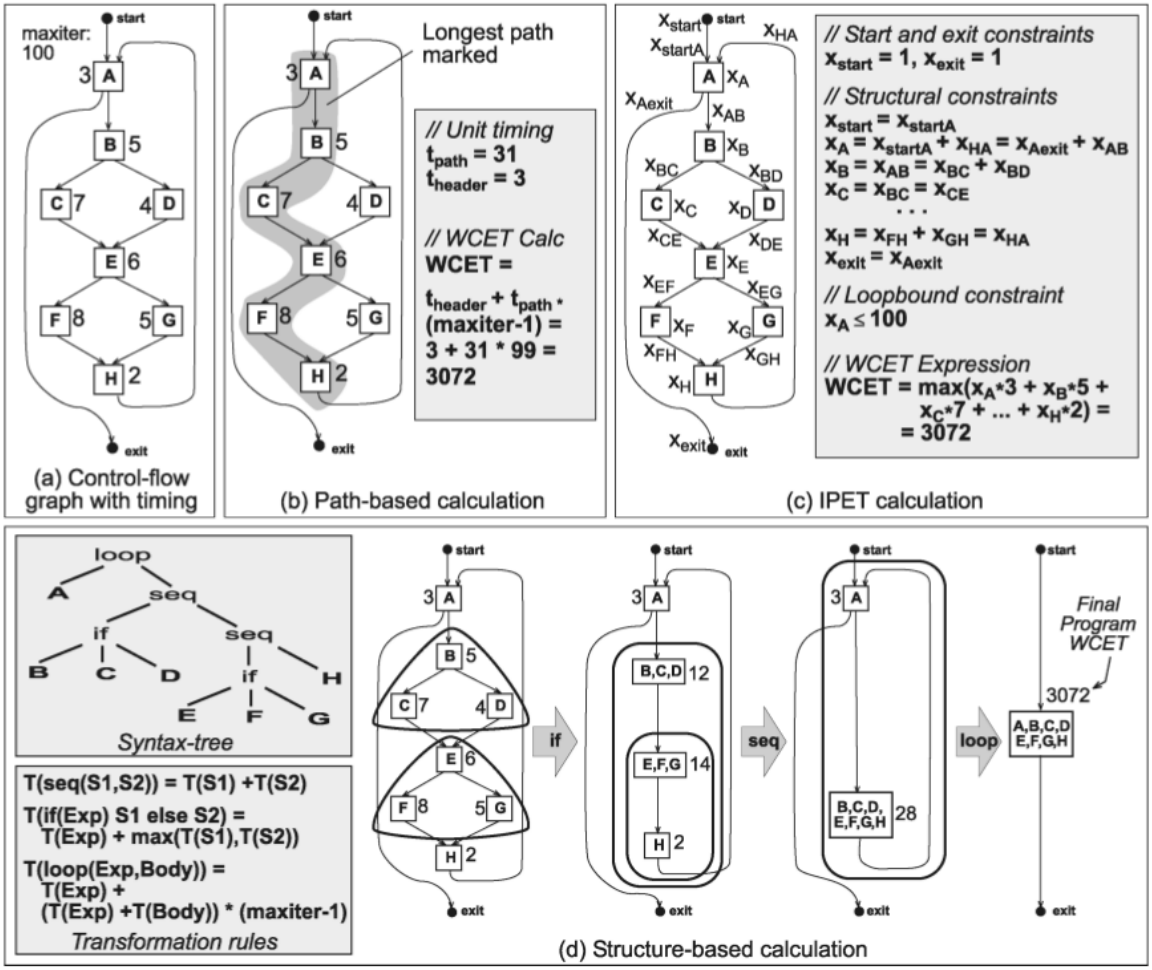


Figure 2.3 – Bound Calculation.
Source: Wilhelm et al., page 15 (2008)

One problem of the structure-based approach is that not every flow control can be expressed by the syntax tree. However, Fig. 2.3.d illustrates how a structure tree emerges to a single node representing the entire structure. Therefore, in the path-based bound calculation, represented in Fig. 2.3.b, the upper bound for a task is determined by the summation of the time execution in the longest path marked, in other words, the most longer way of the program. In the Fig. 2.3.c, the IPET shows the constraints applied over the CFG, supplementing the structure with the representation of arrows of transitions. Moreover, with arithmetic constraints, an upper bound is determined by maximizing the sum of products based on the execution numbers. [45] [19]

## 2.2 Mixed Criticality

Much was discussed in the literature about mixed-criticality; many studies treated mixed-criticality as a binary form. This kind of treatment was known as a dual-criticality system [25][9][11][19]; therefore, other researchers are working with various levels of criticality, maintaining hierarchy priority between the tasks in the system. [3] [7] [9] [44] In addition to assigning levels in mixed-criticality, there are also sub-levels of definitions related to task dynamics. They are divided between two dynamic options or static priorities. Scheduling tasks with static or fixed priorities is commonly used in the literature because of their simplicity compared to the dynamic priorities that change as the task is being performed.[2]

Inevitably not all articles and papers use the same template for job classification. Baruah presents one of this model used. [3] Each task is defined with some criteria, which is the period it occurs ($T_i$), Deadline ($D_i$), computation time ($C_i$), criticality level ($L_i$). These statistics together define one of the models described in the literature. They are typically grouped in an array for the system configurations arranged as follows: $(T_i, D_i, C_i, L_i)$ This attribution model can be used in scheduling techniques such as fixed priority or Earliest Deadline First. (EDF), among others. [9]

There are many types and many ways to classify a system with mixed-criticality. Below we have some classical definitions about these systems and a little explanation about how to identify each one. Some of these systems are found in many papers; actually, it is common to study only one per manuscript. In the latest years, many ways to improve critical systems appeared, with these advances, new problems emerge with them.

### 2.2.1 Single Processor Analysis

In single-core systems, there are many strategies to schedule tasks to guarantee. It is more commonly found in literature in multiple level sections of criticality since it is only one core. The difficulty here is to schedule more Critical Tasks with different weights. Moreover, it introduced a list containing the main studies to solve these problems in single processors systems:

- Job Scheduling
- Fixed Priority Scheduling

  Response-Time Analysis(RTA)-Based Approaches

  Slack Scheduling

  Period Transformation(PT)
- EDF Scheduling
- Shared Resources
- Static and Synchronous Scheduling
- Varying Speed Processors

Analyzing this list, it is possible to notice the many types of fixed priority Scheduling and many techniques of how to schedule the critical tasks or jobs. When the processor calculated something, it is referred to as a job', and every processor task comprises a series of jobs. [1] Initially, this analysis would be introduced, though lately, this scheduling was superseded by other models with more applicability. [9]

#### 2.2.1.1 Job Scheduling

A large number of techniques could be alloyed to schedule jobs. One of these types was Earliest Deadline First (EDF), defined by the nearest deadline would be attending. Still, there is disagreement with this way to prioritize because EDF does not guarantee optimal schedulability. Another type to schedule jobs, according to SOCCI, is theoretically the best among other fixed-job priority scheduling algorithms. This type is the Own-Criticality Based Priority(OCBP) first introduced by Vestal. According to the literature, OCBP is simply a variant of Static Mixed-Criticality (SMC) with less common requirements than OCBP, and this requires a run-time support. [39] [16] [44] [1] The operation of OCBP is determined by the priority order; this means that at each moment in time, the highest-priority available job is executed. The priority is calculated offline, considering the jobs will be executed to the end and recursively determines the lowest priority first. After that, the remaining jobs will be analyzed, and a new lowest priority job will be defined, but this job has more

priority than the first defined. So the remaining Jobs will be determined until all jobs are defined. In other words, the priority of the jobs will be set in growing order until every job has its priority. In this way, the priority of execution is situated with the last job defined; if not mannered to set priorities for all the jobs, the operation is considered not OCBP. [28]

#### 2.2.1.2 Fixed Priority Scheduling

These three scheduling cited, except for the first that can be improved into many levels, are dual-critically systems. There are only two levels of criticality, the high-priority tasks and the low priority tasks. The first technique is the RTA-Based approach; this technique of scheduling tasks is based on run-time behaviors. With the monitoring of task execution time and the prevention of deadline violation, the systems change the operation mode to guarantee the schedulability. This change of operation is triggered by the response of the analysis that indicates if the deadline will be violated or not. The second, Slack Scheduling, according to literature, was first explored by Niz et al. [15] This method consists of using a scheduling scheme to run low criticality tasks in the slack between high critical tasks. Although, issues appeared to this technique, and improvements needed to be executed. This issue was a miss-match of the deadline for a high criticality task if a low criticality task executes beyond its deadline.[22] A lot of improvements were made to make a more robust system. Lastly, an older protocol called Period Transformation (PT) technique, as used before, is for dual-critical systems as well. With the use of PT, it is possible to split a Task with period $T$ and computation $C$ into as many parts as you want. If the task splits in two, the new parameters are $T/2$ and $C/2$. Moreover, the scheduling is facilitated, but this scheme introduces extra overheads because of the increased number of context switches. [9] [6] Although, the PT does not present specific benefit to Mixed Critical Systems (MCS). [44]

#### 2.2.1.3 Earliest Deadline First(EDF) Scheduling

Many techniques are using EDF scheduling, with so many different variations of these techniques, by that fact, it is necessary to introduce the first paper to consider MCS with this type of schedule. The pioneers to use EDF in MCS were Vestal and Baruah. A more complete analysis for the

EDF scheduled system was presented by Ekberg and YI, using two deadlines for each high criticality task, one' real' and other artificial earlier deadline. [5] [9]

This kind of scheduling test prioritizes the execution of the earliest deadline. This prioritization occurs by inserting forward in the execution's queue or even stopping the currently executing task to insert the critical task thereafter. Scheduling techniques using EDF is one of the most widespread techniques, and variations of this technique are widespread in the literature.

#### 2.2.1.4 Shared Resources

This section is to resolve the execution of critical tasks before the deadline based on security protocols to control access for shared resources on the system or employs hardware transactions to solve issues or attempts scheduling approaches to access. These software protocols could block the resources of the system. Because there are so many ways to involve shared resources, it could be vague to include an explanation about that characteristic.

#### 2.2.1.5 Static and Synchronous Scheduling

The alternations between criticality levels can be captured by a static schedule. The Time-Triggered (TT), also called a static scheduler, is divided into tables per mode. With the Single Time Table per Mode (STTM) like Fixed Priority, or Fixed Priority per Mode. The modes can be one per level of criticality. The TT fixes for all jobs their starting time and the time intervals they may execute, and this solution can be used for multiple processors. In other words, it creates one table of priority for each mode of the system. [37] [38]

#### 2.2.1.6 Varying Speed Processors

It is possible to note that most techniques assume a constant speed of the processor but in cases of asynchronous circuitry. With the switching speed, it is possible to decrease or increase the execution time depending on the new clock, like a trade-off involving speed and performance. Moreover, it is possible switching the clock to high values only some high critical task was executing, as well as if it was a low critical task the systems keep the minor clock not to increase power consumption

and maintain the temperature of the core. [4] [9]

### 2.2.2 Multi-Processor Analysis

In multi-core systems, two problems emerged that made impossible the use of the existing real-time Scheduling techniques. These issues are the under-utilization caused by the uncertain WCET analysis and the need to isolate different criticality tasks temporally. By the fact of the guarantee in the CT's execution in the specified deadline, it is common to use pessimistic WCET estimation. Pessimistic WCET is a decrease of the deadline, resulting in an increase of the criticality level of the critical task to guarantee they will finish in time. With that increase of criticality, some processing capabilities are set aside, creating a rate of under-utilization of the computing resources. Furthermore, requirements for the tasks' temporal isolation are necessary because a lower-criticality task can impact the scheduling of a higher-criticality task. If this occurs, the first task should have the same criticality as the last one. Although, it is generally unsuitable because it raises costs, exacerbates the WCET problem, and requires additional solutions in programming constraints. [30] [44] Checking the literature, some approaches appear repeatedly. Here they are:

- Task Allocation
- Schedulability Analysis
- Communication and other Resources

#### 2.2.2.1 Task Allocation

The main approach to work with task allocation contains standard single-threaded tasks. The last issue of task allocation is a guarantee to meet their deadlines in overload scenarios. These scenarios of overload should be treated in scheduling phases. In resume Task Allocation consists of many types of preemptive scheduling approaches with constraints and methods to allocate tasks in a multi-core system to avoid deadline violation. Many scheduling approaches resolve mixed-criticality allocation problems in multi-core systems, such as variation of the EDF scheduling cited previously.

[26] With the help of task placement, it is possible to improve the balance of many partitions of mixed critical systems. In multi-core systems, it is possible to task migrating from one core to another, with task allocation, some tasks were statically allocated to one core exclusively. Without the migration, the other cores reduced some potential delays by this act. This act is also well called by partitioning-based approaches. An important advantage of this approach is the well-established uni-processor scheduling theory that could be applicable to scheduling the subset of tasks allocated to individual cores. [24] [9]

#### 2.2.2.2 Schedulability Analysis

In this section, many techniques are an adaptation of the scheduling techniques used for single- cores improved to work for multi-core systems. Also appears in this section, some studies using global mixed-criticality scheduling. With many adaptations of schedulability research on single-core systems, like the EDF-VD approach and other kinds of EDF, some conclusions appoint that partitioning is more effective instead of global scheduling. [9] [29] Moreover, another approach using two levels of criticality is presented by Kritikakou. With the monitoring of the critical task's execution time in a system with two tasks running simultaneously. After that, the concurrent task is aborted when some violation seems to be possible. [25] There are other schedulability possibilities; in this case, two main options of these variations include pipeline control or Jobs as the focus of the approach. [9]

#### 2.2.2.3 Communication and other Resources

With the advent of more complete platforms such as multiprocessor systems using Network-on-Chip (NoC), some resources should be shared. This entails the need for partitioning so that the low criticality components do not impact higher criticality components so that an Architectural Analysis and Design Language (AADL) was created. That language facilitates the monitoring of the system and the budget enforcement of all computation and communication. [32]

In another way, it is necessary for a bus-based architecture to control the access to the bus so that one core does not negatively affect other cores, making the entire system worse. It is a fact that a task can suffer a substantial increase in its worst-case time execution due to memory access competition. [33] Because of this increase, many approaches aim to improve the bus arbitrator to share

resources. Some of these approaches based on shared communication resources are like a round-robin, Time Division Multiple Access (TDMA), Time-Triggered or protocols of the communication, such as controller Area Network (CAN), that support some architectures.[12] [35] [9]

## 2.3 Leon3

This part of the work intends to show a simple guide about the design of the processor chosen. Created by the European Space Agency (ESA) for space applications, Leon is used for many applications, in general, for those that need high reliability. Due to its large capacity and variety, choosing one for this work was not a problem, since all of the features we need are supplied.

The Leon3 is a synthesizable VHDL, which indicates VHSIC Hardware Description Language, and VHSIC is VLSI (Very-Large-Scale Integration) for ASIC that means Application-Specific Integrated Circuits, with a processor compliant based on SPARC V8 architecture and distributed by Gaisler Aeroflex. The Leon3 template design supports multiprocessing configurations. It is possible to synthesize up to 16 Central Processors Units (CPU) cores and implement synchronous or asynchronous multiprocessing. [43] [27]

The Leon3 multiprocessor has the following features:

- Highly Configurable;
- High Performance;
- Energy Efficient;
- Simple Design Integration;
- Software Support Environment;
- Availability.

In Fig. 2.4, it is possible to visualize a typical configuration with four processors capable of delivering up to 1400 Dhrystone MIPS (Microprocessor without interlocked pipeline stages) of performance. Employing lower frequencies than single processor solutions consequently brings

significant energy efficiency. For more specific information, there is attached documentation about features and designs on the website of Cobham Gaisler.

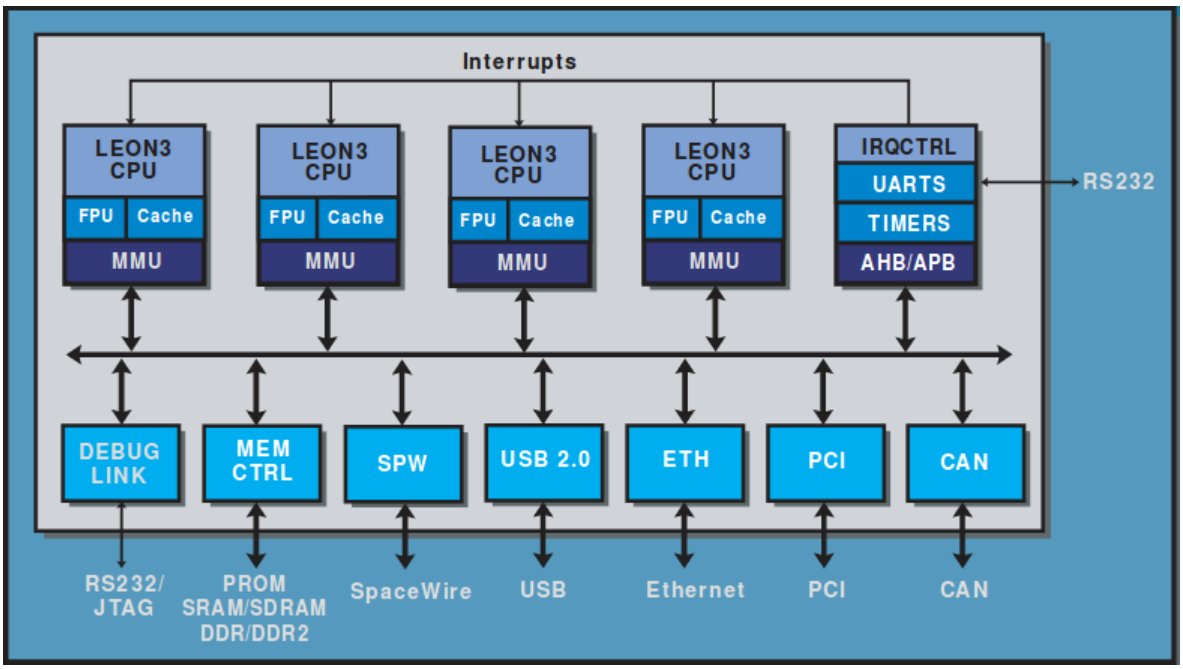


Figure 2.4 – Standard Leon3 Multiprocessor.
Source: Leon, page 1 (2013)

For this work, the minimum requirements are more than only one core processor, a monitored and controllable data bus, one Debug Support Unit (DSU) for one core, and feasible memory cash.

## 2.4 Means of Multiple Access Techniques

To solve many types of conflicts of memory access on multiple processor systems, many ways to divide the access to this shared resource were created. [18] In this section, will be demonstrated some of these techniques. Three ways to multiple access to share resources of systems with multiprocessors is a correlation of time, frequency, and code. They are called Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and lately Code Division Multiple Access (CDMA). Fig. 2.5 shows the correlation between these techniques. In the FDMA concept, transmitting signals occupy non-overlapping radio frequency bands, including the guard band between signals, while code and time of transmission are kept constants. Otherwise, the CDMA uses orthogonal codes to define

which will receive while frequency and time-bandwidth are kept constants. Similarly, TDMA held frequency and code constants, just varying access time. [23] Random access ways, like a Round-Robin, will not be fully presented in this work.

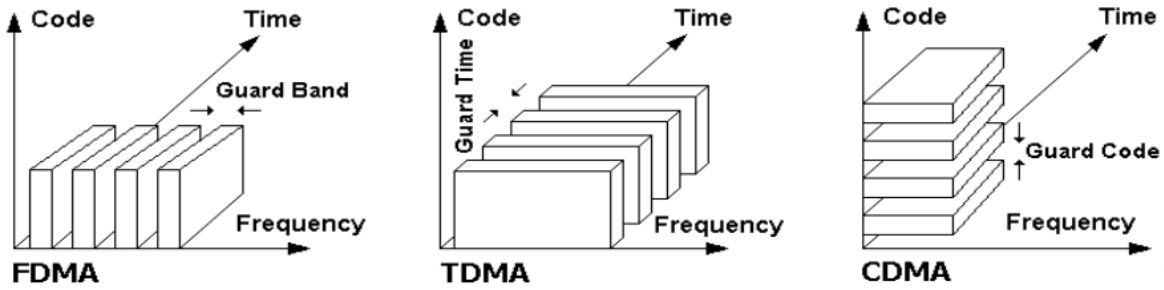


Figure 2.5 – The Difference between Multiple Access Techniques on the Scope.
Source: Ilcev, page 2 (2010)

Moreover, it is interesting to say that these techniques are used in mobile and satellite telecommunication. But, one of the techniques can also be used in the design of multi-core systems.[36] [23] This technique is TDMA, and more details are given below. In the basic concept, it is a time division, where the shared resource is designed as a core for time-space. At the end of this time, the other core can access the resource. In other words, access to the resource is partitioned over the time limit, [36] like the communication of multiple users, where the users, in this case, are the processors included in a shared bus. TDMA divides time into slices and restricts job execution to only one slice at a time.

## 2.5 On-Chip bus communication

There are many types and methods of communication on-chip based buses. For example, most Advanced RISC Machine (ARM) micro-controllers are AMBA (Advanced Micro-controller Bus Architecture). Besides that, AMBA is one of the most widely used on-chip communication standards. The specification about AMBA Advanced High Bus (AHB) is more usually recommended for all new designs. If AHB is compared to Advanced System Bus (ASB), the first presented is preferable to this purpose because of the speed of Bus. Therefore, we will show in this work only details about the AMBA AHB.[31]

The AMBA with AHB, also called AMBA 2.0, is present in many architectures as the main communication between peripherals and similar. So many Intellectual Property (IP) core had a way to connect with AMBA. This type of communication for IP needs an AHB controller with the main function to organize the signals between all the systems. It has a system of fixed-priorities or round-robin. The first consists of happening according to level priority; in other words, if no master requests, the Bus is available to other peripherals. In round-robin mode, the priority is to rotate one by one after each AHB transfer. [43] [31]

Fig. 2.6 shows a typical AMBA AHB with some blocks of IPs connected in the Bus. The bridge to peripherals transforms AHB to Advanced Peripheral Bus (APB). Likewise, the name says it is a bus to supply peripherals.

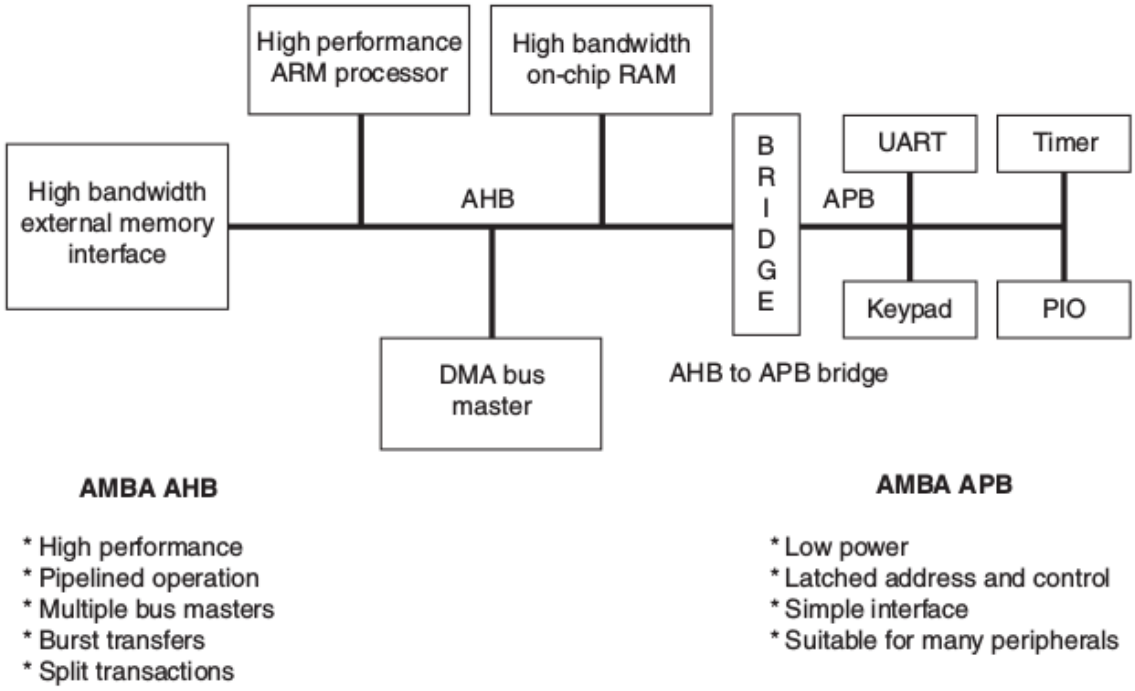


Figure 2.6 – A Typical AMBA 2.0.
Source: Adapted From Pasricha pag. 44 (2008)

On the AHB runs the clock's signal, signals of address ranging from 0 to 31 bits, a signal of control, signs of written data ranging from 0 to 31, a signal of readiness, and at last signals of reading data with a range from 0 to 31. All of these signals are connected in the IPs using multiplexers.

## 2.6 State of the Art

For the present, the approach in this work is necessary to show some basic concepts about the actual state-of-the-art. In this section, some commercial WCET tools and researches prototypes are demonstrated. Moreover, it will introduce the most common model to manage critical tasks and another more recent approach to the topic. Many of these techniques use Control-Flow Analysis (CFA) to gather information about possible paths; therefore, only three commercial tools were chosen. These tools can not be used for all types of processors and platforms, but still, they are used by many industries and companies.

### 2.6.1 WCET Tools

Nowadays, there are many ways to calculate WCET from systems; however, these results are estimations of the exact WCET. In this section will be introduced some tolls that can estimate WCET from specifics systems. Some of these tools are *The RAPITA Tools* of Rapita Systems Ltd, *SWEdish Execution Time tool*(SWEET), and *The aiT Tool of AbsInt Angewandte Informatik*. Moreover, after introducing these tools will be resumed an approach of managing critical tasks called Hard Deadline Enforcer.

#### 2.6.1.1 The RAPITA Tools of Rapita Systems Ltd

This tool aims at medium to large real-time embedded systems on advanced processors. It is an automated performance measurement on-target timing analysis tool. A simple vision of this tool resulted in a perception that this tool reduces the cost and the effort required to conduct timing verification. Apart from that optimization software, it updates legacy systems and can be integrated with critical real-time embedded systems. This tool's main function is to calculate WCET from embedded systems and identify where to focus on optimization. Therefore, its function can be applied to even the most complex targets, including multi-core systems. Another tool provided is Rapita Systems; it is used to schedule tasks in embedded systems. Based on task address, the system provides a high-level overview of the software scheduling, locates rare timing events, and helps to understand the

system capacity and other functions. Furthermore, supporting different targets, including multi-core systems. [10]

#### 2.6.1.2 The aiT Tool of AbsInt Angewandte Informatik

According to information available at ABsInt, aiT WCET Analyzers statically compute tight bounds for the WCET of tasks in real-time systems. The prior objective of this tool is to confront the challenges of computing WCET in real-time systems. Therefore, provide a solution for statically analyzing a task's intrinsic cache and pipeline behavior based on a formal cache and pipeline models. This approach corrects and tights upper bounds to be computed for the WCET, with a technique based on the abstract interpretation. It contains a graphical user interface that supports the visualization of the worst-case program path, thus facilitating the understanding of the platform provided. The benefits of this tool are incredibly tight bounds and these bounds are valid for all inputs and each execution task. Also, it is widely independent of the compiler and source code language used. [42] [17]

#### 2.6.1.3 SWEdish Execution Time tool(SWEET)

Called SWEET tool is a research prototype for flow analysis and BCET/WCET calculation. A research team has developed it in Västerås, Sweden, since 2001. Therefore, the main function of this tool is flowing analysis. Based on analyses of the Artist Flow Analyses (ALF) format for acquiring program C flow analysis. The goal consists of calculating the flow of information as automatically as possible, eliminating the operator's necessity to show results. With powerful loop bound analysis, SWEET can determine the program's estimated WCET even when other tools can not since they know the number of loop iterations. It also includes integrated time estimation on an abstract execution, where all loop iterations are executed separately instead of always merging states. In other words, through calculating program flow, constraints can be derived explicit execution paths, which corresponds to two upper bound times. [20]

### 2.6.2 Approach of Manager Critical Tasks

The most common method to manage critical tasks in avionics is isolated when detected. In this form, the system stops other cores and maintains the core with the critical task executing. In other words, the system entries in a single-core system focused on executing the critical task. [25] Until the critical task ends, the core will continue to perform only the instructions from that task and has the entire system at its disposal. One of the parts shared in multi-core systems is memory. This way, the processor can access memory at any time, preventing access delays from occurring.

#### 2.6.2.1 Hard Deadline Enforcer

The approach proposed by GREEN [19] was a method to guarantee critical task schedulability in multi-core processors. Moreover, with the monitoring of the instructions in the processor pipeline based in the middle of task execution. Together with the analysis of WCET of the task, it would choose specific points, and these points were called reference points (RPs). To chosen checkpoints of a task, these points must not be recurrent; in other words, these instructions are executed only once. These analyses transform WCET static into dynamic because of the critical task end's prediction when the execution state crosses some RP. Through the use of the debug structure, called the Debug Support Unit (DSU), it is possible to collect data in real-time during the system operation. One such set of information is the internal data of the pipeline stages. With this analysis of internal pipeline data, Green states that it is possible to compute the time it takes for each instruction or set of instructions to execute. [19].

Although it is possible to find out on the Bus that instructions are being inserted into the pipeline, this insertion does not necessarily mean it will be executed. Due to the many variations that occur during the computation of each instruction, variations commonly known as "Branch". Due to this execution uncertainty, it is impossible to assert that every instruction carried to the pipeline will be executed, so a more in-depth analysis of the processor's Program Counter will be required. GREEN points out that it is possible to verify that the instruction was executed by monitoring the sixth stage of the Leon3 pipeline. Through this monitoring, if the instruction has reached this specific stage with

the low-level “Annul” signal, it is understood that the instruction has been executed. Keeping this in mind, you can analyze the input of a specific task. [3]

In this chapter, we could see many kinds of solutions for mixed-criticality and some descriptions of the approaches that appear in the literature. Although, after introducing some concepts it will be presented the specification of the proposed approach that contains many of the concepts cited before. Moreover, it is important to note that some of these concepts are mixed with the intent to resolve this work’s objectives.

# 3. THE PROPOSED APPROACH

The main objective of this work is to guarantee the execution of critical tasks before the deadline occurs aiming for better usage of the time available for this execution. To attend this goal, the TDMA policy was deployed into two forms, namely "The Preliminary Technique" and "The Advanced Technique". The difference in these techniques is only the way the task switching process takes place. The overview of techniques resembles but shows different methods of solving the problem cited.

These approaches can be applied to any processor, considering that the designer can collect two signals from the processor. The first signal is the instruction's address in the sixth pipeline's stage($A_{I6PS}$). The latter signal indicates if the current instruction in the pipeline was actually executed or dropped down due to conditional branches or speculation-caused executions in the processor pipeline.

Therefore, the Shared Bus-Access Controller (SBAC) intellectual-property core (I-IP) I-IP operation is described in the following steps:

(a) During CPU operation, the SBAC monitors the starting of the CT execution. This is performed by monitoring the address of the first instruction of this code stored in memory. At this moment, the SBAC is operating as a "sniffer" at the instruction bus, monitoring the moment when the CPU fetches this address from the instruction memory. Before detecting the CT's first instruction address (denoted by CT 1stInstAddr ), the SBAC allows the system bus access policy to be the Shared Mode (i.e., the mode in which all cores have the same time slot duration to access the system bus). For this purpose, the SBAC requires a list containing the first instruction address of the critical task.

(b) Upon detecting the CT 1stInstAddr, the SBAC will proceed with each respective technique. In the first technique, it will initialize an internal counter based on the width between the deadline and WCET to indicate when the Isolated Mode will operate. This time is named $\Delta$T time (the time distance, given in clock cycles, between the CT WCET and the deadline of this code, minus $\Delta$T Completion). The second technique will indicate which shared mode is more suitable for critical tasks and will inform the bus controller. These techniques will be explained in detail along with this chapter. Note: $\Delta$T Completion is the maximum number of clock cycles

(typically, less than 13 clock cycles, or 13cc) required by a given core to properly conclude the instruction that required bus access, currently under execution in the pipeline, when the bus controller triggers the switching process from one task to another, as ruled by the TDMA policy.

(c) During the CT's execution, the SBAC monitors the moment when the last instruction address (CT LastInstAddr) of this task is fetched by the CPU. After the completion of this task, the system bus access policy switches from the current (Isolated) Mode back to the Shared one, in which all cores have the same time slot to access system bus.

(d) The SBAC decrements the counter in the first technique if and only if the instruction fetched by the CPU is executed till the end of the pipeline. Note that not all instructions fetched are executed by the CPU; for instance, in the case of a conditional branch, the CPU will decide if the subsequent instructions will be executed only when such instruction reaches the execution stage of the pipeline. The same reasoning is made during speculative execution, where two pipes are fed with the "then" and the "else" paths of a conditional branch. Then, after the execution stage, the CPU validates one pipe and flushes the other one. To follow up on the above procedure, the SBAC collects, at run time, two signals from the processor ($A_{I6PS}$ and "Annul"). The latter signal indicates if the current instruction in the pipeline was executed or not. Simultaneously, the system sniffs the address of the instructions fetched by the CPU to identify the initial and the end of the CT execution. In the second technique, it is necessary to use these signals because they are very important to detect the CT 1stInstAddr and the CT LastInstAddr. Both are important addresses for the system, and the acquisition of the right moment of the two points is crucial for both approaches.

To perform the above operation steps, the SBAC must be previously configured (during the boot of the system) with the following information:

- TThe WCET (given in clock cycles, cc) of the CT when running in the Isolated Mode;
- The distance (in cc) between the WCET and the Deadline of the CT;
- The first and the last instruction addresses of the memory area where the CT is stored.

Due to these characteristics, it is possible to state that the percentage of access reserved to the Critical Core is effectively delivered. Even the Time Slices do not have symmetry; in the window

of time, the miss-match was supported along the time. It is important to remember that miss-match 76 does not affect the system badly because it represents a negligible percent compared to the whole time in this switching mode.

By the way, it is necessary to choose a platform to develop the proposed system because the system needs to work with multiprocessors. Moreover, it has a type of communication of interconnected IPs and shared memory, with a bus controller and a data flow on that. Leon3 was chosen as the preferred design to develop the proposal due to its wide variety of designs and configuration flexibility. The proposed approach was synthesized for a design Xilinx-510ml, one of the many designs available on the Library of Gaisler. This design is composed of two processors Sparc V8, AHB with AMBA, Leon3 Debug Support Unit, Leon2 memory controller, and AHB status register. More than the IPs cited have synthesized, but for no important reasons, it is only a demonstration of the total configuration of designed Leon3.

For a simple explanation about the system's actions, the approach is designed to alternate the access between the cores and, based on the timing constraints, take some action to avoid the deadline violation. Fig. 3.1 demonstrates the essentials blocks of this system. The block SBAC is designated to monitor the main core and manage the AHB controller based on the critical task's information inserted previously.

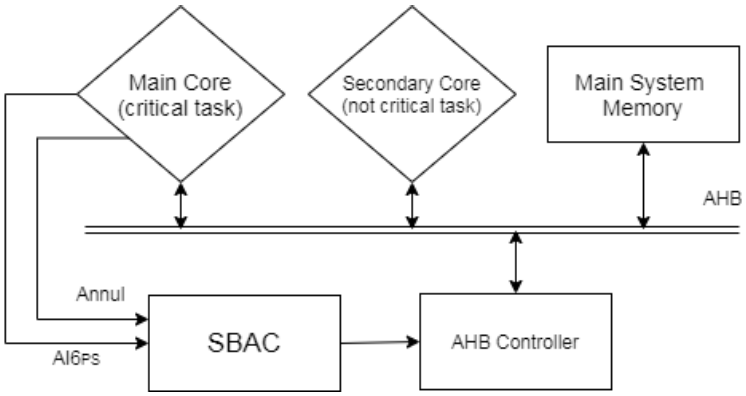


Figure 3.1 – Approach in Block Diagram.
Source: Author (2019)

This approach targets the interconnections between many blocks to decide which action to take. Based on these interconnections, the SBAC set which option to share memory access. The AHB controls the access of the shared bus but does not decide the option; it only executes the order coming

from the previous block.

After introducing the blocks of this approach, we will be introduced each of them and how they work. Firstly, will be seen the operating conditions that indicate the requirement to approach. Moreover, the specifications describe and solute some doubts related to the internal operations of the blocks. Moreover, some examples are presented to elucidate how the approach worked for both techniques presented.

## 3.1 Operating Conditions

In order to facilitate the understanding of the functionality of the entire system, some conditions are defined to guarantee the correct purpose of this model. The conditions cited below are the rules of the environment and should be respected in order to ensure the success of this work:

- Only one Critical Task should be running on the Critical Core;
- The CT is not partitioned and does not take a branch out without ends;
- The other cores (non-critical cores) are not allowed to run critical tasks;
- The execution of each Time Slice will be preserved unless a change of switching mode occurs. For example: if the access was granted for Core 1 and occurs a switching to isolated mode, the access for the core 1 will be withdrawing and will give to core 0.

## 3.2 Specifications

In order to demonstrate distinctly, the main SBAC (top-level) representation block was divided into three other sub-blocks inside. Which is the Bus Monitor, WCET/Deadline Fitter, and TDMA-Based Bus Controller. Each one with its purpose and these blocks can be visualized in Fig. 3.2. Below this section will be explained more details about these blocks.

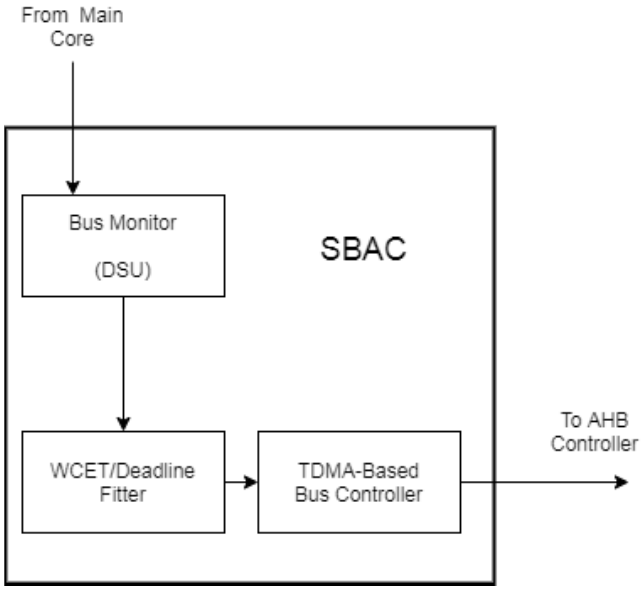


Figure 3.2 – Block SBAC.
Source: Author (2019)

It is important to say that the bus monitor is allocated in DSU that allows creating a side channel of the address access of the memory. Therefore, it is possible to send the address access of the memory to main block without impacting data communication throughput on memory bus.

### 3.2.1 Bus Monitor

The purpose of this block is to provide one of two references needed for TDMA-Based Bus Controller to change the division of access to shared memory. The feature Debug Support Unit is available in more of the designs of Leon3. It is used mainly to resolve bugs and deploy the system. Not only for this purpose, but also can create side channels that provide many signals in the system. To monitor the address bus, a buffer of address can be created without exploiting the system. This copy of the address bus permits to analyze the execution of the tasks, similarly to observe the flow control of the program.

A Bus Monitor has the purpose to monitor the beginning of critical task, based on an address received to trigger the counter. After the address pass on the bus of instructions, it shows the beginning of a critical task. The response for taking the address that transfers on the bus is Debug Support Unit (DSU), where the estimated WCET is also stored. It is necessary to have annotated in the monitor the address of the initial of a critical task. These addresses can be obtained by check-

ing the generated files from Sparc-gaisler-elf-gcc that provide this information. [13] The compiler also provides many other characteristics about program and allows a compilation of program for two cores, a need to this work. With the annotation of strategical points in the critical tasks is possible to confirm how many is missing to finish the execution. Otherwise, based on comparing the configuration tables when the tasks begin, it is possible to set the division of access for the shared memory.

### 3.2.2 WCET/Deadline Fitter

In this block is allocated the estimated WCET and will be considered static, due to easy solving problems with synthesizing. After this, the objective of this WCET/Deadline Fitter Block will supply the system with analyses of the WCET upper bounds. For the estimate of WCET will be analyzed the Control Flow Graphic (CFG) of the critical task to exploit the main address of tasks in the execution. Therefore, these addresses will be triggered by overestimated WCET to the evaluation of the TDMA controller. Beyond information generated by the compiler, it is possible to make the code CFG; this facilitates the conversion to obtain the WCET by means of implicit-path-enumeration (IPET).

With the allocation of this information provided by the compiler is a possibility to use to complete user annotation. Therefore, this information will be put on the WCET analyzer to an estimate the upper bound. It is important to consider for WCET to estimate the state of the divided access because the division of memory access affects directly the execution of the program. The programs that will be chosen for the Critical Core are bubble sort, NMEA (National Marine Electronics Association) Code [8] and Hamming Code for different tests and a cyclical program to the side core. With the intention of preventing non-critical cores from being idle, it is necessary a cyclical program. The same bubble sort code was modified with the intent to become a cyclical program and worked as well, but it re-started when executing was at the end.

### 3.2.3 TDMA-Based Bus Controller

A crucial point of this work is the division of access to shared memory for time slices. For this purpose the AHB control was modified to permit access to window of time for each core in separated time slices, but with the same width. The TDMA bus controller is the block of control of this access based on the signal received from estimated WCET. Therefore, that is a way to guarantee execution of critical tasks before the deadline will be assured. Hence, it could be seen which width time slice provides a better performance. Consisting of two techniques divided by Preliminary Technique and Advanced Technique, both of them use TDMA of one different way and below will be explained their difference in different sections. Both techniques show options between total shutting down and maintaining more than one core. Although, it is possible to say that systems have good malleability. Therefore, this flexibility will be explored to ensure that the system finalizes the critical program along with the operation of the other cores. Generally, a TDMA consists of only one form of running, this form is called Shared Mode in this work, because this option divides in a fair way the access to the bus (i.e., it allocates equal time slots to all cores to access system bus).

## 3.3 Preliminary Technique

Fig. 3.3 depicts the situation where one critical task (CT) and one less critical task are running on two cores (CT is running in Core 0 and the less critical task in Core 1). When both tasks are executed (in the “Shared Mode”), the WCET of the critical task violates deadline D. Thus, the problem is unschedulable (Fig. 3.3.a). However, if CT is executed in the “Isolated Mode”, it is schedulable (Fig. 3.3.b). In contrast with the most common used solution, where only the CT is executed at a time in the multi-core platform till its completion, the first proposed methodology is capable of scheduling the CT by considering the following scenario: initially both tasks are executed on the system. Then, the SBAC I-IP is used to observe on-line the execution of the CT and decides switching the processor from the “Shared Mode” to the “Isolated Mode” (Fig. 3.3.c). This switching mode condition takes place if: the distance (given in clock cycles: cc) from the “point where the CT execution is at that moment” to the beginning of such task is as equal as the distance from the “WCET” to the “Deadline D” of that task.

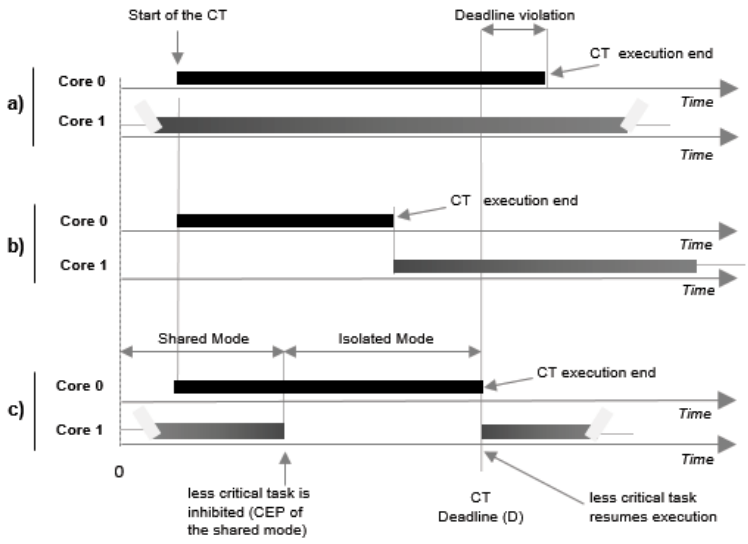


Figure 3.3 – Scheduling Based on WCET When are Considered for Execution (a) Both Tasks are Executed Simultaneously and Consequent Critical Task Deadline Violation; (b) Basic Solution: the Less Critical Task Starts Executing Only After Critical Task Completion; (c) Preliminary Approach.
Source: Author (2019)

A crucial point about this technique is to maintain the Shared Mode until the Critical Execution Point (CEP) is reached by the critical core. After reaching this point, the approach will execute the critical task till the end in Isolated Mode. Moreover, $\Delta$ T represents the distance between the Deadline from the WCET and this distance is the base to find CEP. Executing the CT until arriving CEP is possible to extend the Shared Mode for longer time in comparison to the Conventional Technique that inhibited the other cores when the CT is detected. In Fig. 3.4 it is possible to see the CEP and $\Delta$ T.

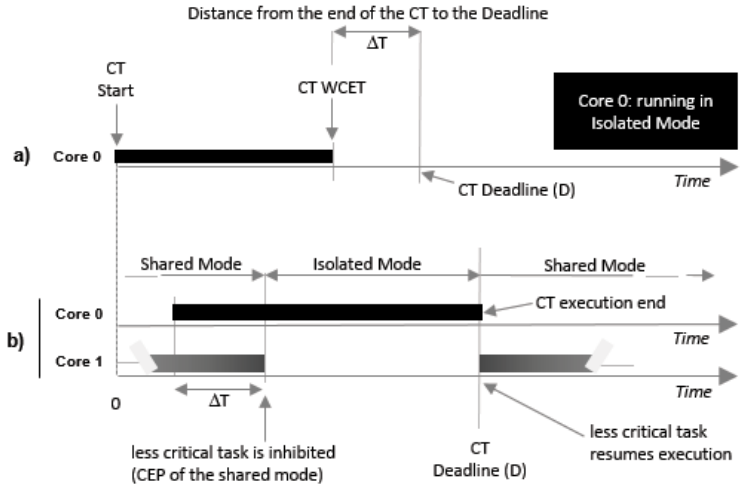


Figure 3.4 – Detailing the Switching Process between the "Shared" and "Isolated" Modes of the Proposed Approach.
Source: Author (2019)

### 3.3.1 Example of Preliminary Technique

Consider the following parameters for a given hypothetical application:

- Length of the TDMA Time Slice (TTS): 10,000 clock cycles(LTTS);
- CT WCET: 3,600,000 cc (which is equal to 360 TTS);
- CT Deadline:4,000,000 cc (400 TTS);
- CT WCET in Shared Mode (one slice per core) is: 14,400,000 cc ($4 * CTWCET$)
- Distance between the CT WCET and the Deadline: 400,000 cc (40 TTS);
- Duration of the CT running in the Critical Core: 3,000.000 cc ($\Delta\, TCT = 300TTS$);
- Duration of the less critical tasks running in the less non-critical cores: periodical tasks, always running;
- $\Delta$T Completion/All is 1 TTS and it corresponds to the summation of $\Delta$T Completion of all cores, during the CT execution in the Shared Model.

In order to elucidate this example the parameters could be seen in Fig. 3.5:

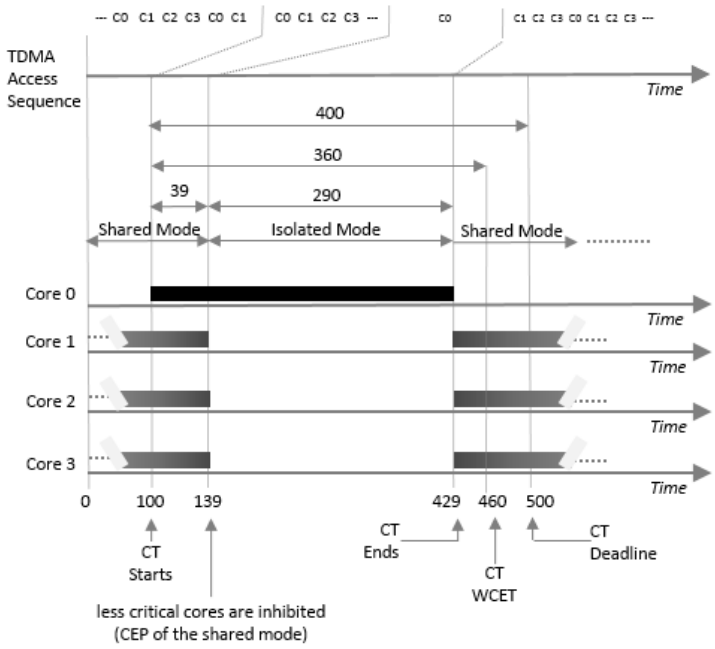


Figure 3.5 – Example of the Proposed Approach Applied to a Quad-core Processor Supporting TDMA-based Bus Access Policy.

Source: Author (2019)

As observed in Fig. 3.5, the following conclusions can be taken:

1. As aforementioned, $\Delta$T Completion/All is 1 TTS. So, the number of TTS to be allocated for the system to run in the Shared Mode is: D – WCET - $\Delta$T Completion/All = 400 – 360 -1 = 39 TTS.

2. In the time interval [100 – 139] TTS, Core 0 used 10 TTS to access the system bus in the Shared Mode (out of a total of 39 TTS shared among the four cores).

3. From TTS = 139, the system runs in the Isolated Mode till the CT completion at TTS = 429.

4. Note that in order for this experiment to run properly, the SBAC has to be configured with the following information:

   - CT WCET: 3,600,000 cc (= 360 TTS);
   - CT Deadline: 4,000,000 cc (= 400 TTS);
   - CT 1st InstAddr;
   - CTLastInstAddr.

5. Note that when CT starts running (at TTS = 100) the SBAC allocates automatically the immediate TDMA time slice to Core 0, no matters what core was running in the preceding time slice. By doing so, the designer can precisely determine how many slices have been allocated to the CT in the Shared Mode. If such number of time slices can be determined, it is easy to compute the remaining number of time slices that is required for Core 0 to complete in the Isolated Mode (in Fig. 3.5, from TTS = 139 to TTS = 429) to complete the CT execution, i.e. 290 TTS.

In summary, with the presentation of examples of the Preliminary Technique, it is possible to elucidate the principal functionality of this part of the approach based on the detection of the critical task and taking action after that. Therefore, after the specification and definition of how the Preliminary Technique works will be presented the Advanced Technique, which maintains for a longer time all the cores than the first technique introduces. Even if some blocks of these techniques are the same, both techniques operate in different ways, the first technique adopts the abort of the other cores on the other hand the advanced technique maintains the operations all the cores.

## 3.4 Advanced Technique

Unlike the Preliminary Technique, the Advanced Technique, in addition to the "Shared" and "Isolated" Modes, presents an additional: the "Native" one. The Native Mode is also known as the "Fair Mode", since it allocates equal time slots for each of the processor core to access system bus. Below, Fig. 3.6 depicts the situation where one critical task (CT) and one less critical task are running on two cores (CT is running in Core 0 and the less critical task in Core 1). When both tasks are executed (in the “Shared Mode”), the WCET of the critical task violates deadline D, likewise technique prior. Thus, the problem is unschedulable (Fig.3.6.a). However, if CT is executed in the “Isolated Mode”, it is schedulable (Fig.3.6.b). Then, the SBAC I-IP is used to observe execution in the critical core and when the CT is detected the system decides to switch from the “Fair Mode” to the other option of “Shared Mode” with different percentage dedicated to Critical core (Fig.3.6.c). The Shared Mode provides longer access time (by proving a larger number of time slots) for the critical core (as compared to the number of time slots allocated to the other cores), thus accelerating CT execution.

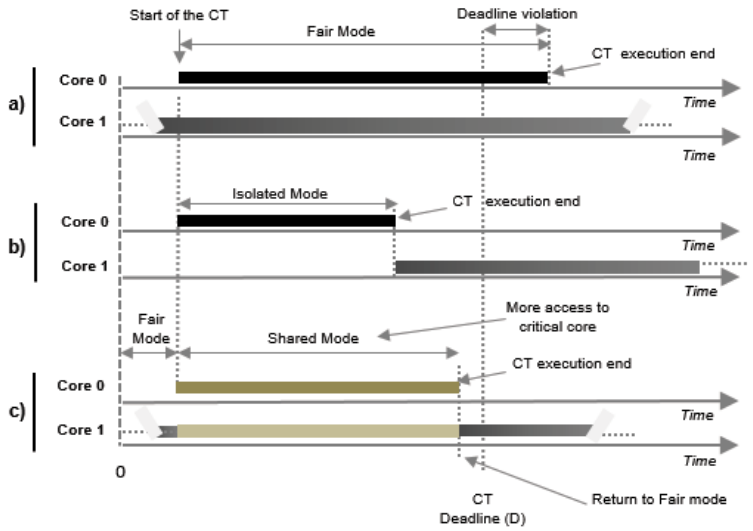


Figure 3.6 – Scheduling Based on WCET when are Considered for Execution (a) Both Tasks are Executed Simultaneously and Consequent Critical Task Deadline Violation; (b) Basic Solution: the Less Critical Task Starts Executing Only After Critical Task Completion; (c) Proposed Approach.
Source: Author (2019)

Firstly, the approach consists of monitoring the Critical Core (CC) and after detecting the critical Task, the SBAC decides which mode the system will switch to. The task switching options consist of three different operating modes : "Isolated", "Shared" and "Native". The Native Mode is also known as the "Fair Mode". The Shared Mode is sub-divided into five time sharing options, which can be used to allocate bus access to the processor cores: 50%, 60%, 70%, 80% and 90%. These options can be seen in Table 3.1.

Table 3.1 – Table of options

| | **MODE** | Dedicated to CC |
|---|---|---|
| 1 | Isolated | 100% |
| 2 | Shared V | 90% |
| 3 | Shared IV | 80% |
| 4 | Shared III | 70% |
| 5 | Shared II | 60% |
| 6 | Shared I | 50% |
| 7 | Fair | same time slot allocated to all cores |

For these options showed, there are a lot of division possibilities depending on the number of cores that exists in the system. Therefore some little variations could appear in the number allocate to the Critical Core in comparison with Table 3.1. These variations appear because with the intention

to maintain the Time Slice some approximations should be taken. By catching the percent division with the minimum time slices possible some Tables were generate, These tables represent the percent division catch up by different quantity of cores for some systems, which the Table 3.2 is for a dual-core system, the Table 3.3 for a quad-core system and the Table 3.4 for octa-core systems. This kind of Table could be generated for N cores, just staggering the division according to Table 3.1. The division between slices to the critical core and total slices of a cycle can also be called by TDMA cycle, naturally, the cycle resets when delivered the TS for the other cores. In other words, every time the TDMA-cycle finishes, the system starts delivering access to CC again in the configured average.

Table 3.2 – Options for a dual-core system

| | **MODE** | Dedicated to CC | Slices to CC/Total of slices |
|---|---|---|---|
| 1 | Isolated | 100% | 1/1 |
| 2 | Shared V | 90% | 9/10 |
| 3 | Shared IV | 80% | 4/5 |
| 4 | Shared III | 75% | 3/4 |
| 5 | Shared II | 66,66% | 2/3 |
| 6 | Shared I | 50% | 1/2 |
| 7 | Fair | 50% | 1/2 |

It is important to note, in this case that the fair mode and Shared Mode I are equal. This occurs only in this type of system because the number of CC and Non-critical Core (NC) are equal. The reduced number of cores entails in reducing the number of Shared Modes. Those peculiarities do not repeat themselves over for systems with other numbers of cores. There are many differences between these systems, because of this it is important to know exactly which are the modes available to switch when the CT is detected. This is showed at Table 3.3, where later is demonstrated an example of functionality in a specific system.

Table 3.3 – Options for a quad-core system

| | **MODE** | Dedicated to CC | Slices to CC/Total of slices |
|---|---|---|---|
| 1 | Isolated | 100% | 1/1 |
| 2 | Shared V | 90% | 27/30 |
| 3 | Shared IV | 80% | 12/15 |
| 4 | Shared III | 70% | 7/10 |
| 5 | Shared II | 62,5% | 5/8 |
| 6 | Shared I | 50% | 3/6 |
| 7 | Fair | 25% | 1/4 |

The Shared Mode II is observed at Table 3.3, where the percentage delivered to the Critical Core is $62,5\%$. This value ($62.5\%$) was selected because it is closer to the ideal timing allocated to the Critical Core ($60.0\%$), compared to the other option ($57.5\%$, i.e., 4/7). This ideal timing allocated is based on Table 3.1 aforementioned.

Table 3.4 – Options for an octa-core system

| | **MODE** | Dedicated to CC | Slices to CC/Total of slices |
|---|---|---|---|
| 1 | Isolated | 100% | 1/1 |
| 2 | Shared V | 90% | 63/70 |
| 3 | Shared IV | 80% | 28/35 |
| 4 | Shared III | 69,56% | 16/23 |
| 5 | Shared II | 61,11% | 11/18 |
| 6 | Shared I | 50% | 7/14 |
| 7 | Fair | 25% | 1/8 |

In the Table 3.4, in the line four and five, appears two truncated percent. Defined by the same method described before, considering the proximity with the reference. It is important to note that the reference proposed for the approach has the objective to allow a solid margin between all the possible methods of division that could be implemented in this TDMA model. With these possibilities of switching forms it is possible to raise the security margin for this approach. Where will be introduced how to choose the operating mode based into certain conditions.

After defining the switching options, it is possible to introduce the calculation to chose which option will be started when the critical task would be detected. By the fact that the maximum time of one CT is approximately on the WCET calculation and do not exceed this time.

On the other hand, the Deadline of this task is also known, which in turn must always be superior to the WCET of the analyzed CT. In addition, a deadline decrease of $10\%$ is obtained to DeadlineSafe and thus it applies being the definitive Deadline of the equation, the reason for this decrease of Deadline will be explained later in the applied Example, aiming to elucidate the explanation. Based on these two system attributes, WCET and DeadLineSafe, the relationship between them is performed as shown in the Equation 3.1 presented below. This equation aims to know the margin between these two attributes of the critical task and therefore allows the choice of a mode of operation that makes the execution of the CT not exceed the deadline.

$$\frac{WCET}{DeadlineSafe} = SecurityMargin \tag{3.1}$$

Based on the result of this equation, select the upper mode of operation that most closely approximates the percentage obtained by the equation. Thus, with the mode of operation selected it is possible to estimate the delay it will have if memory access division is implemented. This increase in its execution time is defined by Equation 3.2, shown below. Where %MODE is defined by the percentage of the module chosen by the system. Therefore, NWCET being the new worst-case execution time, taking into account the access division defined by the memory access controller.

$$\frac{WCET}{\%MODE} = NWCET \tag{3.2}$$

In addition to the delay previously entered by the time division, there is also the delay that occurs with each switch. This delay can be quantified by the number of CT predicted switches along with the number of system cores. It was found that with each permission changes the access controller delays a few clock cycles. This delay occurs because of the need to terminate the statement before transferring access. The equation that analyzes the effect of this delay on how many switches are made is represented in Equation 3.3. Since each switching has an average delay of 7 clocks and is based on the width of SLICE. This average delay was verified through the analysis made by simulation, where it

was verified toggles that demonstrated a delay ranging from at least zero and at most 13 clocks during the CT execution. By multiplying this value by the number of switching you can see if the delay will negatively affect the system that is based on DeadlineSafe, or simply can be disregarded because it is less than $10\%$ previously discounted from the actual deadline.

$$Delay = NWCET \times \frac{7}{TTS} \times (NC - 2) \tag{3.3}$$

It is important to note that the delay is directly related to TTS, which means that the larger the time slice, the smaller the significance of the switching delay in the system. In other words, the switching delay has significance inversely proportional to the slice size. Another factor to note is that if the system is dual-core, the delay exists for both the critical core and the secondary core. Due to this duplicity, they would cancel each other out because, technically, the same delay exists for both, thus providing an equal execution. Which causes the core to also hold access longer, thereby decreasing the delay suffered by the other core. Because of these circumstances there is the last factor in Equation 3.3, which means that the more cores in the system, the more delays will be added, if the total number of cores is 2, the delay becomes negligible.

### 3.4.1 Examples of Advanced Technique

In this session, two examples of how the proposed approach works is presented. The first example describes a small timing margin between WCET and Deadline, whereas the second one depicts a large margin between these two parameters. These two scenarios are necessary to describe how the proposed technique switches the system operation between the Shared and Isolated Modes, as a function of the distance between the WCET and the Deadline of the task.

Example 1: "Quad-core Small Timing Margin"

Consider the following parameters for a given hypothetical application:

- Length of the TDMA Time Slice (TTS): 10,000 clock cycles(LTTS);
- CT WCET: 3,600,000 cc (which is equal to 360 TTS);
- CT Deadline:5,000,000 cc (500 TTS);

- Duration of the CT running in the Critical Core: 3,000.000 cc ($\Delta TCT$ =300 TTS);
- Duration of the less critical tasks running in the non-critical cores: periodical tasks, always running;
- $\Delta$T Completion/All is 1 TTS and it corresponds to the summation of $\Delta$T Completion of all cores, during the CT execution in the Shared Mode.

First, DeadlineSafe is calculated, which as explained earlier is $10\%$ lower than the original Deadline. This calculation is presented in Equation 3.4.

$$DeadlineSafe = Deadline \times 0.9 = 500 \times 0.9 = 450 \tag{3.4}$$

Now apply the value obtained for calculating the safety margin between WCET and DeadlineSafe, as shown in Equation 3.5.

$$SecurityMargin = \frac{WCET}{DeadlineSafe} = \frac{360}{450} = 0.8 = 80\% \tag{3.5}$$

Depending on the percentage obtained from the ratio in the previous calculation, the mode that represents a greater or equal percentage that comes closest is chosen. In these circumstances the mode chosen will be Shared Mode IV, which is also $80\%$. Because the percentage is equal, it is not necessary to calculate the new WCET, because the result obtained would be the same, since there was no divergence between the calculated percentage and the chosen percentage. In the next example, the importance of NWCET calculation will be clearer. Knowing that the value of NWCET is equal to DeadlineSafe it is possible to calculate the delay generated by the switches, the delay is calculated in Equation 3.6, shown below.

$$Delay = NWCET \times \frac{7}{LTTS} \times (NC - 2) = 0.63TTS \approx 1TTS \tag{3.6}$$

Even with 1 TTS delay the system does not go beyond the deadline because it does not pose a threat to the system. Based on the $10\%$ margin placed on the actual CT Deadline. You can reduce this margin from $10\%$ to $5\%$ or even less. In contrast, because the switching delay is directly proportional to the number of cores, if this decrease was to be made for a system with more cores

and a smaller TTS, it should be increased again. Later this analysis will be studied in more details. Thus, we note the importance of decreasing Deadline for DeadlineSafe, given that if the CT ends in its WCET the delay would cause the CT execution to be delayed by some TTS, so the margin prevents that if the delay is significant, it cannot properly exceed the security rate set by the actual Deadline $10\%$ reduction in this case. Since the critical task execution time has been defined, see assignments in this example, the next step is to calculate the moment when the critical task is completed to get a sense of how many TTS were used by the Critical Core. In addition, the $N\Delta TCT$ is calculated to obtain the point where it is terminated. The calculus is demonstrated below in the Equation 3.7.

$$N\Delta TCT = \frac{\Delta\, TCT}{\%MODE} = \frac{300}{0.8} = 375 \tag{3.7}$$

In Fig. 3.7 you can see how the system affected the CT in this example.

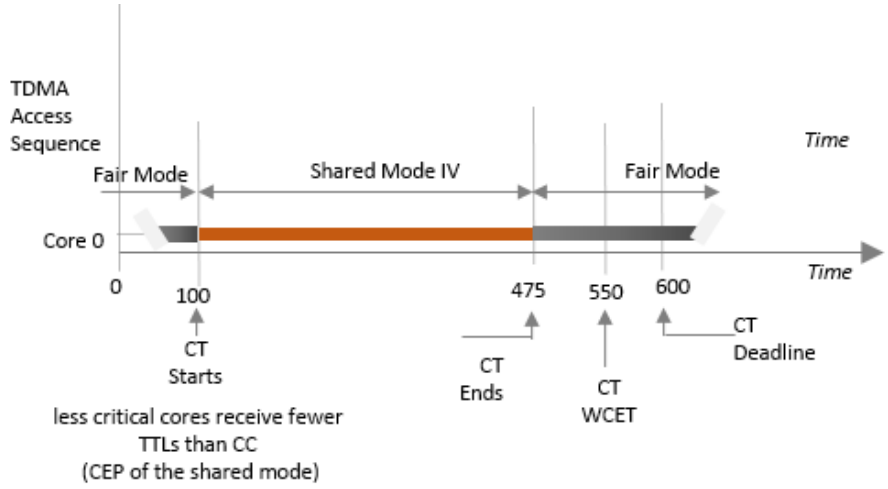


Figure 3.7 – Example of Low Margin.
Source: Author (2019)

Example 2: "Quad-core Large Timing Margin"

Consider the following parameters for a given hypothetical application:

- Length of the TDMA Time Slice (TTS): 10,000 clock cycles (LTTS);
- CT WCET: 3,600,000 cc (which is equal to 360 TTS);
- CT Deadline:10,000,000 cc (1000 TTS);
- Duration of the CT running in the Critical Core: 3,000.000 cc ($\Delta TCT$ =300 TTS);

- Duration of the less critical tasks running in the non-critical cores: periodical tasks, always running;
- $\Delta$T Completion/All is 1 TTS and it corresponds to the summation of $\Delta$T Completion of all cores, during the CT execution in the Shared Mode.

Repeating the steps of the previous example this time with a large margin of distance between WCET and Deadline.

$$DeadlineSafe = Deadline \times 0.9 = 1000 \times 0.9 = 900 \tag{3.8}$$

After DeadlineSafe is calculated, the procedure is performed to find out which sharing mode the critical task will act on.

$$SecurityMargin = \frac{WCET}{DeadlineSafe} = \frac{360}{900} = 0.4 = 40\% \tag{3.9}$$

Note that the result of the keying percentage was $40\%$, this split option is unavailable, so a change must be done for a higher option than that. That in this case would be Shared Mode I, which has a split percentage where $50\%$ of access is destined to the main core. Knowing the chosen option it is possible to verify the new WCET in this mode, the calculation of the new WCET in this switch option is shown below in Equation 3.10.

$$NWCET = \frac{360}{\%MODE} = \frac{360}{0.5} = 720 \tag{3.10}$$

Knowing the number of time slices that the CT will be executed it is possible to calculate the delay generated by the memory access switch, as previously mentioned. In this example the delay due to switching is longer than in the previous one, this is due to more access transactions. This delay is shown in Equation 3.11.

$$Delay = 720 \times \frac{7}{TTS} \times (NC - 2) = 1.008TTS \approx 2TTS \tag{3.11}$$

Unlike the previous example, due to the fact that it has more switching by the large margin of difference, there is a considerable increase in delay. However, as the margin is high, the delay

due to switching is not substantial and does not harm the system. Note that rounding will always be performed for the next full TTS due to the preference to not to split the time slice. This is due to the fact that TDMA has a defined access time and no matter how much time is left to finish execution. It will only occur on the next time slice, no matter how much is left, it will always be performed on the next time slice.

Knowing the way the system will work and the task execution time, you can calculate when it will end. The Equation 3.12 demonstrates this.

$$N\Delta\, TCT = \frac{\Delta\, TCT}{\%MODE} = \frac{300}{0.5} = 600 \tag{3.12}$$

With all points acquired it is possible to generate a figure that represents the points of the CT. In the Fig. 3.8 you can see how the system affected the CT in this example.

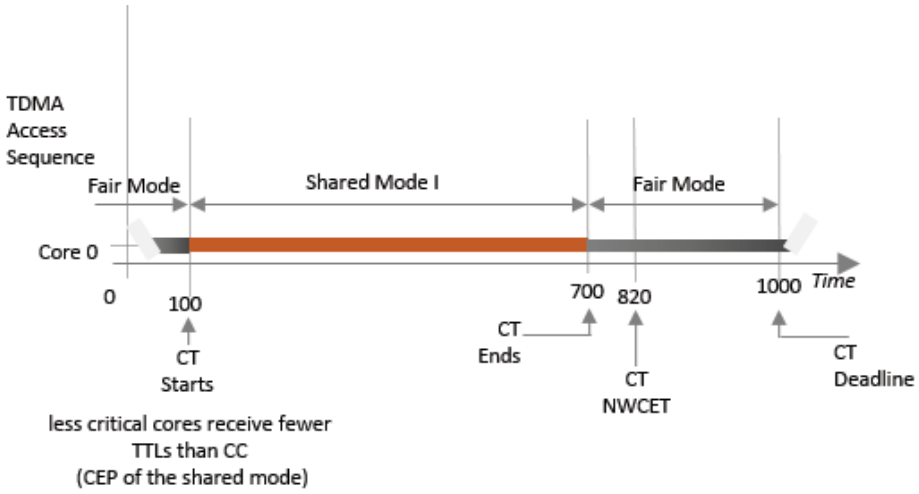


Figure 3.8 – Example of Larger Margin.
Source: Author (2019)

Note that the critical task has just been executed before the deadline, with a large margin because the execution time is shorter than WCET. Even if the runtime was WCET, it would respect the deadline by a considerable margin. Due to the fact that the deadline used in the calculations is a DeadlineSafe. It is also noted that in this example because the system has a large TTS the delay is relatively insignificant and may even be disregarded.

In this section we had showed many concepts about this approach, the main block was divided into sub-blocks and explain each one. One of these blocks changes significantly depending

on the switching techniques. The changes mainly affect the AHB-controller by the fact of inserting other options beyond the isolated mode that are present in both techniques. After the description of both techniques, it is possible to note that the Advanced Technique receives majority changes when the number of cores in the system changes. This Change does not occur in the Preliminary Technique because the parameter as referenced is the distance between deadline and WCET, and this width is not attached from the number of cores in the system. After presenting examples of both switching techniques to clarify the functioning of the approach, will be presented the implementation of these techniques. Many simulations were made to prove examples cited before and moreover some comparisons between techniques will be discussed.

## 3.5 Implementation

The implementation was made in Modelsim, using one of the Leon models provided by Cobham Gaisler. To make the necessary changes for the implementation of the proposal were created VHDL codes that integrated the architecture to later validate the proposal. After a study of the architecture, it was necessary to make some internal connections to implement the bus controller. The signals that were picked up from the Critical Core were the sixth stage pipeline instruction and the signal "annul" of this stage. In this way you can see if the instruction was executed or if there was no deviation taken. In addition to capture these signals, it is also necessary to carry a signal from the Debug Support Unit to AHB-Control, this signal represents the state that the access division must be in. Also was inserted a TDMA controller for the access bus because of this policy of division not available for the Leon. The added links can be seen in Fig. 3.9 where we notice the different levels that the signals are in some groups and subgroups that make up the model used.

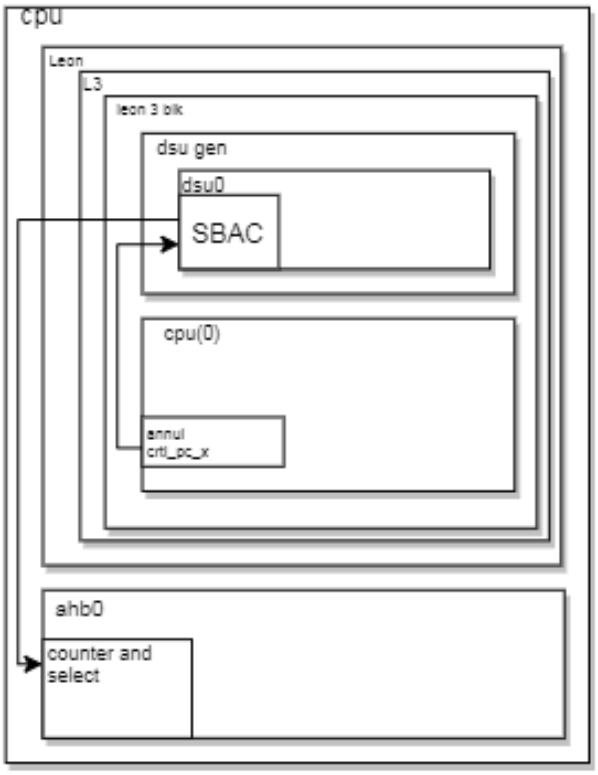


Figure 3.9 – Signals and Connections of Approach.
Source: Author (2019)

Among these stages, the codes described in VHDL only differed in the forms of bus access control within the DSU. Knowing that the TDMA-controller only obeys the DSU, and the signals that are acquired are necessary in both approaches created. In this way then they will be shown once in this document only.

Fig. 3.10 depicts the code added in order to copy the instruction address that is currently in the sixth stage of the pipeline (Exception Stage) and its Annul signal. It is also possible to observe in the signals being transferred to the output to be interconnected to the DSU input. In Fig. 3.11 it is possible to see the signal_entrada_DSU signal that enters AHB-controller as input. This signal is used to change the allocation of time slices between cores. As stated earlier, Leon3 consists of 7 pipeline stages, which are: Instruction Fetch (FE), Decode (DC), Register Access (RA), Execute (EX), Memory (ME), XC(Exception ) and Write( WR). The code placed at end of the Exception Stage is described in Fig. 3.10. Note the addition of the code in line 4272 on file $iu3.vhd$, of the segment intern Leon3v3, from the Gaisler library (see in Fig. 3.10). The Program Counter (PC) is found in this file.

```
4271
4272      if r.x.ctrl.annul = '0'then
4273          annul_x <= '0';
4274      else
4275          annul_x <= '1';
4276      end if;
4277      crtl_pc_x.crtl_pc_x  <= r.x.ctrl.pc;
4278
```

Figure 3.10 – Copy of the "Instruction Address" and "Annul bit" at the End of the Exception Stage.
Source: Author (2019)

Upon detection of the instructions that has been executed it is possible to work with the DSU to operate the switching logic of the TDMA controller placed on the AHB-control. This is shown below, in Fig. 3.11 the key code for the preliminary approach is set after detection of the CT start instruction in Variable $START\_CT$, which activates the Timer to keep Shared Mode $\Delta T$ previously calculated by the difference between the calculated WCET and the deadline. Finally upon detection of the ending statement defined in the $END\_CT$ variable, the code returns to default memory access operation. In Fig. 3.11, $\Delta T$ is represented by Td in variables.

```
WDT_reg : process (cpuclk, crtl_x_pc, annul_x)
                variable count_clk : integer range 0 to 30001:=0;

                variable Td : integer range 0 to 100:=39;
                variable START_CT:  std_logic_vector(31 downto 0):= x"4000031c";  --START BSORT
                variable END_CT:  std_logic_vector(31 downto 0):=  x"40000430";   --END BSORT

        begin
                if rising_edge(cpuclk) then

                        if RESET_ALL and (rst = '0') then
                                user_signal <= '0';
                                sinal_entrada_dsu <= "001";    --SHARED MODE

                        elsif crtl_x_pc.crtl_pc_x = START_CT and annul_x /= '1' then
                                user_signal <= '1';

                        elsif crtl_x_pc.crtl_pc_x = END_CT and annul_x /= '1' then
                                user_signal <= '0';
                                sinal_entrada_dsu <= "001";  --SHARED MODE
                        end if;

                        if user_signal(0) = '1' then
                                count_clk:=count_clk+1;

                                if (count_clk >=Td*300) then    -- TIME TO OPERATE AFTER OF THE DETECT
                                        sinal_entrada_dsu <= "000"; -- ISOLATED MODE
                                        count_clk:=0;
                                end if;
                        end if;
                end if;
end process;
```

Figure 3.11 – Detection of the CT Start Instruction and Timer Setup to Trigger Isolated Mode.
Source: Author (2019)

In Fig. 3.12 the advanced approach VHDL code is shown, where the CT starting instruction in Variable $START\_CT$ is defined and the end statement in $END\_CT$ is defined. Upon detection of the CT starting instruction it is indicated that $AHB\_control$ operates in the best way to ensure the execution of the critical task and to ensure parallelism.

```
                -- TABLE OF OPTIONS OF THE AHB_CONTROL
                -- ISOLATED MODE "000"
                -- Shared/Fair MODE "001"
                -- SHARED MODE 1    "010"
                -- SHARED MODE 2    "011"
                -- SHARED MODE 3    "100"
                -- SHARED MODE 4    "101"
                -- SHARED MODE 5    "110"
WDT_reg : process (cpuclk, crtl_x_pc, annul_x)
                variable START_CT:  std_logic_vector(31 downto 0):= x"4000031c";
                variable END_CT:  std_logic_vector(31 downto 0):=  x"40000430";
        begin
                if rising_edge(cpuclk) then

                        if RESET_ALL and (rst = '0') then
                                user_signal <= '0';
                                sinal_entrada_dsu <= "000";  -- In reset, others cores still don't started
                        elsif crtl_x_pc.crtl_pc_x = START_CT and annul_x /= '1' then
                                        user_signal <= '1';   --Signal to indicate the detection
                                 sinal_entrada_dsu <= "101"; -- Shared Mode 4

                        elsif crtl_x_pc.crtl_pc_x = END_CT and annul_x /= '1' then
                                        user_signal <= '0';   --Signal to indicate the detection
                                 sinal_entrada_dsu <= "001"; --Fair mode
                        end if;
                end if;
end process;
```

Figure 3.12 – Detection of the CT Start Instruction and Change to Respective Shared Mode.
Source: Author (2019)

If there are more than one CTs running at different times, you can add to the Advanced Approach VHDL and define an operating mode for each one, in Fig. 3.13 the code in VHDL is shown, which shows the instructions of each of the so-called CT programs and the modes chosen respectively for each, based on the calculations shown in the Advanced Approach equations. In the following chapter we will better discuss the programs presented and considered critical tasks, as well as the WCET calculation performed on them and different analysis of the simulations using the VHDL codes presented here.

```
-------------------------------------------------------------------------
                USER PROCESS
-------------------------------------------------------------------------
                -- TABLE OF OPTIONS OF THE AHB_CONTROL
                -- ISOLATED MODE "000"
                -- FAIR MODE     "001"
                -- SHARED MODE 1 "010"
                -- SHARED MODE 2 "011"
                -- SHARED MODE 3 "100"
                -- SHARED MODE 4 "101"
                -- SHARED MODE 5 "110"
 WDT_reg : process (cpuclk, crtl_x_pc, annul_x)

                variable START_CT1:  std_logic_vector(31 downto 0):= x"400006b8"; -- START BSORT
                variable START_CT2:  std_logic_vector(31 downto 0):= x"40000764"; -- START NMEA
                variable START_CT3:  std_logic_vector(31 downto 0):= x"400007f4"; -- START HAMMING

                variable END_CT1:  std_logic_vector(31 downto 0):=  x"40000730"; -- END BSORT
                variable END_CT2:  std_logic_vector(31 downto 0):=  x"400007c0"; -- END NMEA
                variable END_CT3:  std_logic_vector(31 downto 0):=  x"40000880"; -- END HAMMING

        begin
                if rising_edge(cpuclk) then

                        if RESET_ALL and (rst = '0') then
                                 user_signal <= '0';
                                 sinal_entrada_dsu <= "000";  -- NATIVE MODE

                        elsif crtl_x_pc.crtl_pc_x = START_CT1 and annul_x /= '1' then   --START BSORT
                                user_signal <= '1';
                                sinal_entrada_dsu <= "101"; -- SHARED MODE 5

                        elsif crtl_x_pc.crtl_pc_x = START_CT2 and annul_x /= '1' then   -- START NMEA
                                user_signal <= '1';
                                sinal_entrada_dsu <= "110"; -- SHARED MODE 6

                        elsif crtl_x_pc.crtl_pc_x = START_CT3 and annul_x /= '1' then   -- START HAMMING
                                user_signal <= '1';
                                sinal_entrada_dsu <= "010"; -- SHARED MODE 2

                        elsif crtl_x_pc.crtl_pc_x = END_CT1 and annul_x /= '1' then  -- END BSORT
                                user_signal <= '0';
                                sinal_entrada_dsu <= "001"; -- NATIVE MODE

                        elsif crtl_x_pc.crtl_pc_x = END_CT2 and annul_x /= '1' then  -- END NMEA
                                user_signal <= '0';
                                sinal_entrada_dsu <= "001"; -- NATIVE MODE

                        elsif crtl_x_pc.crtl_pc_x = END_CT3 and annul_x /= '1' then  -- END HAMMING
                                        user_signal <= '0';
                                  sinal_entrada_dsu <= "001"; -- NATIVE MODE
                        end if;
                end if;
 end process;
```

Figure 3.13 – Detection of the Start Instruction of Several CT's and the Switch to the Shared Mode to Each One.

Source: Author (2019)

After displaying the capture code of the executed instruction, the code of choosing mode in which the bus controller will operate. is important to display the code in VHDL used for the division of memory access. In Fig. 3.14 you can see the counter displayed that increments with each clock cycle accounting for the size of $TimeSlices$, which has their width defined by the $TTS$ variable. TDMA controller code that was used for dual-core systems.

The signal $REQ1\_cactch$ detected when the second core started and this possibility signalizes the system starts the division for them, before this point the system operates in isolated mode because the controller does not have reasons to grant the access for one core when he is in turn off. The signal_count is used in the function to select the next core to access the memory, when this signal was set with low state logic the Critical Core will receive the grant to access the memory. If it stated

logic high the other core will receive the grant to access the memory. In quad-core systems a similar type of TDMA control, however, still maintains aspects based on tdma_controller.

```
COUNT_proc : process (clk, REQ1_cacth , sinal_entrada_dsu)
variable TTS : integer range 0 to 500 := 300;
variable count : integer range 0 to 1001:= 0;
variable    i : integer range 0 to 1001 :=0;

begin

        if rising_edge(clk) then

                if sinal_entrada_dsu = "000" then -- ISOLATED MODE
                        i:=0;

                elsif sinal_entrada_dsu = "001" then -- SHARED MODE
                        i:=TTS;

                end if;

                if  RESET_ALL and rst = '0' then
                        signal_count <= '0';
                        count :=0;
                elsif REQ1_cacth = '1'  then

                        if sinal_entrada_dsu = "000" then
                        signal_count <=  '1';
                        count := 0;
                        elsif count < i  then
                                signal_count <=  '0';
                        elsif count < i+TTS and count >= i then
                                signal_count <= '1';
                        elsif count >= i+TTS then
                        count := 0;
                        end if;
                        count := count+1;
                end if;
        end if;
end process;
```

Figure 3.14 – TDMA Controller for the Preliminary Technique Applied to the Dual-core Version of the Leon3 Processor.

Source: Author (2019)

In the Fig. 3.15 for advanced approach will be demonstrated tdma_controller for quad-core system. This tdma_controller is used to divide the access for four cores based in Table 3.3. The order was the table organized to respond the ordering of the control, but was allocated in different order.

```
COUNT_proc : process (clk, REQ1_cacth , sinal_entrada_dsu)
variable TTS : integer range 0 to 900 := 300;   -- Set TIME-SLICE
variable count : integer range 0 to 9200:= 0;
variable    i : integer range 0 to 9200 :=0;

begin
        if rising_edge(clk) then
                case sinal_entrada_dsu is

                        when "000" => i:=0;     -- ISOLATED MODE
                        when "001" => i:=TTS;   -- SHARED FAIR MODE
                        when "010" => i:=3*TTS; -- SHARED MODE 1
                        when "011" => i:=5*TTS; -- SHARED MODE 2
                        when "100" => i:=7*TTS; -- SHARED MODE 3
                        when "101" => i:=12*TTS;-- SHARED MODE 4
                        when "110" => i:=27*TTS;-- SHARED MODE 5
                        when others => i:=0;
                end case;

                if  RESET_ALL and rst = '0' then
                        signal_count <= "00";
                        count :=0;
                elsif REQ1_cacth = '1'  then
                        if sinal_entrada_dsu = "000" then
                        signal_count <=  "00";
                        count := 0;
                        elsif count < i  then
                                signal_count <= "00";
                        elsif count < i+TTS and count >= i then
                                signal_count <= "01";
                        elsif count < i+2*TTS and count >= i+TTS then
                                signal_count <= "10";
                        elsif count < i+3*TTS and count >= i+2*TTS then
                                signal_count <= "11";
                        elsif count >= i+3*TTS then
                        count := 0;
                        end if;
                        count := count+1;
                end if;
        end if;
end process;
```

Figure 3.15 – TDMA Controller for the Advanced Technique for Models of Leon with Quad-core.
Source: Author (2019)

In the next section, will be focused on the validation of these approach's and demonstrate how these blocks affects the distribution of the access of the memory. Although will be showed the simulations based on these approaches and a comparison about them. Moreover is presented some discussions about the width of the Time Slice and how it affects the system.

# 4. VALIDATION OF THE PROPOSED APPROACH

Practical experiments have been developed in order to validate the proposed approach. With this purpose, a dual-core version of the Leon3 soft-core processor was implemented in VHDL language. Leon3 is the processor chosen as the target system of this work due to the significant acceptance in the scope of critical applications, in particular, aerospace. It is a synthesizable model of a 32-bit processor compatible with the SPARC V8 architecture, made available by the company Aeroflex Gaisler, under the GNU GPL license. The source code is free to use for research and educational purposes and is distributed as part of the GRLIB IP library. The whole system (processor, WDC, shared memory and applications) was running in the ModelSim Simulator, from Synopsys. Fig. 4.1 depicts a general overview of the implemented system.

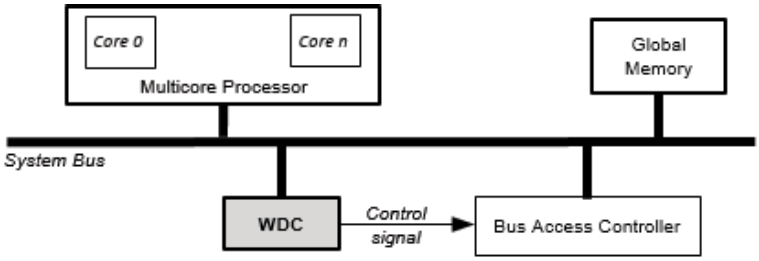


Figure 4.1 – Overview of the System Architecture Described and Simulated in VHDL to Validate the Proposed Approach.

Source: Author (2019)

The validation procedure is described as follows:

(a) First, the whole system was simulated with the CT running in Core 0 while a less critical task was running in Core 1, both struggling for bus access under the TDMA bus access policy, without applying the proposed approach, i.e., without using the SBAC I-IP.

(b) Second, the same simulation was performed, but the SBAC I-IP was connected to the system in order to guarantee the mixed-criticality execution of tasks running in Core 0 and Core 1.

(c) For both steps (a) and (b) above, three critical tasks were developed to run on Core 0, one at a time: a Hamming Coder, a Bubble-Sort and a NMEA Coder. The format chosen for NMEA is Global Positioning System Fix Data (GPGGA) [8]. During the whole validation process, a Bubble-Sort program was running in Core 1, as the less critical task.

Fig. 4.2 depicts the Control Flow Graph (CFG) of the three CTs. These tasks were selected having in mind the different complexities required to compute the WCET, from the simplest CFG (Hamming Coder) to the most complex (Bubble Sort). Hamming Coder was the easiest program to compute the WCET because it always takes the same time to run, no matter what the inputs are. On the other hand, the Bubble Sort program is the most complex to compute the WCET, as can be observed in the CFG of Fig 4.2.c: several loops and a larger number of paths between the input and output of the CFG when compared to the NMEA coder.

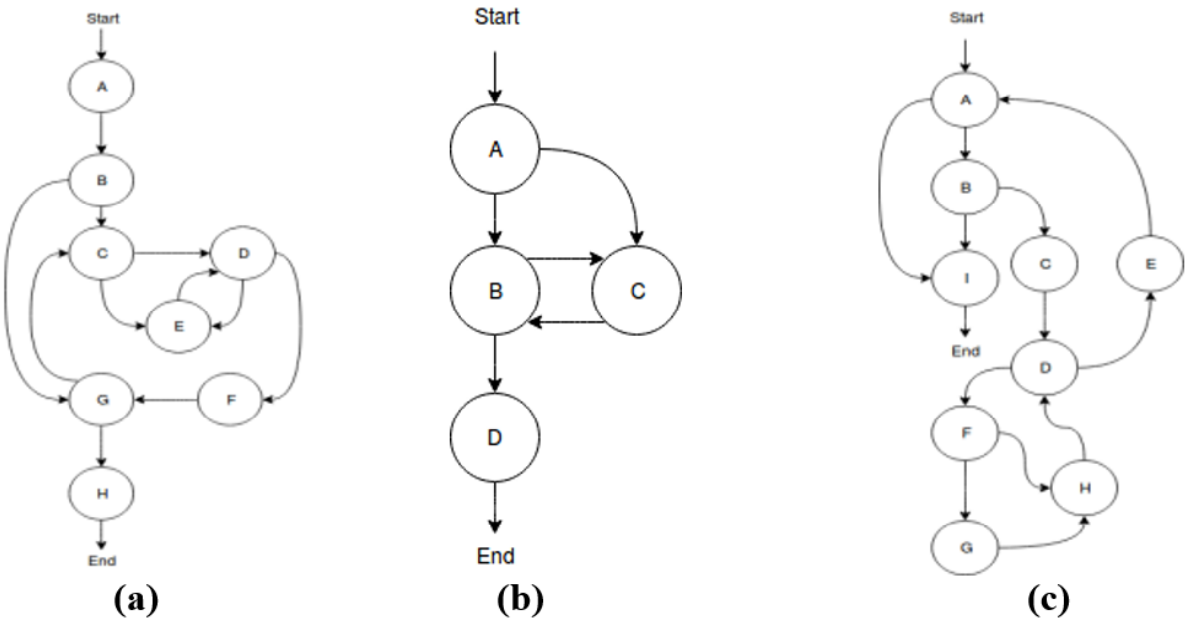


Figure 4.2 – Control Flow Graph (CFG) of the Three CTs to Run on Core0: (a) Hamming Coder; (b) NMEA Coder; (c) Bubble Sort.

Source: Author (2019)

Tables 4.1, 4.2 and 4.3 summarize the following information about the three CTs aforementioned (values are given in terms of nanoseconds (ns) and clock cycles (cc) for Core 0):

- WCET computed for each of the CTs;
- Execution Time computed for each of the CTs according to a given randomly selected input;
- Deadline defined for each task (for this experiment, Deadline was arbitrarily defined to be "approximately 1.35 x WCET").

Therefore, for the Hamming Code program the Deadline was arbitrarily set to 250,000 ns (or 2,000 cc) after the beginning of the program execution. For the experiment, the length of the TDMA Time Slice (TTS) was also defined as 300 cc.

Table 4.1 – WCET computed for the three cts

| **Critical Task(CT)** | WCET Computed | |
|---|---|---|
| | Time(ns) | Clock Cycle(cc) |
| Hamming Coder | 184,612.5 | 14,769 |
| NMEA Coder | 212,050 | 16,964 |
| Bubble Sort | 447,025 | 35,762 |

Table 4.2 – Execution time for the three ct's according to a selected input vector

| **Critical Task(CT)** | Execution Time | |
|---|---|---|
| | Time(ns) | Clock Cycle(cc) |
| Hamming Coder | 162,387.5 | 12,991 |
| NMEA Coder | 153,662.5 | 12,293 |
| Bubble Sort | 353,175 | 28,254 |

Table 4.3 – Deadline arbitrarily defined for the three cts

| **Critical Task(CT)** | Deadline Defined | |
|---|---|---|
| | Time(ns) | Clock Cycle(cc) |
| Hamming Coder | 250,000 | 20,000 |
| NMEA Coder | 300,000 | 24,000 |
| Bubble Sort | 600,000 | 48,000 |

In this technique, the WCET is computed as the summation of the number of clock cycles required to execute the instructions present in all basic blocks implementing the CFG of the task, including their feedbacks and loops. With this purpose, the code was running in the stand-alone mode in the LEON3 processor and it was applied the IPET technique as formulated by [19]. In addition,

after calculating the WCET of each task, all tasks were simulated individually in both Preliminary and Advanced Techniques. The next section presents the results of such experiments for the proposed (Preliminary and Advanced) techniques. Then, the results are discussed and the differences yielded by using both techniques underlined.

## 4.1 Preliminary Experiment

After the implementation of the Preliminary Technique, it is necessary to calculate the $\Delta$ T minus $\Delta$ T Completion. This result indicates the time interval the system remains in the Shared Mode before switching to the Isolated Mode. In a dual-core system, as explained previously, the $\Delta$ T Completion is insignificant. By this reason $\Delta$ T Completion will be considered in the lost value by rounding down of the $\Delta$ T. For Example in Hamming Coder $\Delta$ T is $9,575cc$, in Time Slices this number is $31.9166$. Round down the number of $\Delta$ T in Time Slices is $31$ TTS. Using this method, the $\Delta\, T$ was calculated $40$ TTS and $17$ TTS for the Bsort coder and NMEA coder, respectively. Table 4.4 depicts results for the Preliminary Experiment. These numbers were measured by simulations as shown in Fig. 4.3.

Table 4.4 – Time (ns) in different modes in simulations for Preliminary Technique

| Critical Task(CT) | Shared Mode | Isolated Mode | Total |
|---|---|---|---|
| **Hamming coder** | 63,732.5 | 129,675 | 193,437.5 |
| **NMEA coder** | 116,262.5 | 95,975 | 212,237.5 |
| **Bubble Sort** | 146,837.5 | 281,425 | 428,262.5 |

Fig. 4.3 depicts ModelSim simulation results for the Bubble Sort program, NMEA coder program and Hamming coder program. This figure is explained as follows:

(a) Signal “CT START/STOP DETECTION” (Fig. 4.3.a): this is an internal signal generated by the SBAC to indicate when it detected the first CT instruction (CT 1stInstAddr ) passing through the memory address bus towards Core 0. At this moment, the I-IP sets up this signal to “1”.

Similarly, when it detects the CT last instruction (CT LastInstAddr ), it sets down this signal to "0".

(b) Signal "TDMA BUS-ACCESS POLICY" (Fig. 4.3.b): this is also an internal signal generated by the SBAC to indicate to the AHB Bus Controller which is the bus access policy to be applied to the system at a given moment. "01" indicates that Shared Mode should be selected from that moment on, whereas "00" indicates Isolated Mode.

(c) Signal "GLOBAL MEMORY ACCESS" (Fig. 4.3.c): this signal is used only for simulation purpose in order to observe the moments when the memory system is being accessed by the Cores 0 and 1. When this signal is a logical "1", it means that the memory is being accessed by Core 0, otherwise, it is being accessed by Core 1. It is worth noting that by combining this signal with "CORE0 BUS-ACCESS REQUEST" and "CORE1 BUS-ACCESS REQUEST" one can observe that the memory access is granted in an interleaved way between the two cores.

(d) Signals "CORE0 BUS-ACCESS REQUEST" and "CORE1 BUS-ACCESS REQUEST" (Fig. 4.3.d and 4.3.e, respectively): they are bus-access request signals sent by Core0 and Core1, respectively, to the AHB Bus Controller. These signals are active when they are equal to a logical "1". It is worth noting that during the time period when the bus is operating in the Isolated Mode ("00") the "CORE0 BUS-ACCESS REQUEST" signal is continuously switching from "0" to "1", which means that Core0 is frequently accessing the system bus. However, "CORE1 BUS-ACCESS REQUEST" is always "1" because Core1 is requesting bus access, but the AHB Bus Controller prevents this core from accessing the system bus during the whole period when the Isolated Mode is on.

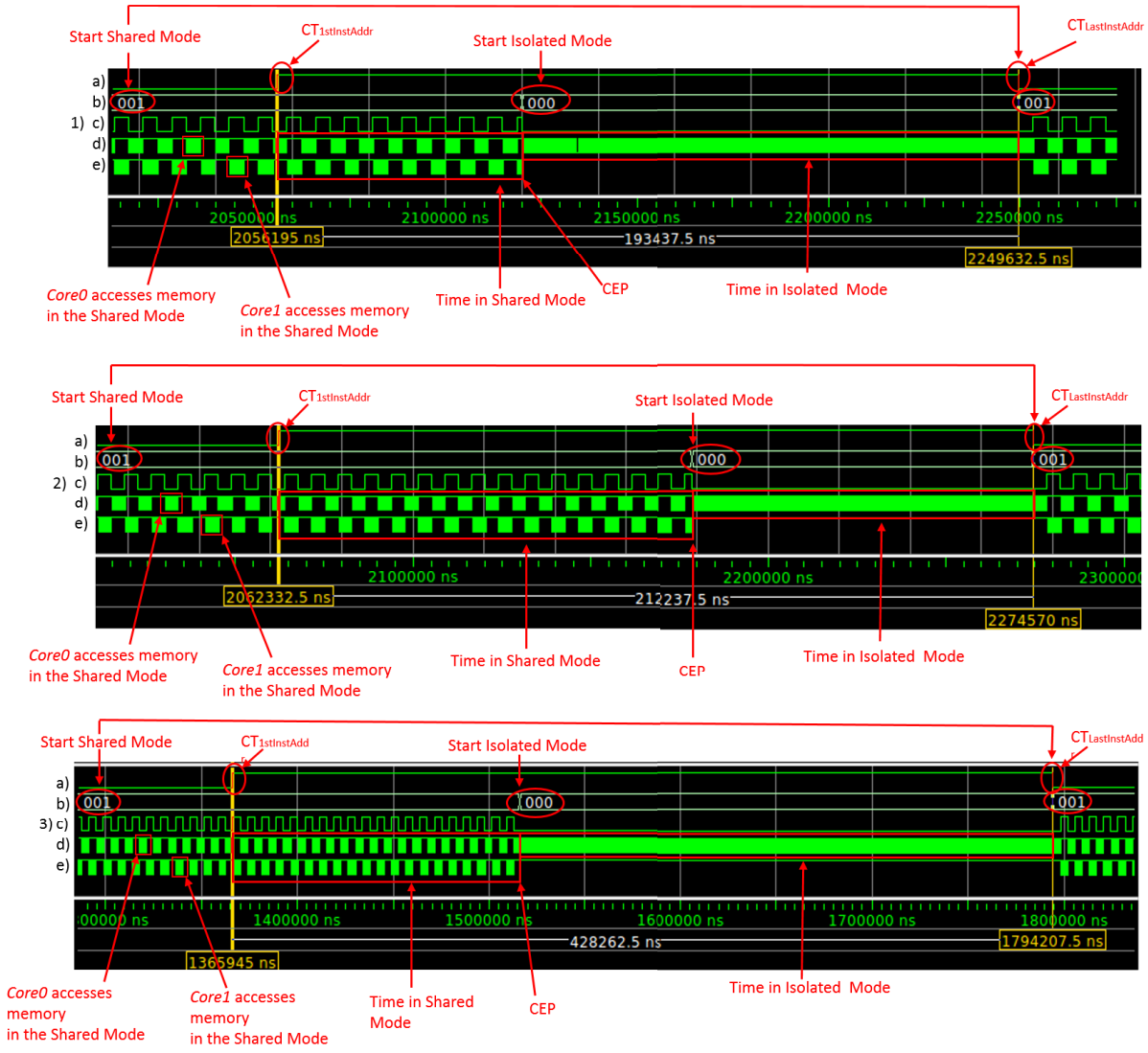


Figure 4.3 – Simulations for Dual-core Preliminary Technique for Hamming, NMEA and Bubble Sort.
Source: Author (2019)

Furthermore, was simulated a quad-core model with the bubble-sort program. The results are similar, but it is important to note that the time used in Shared Mode is the same in the dual-core systems or quad-core system. On the other hand, the time used to execute the CT was different in these models. This occurs because in the quad-core model in the Shared Modes there are two more cores requesting access to memory. In Fig. 4.4 it is possible to note the simulation timeline and the switching modes changing between Shared and Isolated Modes.

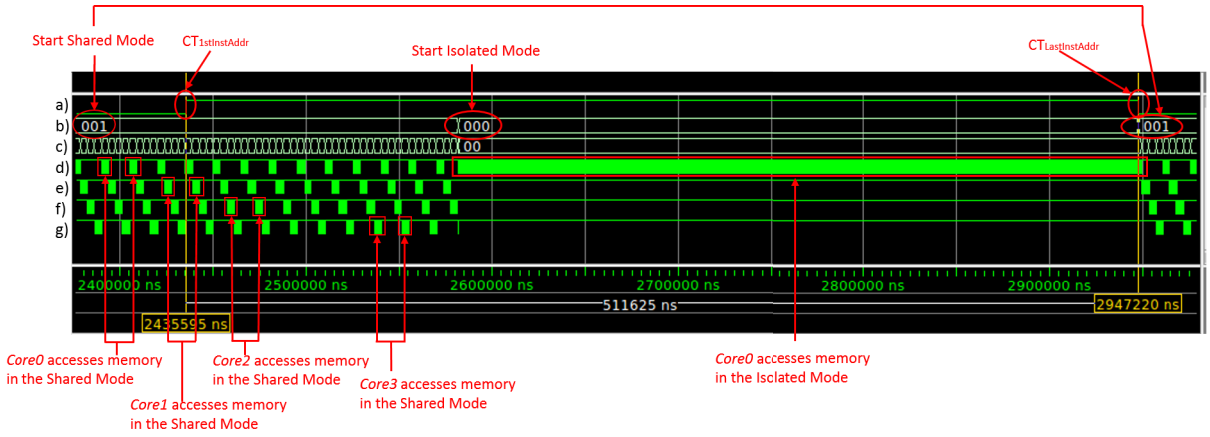


Figure 4.4 – Quad-core Simulation Results for the Bubble Sort Program.
Source: Author (2019)

Such as in Fig. 4.3 c) and d), Fig. 4.4 f) and g) represent the requests of Core 2 and Core 3, respectively. Nevertheless, in every simulation, the system guarantees the deadline and executes the CT without issues. Besides this, the system operates in the Shared Mode for a certain period of time. On the other hand, in the Conventional Technique, the system enters in Isolated Mode until the end of the CT. This period of time in the Isolated Mode represents a time larger than the period of time in the Isolated Mode, as shown in Fig. 4.4. Note that even in Fig. 4.3 the time in the Isolated Mode will be shorter than Conventional Method. In Table 4.5 it is shown a comparison between the Conventional Method and the Preliminary Method. These values were obtained from the simulation represented in Fig. 4.3.

Table 4.5 – Comparison in the Isolated Mode between Conventional Method and Preliminary Technique

| **Critical Task(CT)** | Execution Time In Isolated Mode in dual-core system(ns) | | |
|---|---|---|---|
| | Conventional Technique | Isolated of Preliminary | Total of Preliminary |
| Hamming Coder | 162,687.5 | 129,675 | 193,437.5 |
| NMEA Coder | 157,887.5 | 95,975 | 212,237.5 |
| Bubble Sort | 353,125 | 281,425 | 428,262.5 |

Finally, after these simulations, it is possible to note an increase in the execution time of the CT. This increase is generated by the duration of the Shared Mode after the detection of the CT. Sharing the access bus between both cores for a certain time after the detection of CT makes possible to improve the usability of the available time and also guarantees the execution before the deadline

comes. Similar simulations will now be presented to validate the Advanced Technique and after a comparison between techniques will be analyzed.

## 4.2 Advanced Experiment

Such as The Preliminary Experiment, the programs used have been maintained to facility comparisons between methods. The differences of these methods only stay located in the method of changing the access for the cores and the logic used for these changes. Moreover, in VHDL the block changing is the DSU logic to change the access. The block of detect instruction will be the same, while some minor changes implemented in the block of AHB-controller. Some different levels of distribution were inserted in TDMA-controller because this modification of the signal to choose the Shared Modes was greater in comparison to the other techniques. The simulation of Advanced Technique, was executed in a dual-core system and after this was implemented in a quad-core system with two variant styles for this technique. The first variant technique is the simplest technique with only one switching mode for systems with one critical task. The second variety is switching mode specifically for each task, without rounding according to percent rate in the Table 3.1, with a limited number of critical tasks. Lastly, the entire Advanced Technique without limit of critical tasks because of the rounding with a more suitable switching mode for every task. Although, in the Table 4.6 it is possible to check the data and the percentage of the distribution of each critical task. Below the Table 4.6 will show the simulations in dual-core with the most complete Advanced Technique.

Table 4.6 – Configurations of the programs to Advanced Technique

| **Critical Task(CT)** | Data of the Programs to Advanced Mode | | |
|---|---|---|---|
| | WCET(cc) | Deadline defined(cc) | Shared Mode |
| Hamming Coder | 14,769 | 20,000 | V |
| NMEA Coder | 16,964 | 24,000 | IV |
| Bubble Sort | 35,762 | 60,000 | III |

The signals presented in the Fig. 4.5 follow the same names and represented previously section. Moreover, the time measured in these simulations will be allocated in Table 4.7, in comparison with the usual method. The entire comparison will be discussed in the next section will more details to compare.

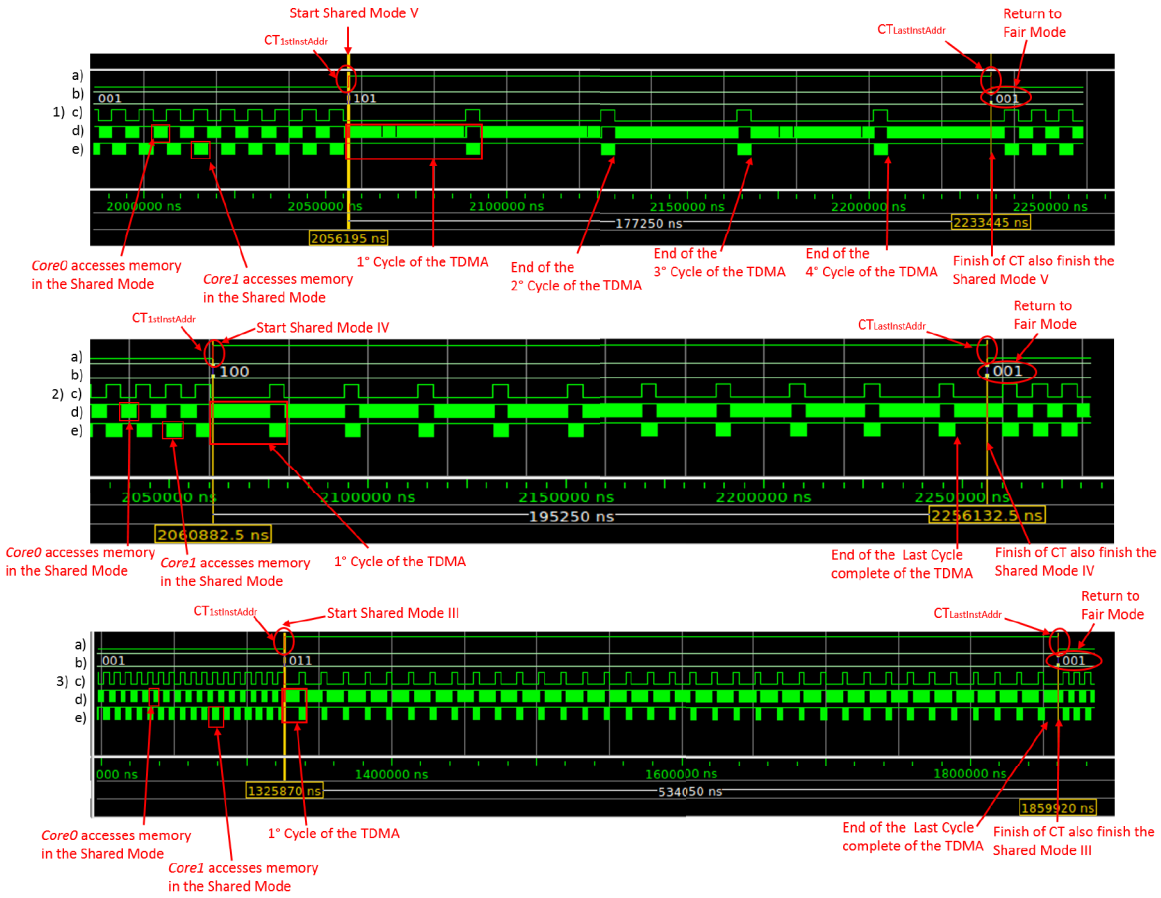


Figure 4.5 – Simulations for Dual-core in the Advanced Technique for Hamming, NMEA and Bubble Sort.

Source: Author (2019)

The values from de simulation will be allocated in the Table 4.7, but it is important to note that the Deadline set for bubble sort in Advanced Technique is different than previously used in the Preliminary Technique. This different value was changed to show more different switching modes. If was used the same Deadline set in the Preliminary Technique $(48,000cc)$, the result for the time was $390587.5ns$ in the Shared Mode V, with ninety percent of the memory access to Critical Core. In the next Section, this value will be used to compare with other proposed, to maintain the same deadline for that comparison.

Table 4.7 – Comparison between usual method and Advanced Technique

| **Critical Task(CT)** | Time Measured(ns) | |
|---|---|---|
| | Usual Method | Advanced Technique |
| Hamming Coder | 162,687.5 | 177,250 |
| NMEA Coder | 157,887.5 | 195,250 |
| Bubble Sort | 353,125 | 534,050 |

After these dual-core simulations of the Advanced Technique, was displayed the results for a quad-core system with bubble sort program. The Shared Modes were represented by Table 3.3. The value increased to the execution time of the critical task does not violate Deadline. In addition, by maintaining a specific Shared Mode it is possible to ensure execution before the deadline. In Fig. 4.6 is presented Advanced Technique in a quad-core system where appears another Shared Mode also ensuring a guarantee of the execution of the CT.

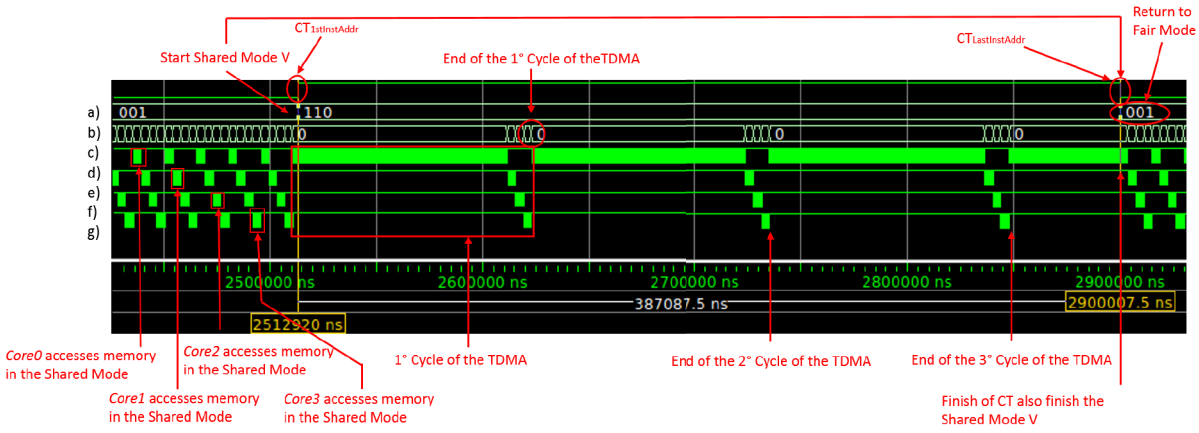


Figure 4.6 – Quad-core Simulation Results for the Bubble Sort Program in the Advanced Technique. Source: Author (2019)

Although, with the intent to explore all possibilities, the deadline set before was changed to demonstrate other options for the AHB-controller. This simulation reveals the functionality of this technique with various options of the TDMA-controller. These results can be seen in Table 4.8, note that in every test the time necessary to execute the critical task does not reach the Deadline and not abort the access of other cores.

Table 4.8 – Simulation results for Advanced Technique in Quad-core

| | Total Time | WCET | Deadline | Shared Mode | TDMA Cycle | Average |
|---|---|---|---|---|---|---|
| Hamming Coder | 11,775 cc | 14,769 cc | 35,000 cc | I | 3/6 | 50% |
| NMEA Coder | 14,425 cc | 16,964 cc | 20,000 cc | V | 27/30 | 90% |
| Bubble Sort | 34,238 cc | 35,762 cc | 48,000 cc | IV | 12/15 | 80% |

In fact, the percent reserved for the Critical Core does not totally delivery. This occurs because of the delay to finish the instruction changing the access to the next core, mentioned in the previous section. Even though the distribution is not exactly as the number configured along the time this contrast gets closer. To show this mismatch the Fig. 4.7 demonstrates the delay to the transition cited above.

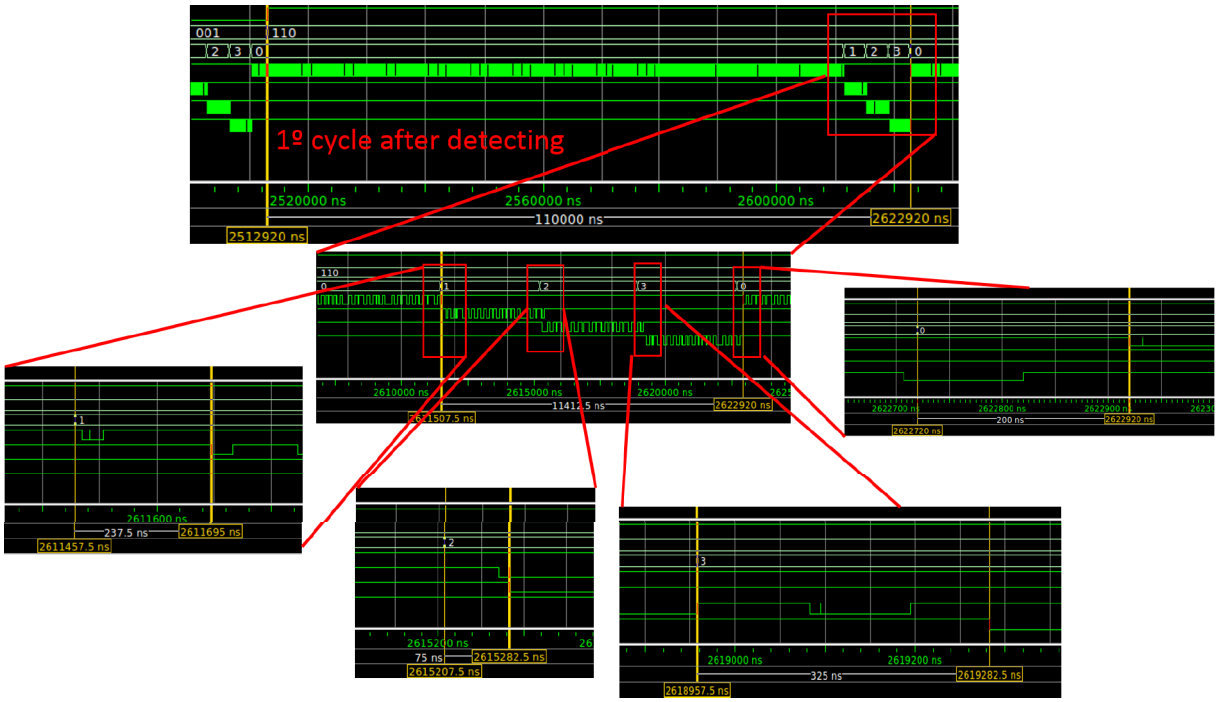


Figure 4.7 – Zoom of the Quad-core Simulation for the Bubble Sort Coder in the Advanced Technique.
Source: Author (2019)

In Table 4.9 is presented the delay that appears in the first cycle completed in Fig. 4.6. Using the same method, based on the simulation in Fig. 4.7 was measured the effective percent delivery to the Critical Core with a width of $300cc$ for each Time Slice. Table 4.10 presents theses measures; it is for every period completed in the selected mode. Note that in the Fig. 4.7 there were three completed periods because in the end of the critical task the system returned to the Native-Mode. As a result of this interruption, the period was not terminated and would represent a number far enough

in comparison to the previously chosen by the AHB-control. Therefore, only the completed period of division will be analyzed.

Table 4.9 – Delay to switching the Access

| Period | Delay (ns) current core to next core | | | |
|---|---|---|---|---|
| | core 0 to 1 | core 1 to 2 | core 2 to 3 | core 3 to 0 |
| First | 237.5 | 75 | 325 | 200 |
| Second | 125 | 125 | 275 | 50 |
| Third | 112.5 | 150 | 150 | 175 |

Table 4.10 – Effective access dedicated to Critical Core for each period

| Period | Total Time (ns) | To core 0 (ns) | To other cores (ns) | Effective % |
|---|---|---|---|---|
| First | 110,000 | 98,587.5 | 11,412.5 | 89.625 % |
| Second | 112,350 | 101,050 | 11,300 | 89.942 % |
| Third | 112,625 | 101,212.5 | 11,412.5 | 89.866 % |

This delay becomes insignificant as the Width of Time Slice increases. By this, was changing the width of Time Slices to show the effective access dedicated to Critical core with different Time Slices. This is displayed in the Fig. 4.8 below:

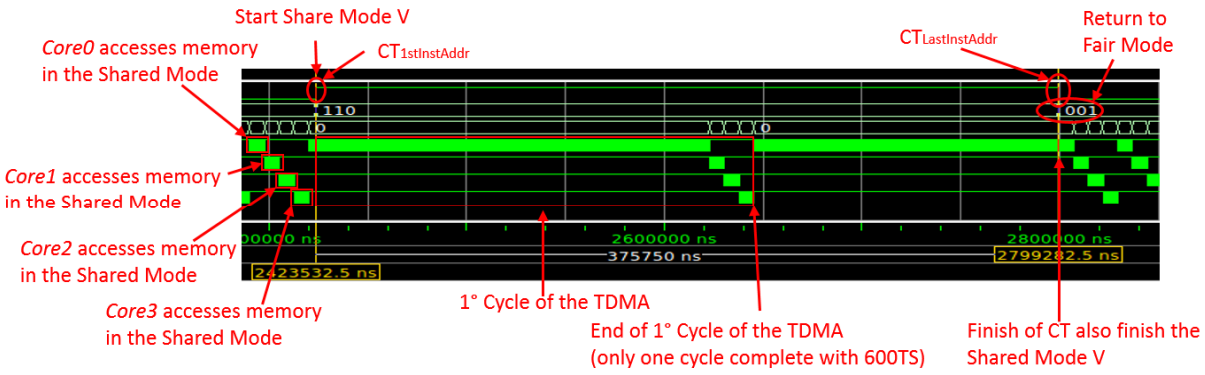


Figure 4.8 – Quad-core Simulation with 600 cc Width of Time-Slices for the Bubble Sort Coder in the Advanced Technique.

Source: Author (2019)

By the fact of the decrease of periods completed by the controller after the detection of the CT, the comparison between different widths of Time Slices will be restrained for the only first period completed. If a large width of TTS had chosen it would be necessary to change the program to permit one complete period of comparison. Therefore, as larger the width of the time slice, less the delay between switches affects the system. Table 4.11 and Table 4.12 were obtained using different widths of TTS in a quad-core system with BSort coder.

Table 4.11 – Delay to switching the access of the first cycle with different width of Time-Slices

| Width of TTS | Delay (ns) current core to next core | | | |
|---|---|---|---|---|
| | core 0 to 1 | core 1 to 2 | core 2 to 3 | core 3 to 0 |
| 100cc | 312.5 | 187.5 | 175 | 262.5 |
| 600cc | 150 | 187.5 | 100 | 250 |
| 900cc | 200 | 175 | 200 | 162.5 |

Table 4.12 – Effective access dedicated to Critical Core for first period with different width for TTS

| Period | Total Time (ns) | To core 0 (ns) | To other cores (ns) | Effective % |
|---|---|---|---|---|
| 100cc/TTS | 36,987.5 | 33,275 | 3,712.5 | 89.962 % |
| 600cc/TTS | 221,937.5 | 199,325 | 22,612.5 | 89.811 % |
| 900cc/TTS | 326,450 | 292,725 | 33,725 | 89.669 % |

In the system set with $100cc$ there are 10 cycles of switching completed before the end of the critical task, instead the system sets with $900cc$ that presents only one completed cycle. With arithmetic using the effective percentages delivered to Critical Core results in $90,154\%$. This indicates that delay between switches affects significantly the operation with this reduced time slice width. By the fact, with so many switches was even delivered more percentage rate than the necessary, this number of switching cycles results in the execution time $353,137.5$, if consider only the amount of time to Critical Core.

Also, this chapter presented a lot of information about the systems with the intent to validate the AHB-control and other blocks to support the entire functionality of the approach. With these simulating experiments it is possible to relate the isolated time between tasks and analyze the shared

time. Moreover, was presented the differences in which the choice of the width of Time Slice and how this choice affects the switching cycles for a Critical Task.

# 5. EVALUATION

In this section, it will be presented and discussed some comparisons between both techniques. A comparison of the approaches aims to elucidate doubts about the time of the execution and comparisons between simulations displayed previously. In summary, in terms of definitions presented before for mixed-critical multi-core systems this approach could be defined as a mix between task allocation and communication resources. By this fact, it is necessary to allocate all the critical tasks in the Critical Core. Moreover, also use a TDMA-based resource on the memory bus. The definition of a mixed approach between these two definitions appears suitable for the proposed approach.

In the first view of the approaches presented in this work, it is possible to conclude a lot of differences among the techniques presented. The most important in this case is the possibility to maintain the parallelism without affecting the deadline duty. In this way, it gets intrinsic that the best approach, in this case, is the Advanced Technique because it is the only one not to abort the parallelism. In addition, maintaining the parallelism allows several benefits to the system and based on the security margin it is possible to affirm that the system will respect the deadline. In Table 5.1 it is possible to note how the techniques affect the time of execution of the critical tasks and how the control of memory access maintains the parallelism using the overtime delivered by the difference from the WCET and the Deadline.

Table 5.1 – Division of the time between the techniques of this work

| | Total Execution Time Necessary to Execute the CT (ns) | | | | | |
|---|---|---|---|---|---|---|
| Technique | Conventional | Preliminary | | | Advanced | |
| Time | Total | Shared Mode | Only CC | Total | % Only CC | Total |
| Hamming Coder | 162,387.5 | 63,732.5 | 129,675 | 193,437.5 | 90% | 177,250 |
| NMEA Coder | 153,662.5 | 116,262.5 | 95,975 | 212,237.5 | 80% | 195,250 |
| Bubble Sort | 353,175 | 146,837.5 | 281,425 | 428,262.5 | 90% | 390,587.5 |

The time for the Preliminary Technique is $511,625ns$ and for the Advanced Technique is $387,087ns$. In addition, if it is compared to the total time of execution between the simulations in quad-core system for the bsort coder, the Advanced Technique results in a smaller total time than the Preliminary Technique. This occurs because the TDMA-cycle increases when the number of cores

increases, and for this reason the time to finish the CT is longer after the system switches for the isolated mode. The

The Table 5.2 was calculated based in the Table 5.1 aforementioned. In Table 5.2 it is possible to note how the techniques affect the execution time of CTs in comparison with the Deadline for each task. Note that the percentage represented for each task are shorter in conventional technique. Furthermore, the Advanced Technique presents one increase moderate and proportional in the use of the available time to execute the CT in contrast to the Preliminary Technique that increases this useful time distinctly.

Table 5.2 – Division of the time between the techniques of this work

| | Percentage of Time Used in Deadline Time | | |
|---|---|---|---|
| Technique | Conventional | Preliminary | Advanced |
| Hamming Coder | 64.955% | 77.375% | 70.900% |
| NMEA Coder | 51.220% | 70.745% | 65.083% |
| Bubble Sort | 58.862% | 71.377 % | 65.097% |

Moreover, maintaining the full operation of all cores in the system is the priority in this work. By this fact, the Advanced Technique presents the best way to maintain all cores without violating the deadline in comparison to the usual technique and Preliminary Technique.

Another aspect of this work is the Width of the Time Slice and which is the best choice. By the reason of the Tasks chosen being not long, it is possible to affirm that if longer tasks were studied the conclusions about this topic would possible change. For the reduction reason of TDMA cycles as the width of TTS was increased. It should be sustained in mind that with lesser switching cycles, there will be less delay in the entire system. In Respect of the Width of the Time Slice chosen, the most effective in this work was 300cc because it presents more cycles for the three tasks chosen thus enabling a comparison between tasks.

In addition, the delay carried between the access of the cores does not present effects with few cores, but in systems with a greater number of cores need to carefully analyze the effects caused by the delay. So, in those cases where there are a lot of cores it is necessary to consider if the Preliminary Technique does not present more benefit results. After all, the system control in these cases is the easiest.

The overhead of this approach was computed to be as low as 5%. This difference represents a minimal value because there are some minor changes between switching techniques. The Preliminary

Technique does not increase the overhead when the number of cores increase, because it only considers the time between the deadline and WCET, if the cores are changed this width maintains the same. Unfortunately, in the Advanced Technique it is different, because if the number of cores changes the counter increases as well. An increment to the number of the cores entails for changing in the proportion set in percentages allocated for the Critical Core. Although, a core also represents an increase in the overhead and this increase is bigger than the changes to maintain the division of Time Slices.

The main aspect to identify is the increase generated by the increase of the Time Slice counter defined only in the Advanced Technique. Because with larger widths of TTS, a bigger counter becomes necessary and this increase may present some variances of the overhead about this approach. Overall, it is possible to conclude that to maintain a balanced choice between the width of TTS and the number of cores should be necessary not to affect the overhead substantially.

# 6. CONCLUSIONS

After some study and development of techniques based on TDMA access for memory access, it is possible to conclude this thesis present suitable techniques for critical tasks processor management. Although the TDMA technique is the old method used in telecommunications studies, this level of computer architectures is still useful.

We presented a new approach that supports two techniques of switching for mixed-criticality workload execution by fully exploiting processor parallelism, namely "The Preliminary Technique" and "The Advanced Technique". It allows any number of cores to run less-critical tasks concurrently with the Critical Core, which is running the critical task. The proposed approach is based on the use of a dedicated intellectual infrastructure property (I-IP) core named Shared Bus-Access Controller (SBAC), which the first Technique (Preliminary Technique) as follows: when a critical task starts running, the SBAC allows the execution of any number of cores (running less-critical workloads) concurrently with the Critical Core (executing the critical workload) until the Critical Execution Point (CEP) comes. If the SBAC does not take action and changes the policy access in the CEP, the critical workload deadline will be violated if the processor continues running all cores concurrently from that point. At this moment, the SBAC inhibits the non-critical cores to execute less-critical tasks until the completion of the critical task by the Critical Core. Instead, the second technique (Advanced Technique), that works intending to explore all the time available with a lot of configurations for the access the memory. In other words, after the detection of the critical task, the approach sets the best-suited configuration to fully harnessing the time available to perform the critical task without violating the Deadline. A comparison between two techniques is presented and some peculiarities about these techniques are discussed, resulting in the conclusion that the second provides a bigger harnessing to the time, however, it is more complex, while the first can be implemented more easily with a different policy of bus access.

It is worth noting that the SBAC I-IP of the Preliminary Technique can be connected to bus controller implementing not only the TDMA-based bus access policy but implementing any other bus access policy such as First-Come First-Served (FCFS) and Round-Robin approaches. Since the second technique uses specifically TDMA for predicting the end of the task, it is possible to use another policy

of division before critical task arrives and thus turn on TDMA policy at the time. In other words, the system only operates in TDMA policy when the critical task is executing.

The SBAC I-IP was described in VHDL language and coupled with the AHB bus and AHB bus controller to monitor the mixed-criticality execution of a dual-core version of the LEON3 soft-core processor. Moreover, the AHB bus controller implemented the TDMA-based bus access policy. Such an approach was validated by simulating in ModelSim the whole system comprised of the processor, SBAC I-IP, bus, bus controller, shared memory, three different programs representing critical task running in the LEON3 Core0 and a (less critical) application workload mapped to Core1. The experimental results demonstrated that the proposed approach is very effective in combining system high performance with critical task schedulability within the deadline. This approach can be applied to any type of processor, considering that the designer is able to collect two signals from the processor ("Program Counter" and "Annul"). The latter signal indicates if the current instruction in the pipeline was executed or not.

# BIBLIOGRAPHY

[1] Arlock, C. C.; Linderoth-olson, E. "A Practical Comparison of Scheduling Algorithms for Mixed Criticality Embedded Systems".

[2] Audsley, N. C. "On priority assignment in fixed priority scheduling", vol. 79–July 2000, 2001, pp. 39–44.

[3] Baruah, S.; Bonifaci, V.; D'Angelo, G.; Li, H.; Marchetti-Spaccamela, A.; Megow, N.; Stougie, L. "Scheduling real-time mixed-criticality jobs", *IEEE Transactions on Computers*, vol. 61–8, 2012, pp. 1140–1152.

[4] Baruah, S.; Guo, Z. "Mixed-criticality scheduling upon varying-speed processors", *Proceedings - Real-Time Systems Symposium*, 2013, pp. 68–77.

[5] Baruah, S.; Vestal, S. "Schedulability analysis of sporadic tasks with multiple criticality specifications", *Proceedings - Euromicro Conference on Real-Time Systems*, 2008, pp. 147–155.

[6] Baruah, S. K. "Fixed-priority scheduling of dual-criticality systems".

[7] Baruah, S. K.; Burns, A.; Davis, R. I. "Response-time analysis for mixed criticality systems", *Proceedings - Real-Time Systems Symposium*, 2011, pp. 34–43.

[8] Betke, K. "The NMEA 0183 protocol", *Standard for Interfacing Marine Electronics Devices,* . . . , –January 2000, 2001, pp. 1–28.

[9] Burns, A.; Davis, R. "Mixed Criticality Systems - A Review", *Department of Computer Science, University of York, Tech. Rep*, –Ninth edition, 2017, pp. 69.

[10] Center for Sustainable Chemical Technologies. "Brochure 2018", 2018, pp. 34.

[11] Cheng, L.; Huang, K.; Chen, G.; Hu, B.; Knoll, A. "Mixed-criticality control system with performance and robustness guarantees", *Proceedings - 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 11th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems*, 2017, pp. 767–775.

[12] Cilku, B.; Crespo, A.; Puschner, P.; Coronel, J.; Peiro, S. "A TDMA-Based arbitration scheme for mixed-criticality multicore platforms", *Proceedings of 1st International Conference on Event-Based Control, Communication and Signal Processing, EBCCSP 2015*, –June, 2015.

[13] Cobham Gaisler. "BCC User ' s Manual", –June, 2017.

[14] Cucu-Grosjean, L.; Santinelli, L.; Houston, M.; Lo, C.; Vardanega, T.; Kosmidis, L.; Abella, J.; Mezzetti, E.; Quiñones, E.; Cazorla, F. J. "Measurement-based probabilistic timing analysis for multi-path programs", *Proceedings - Euromicro Conference on Real-Time Systems*, 2012, pp. 91–101.

[15] De Niz, D.; Lakshmanan, K.; Rajkumar, R. "On the scheduling of mixed-criticality real-time task sets", *Proceedings - Real-Time Systems Symposium*, 2009, pp. 291–300.

[16] Ekberg, P.; Yi, W. "Bounding and shaping the demand of generalized Mixed-criticality sporadic task systems", *Real-Time Systems*, vol. 50–1, 2014, pp. 48–86.

[17] Ferdinand, C.; Heckmann, R. "Worst-Case Execution Time Prediction by Static Program Analysis", *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, vol. 3, 2004, pp. 125a.

[18] G.G. Liversidge, K.C. Cundy, J. B. D. C. "United States Patent (19) 54", 1980.

[19] Green, B. N. "Hardware-Based Approach to Support Mixed-Critical Workload Execution in Multicore Processors", 2015.

[20] Gustafsson, J. "SWEET manual", 2018, pp. 1–62.

[21] Hernandez, C.; Abella, J.; Cazorla, F. J.; Andersson, J.; Gianarro, A. "Towards making a LEON3 multicore compatible with probabilistic timing analysis", *European Space Agency, (Special Publication) ESA SP*, vol. SP-732, 2015.

[22] Huang, H. M.; Gill, C.; Lu, C. "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks". In: Real-Time Technology and Applications - Proceedings, 2012, pp. 23–32.

[23] Ilcev, S. D. "Multiple Access Technique Applicable for Maritime Satellite Communications", *Marine Navigation and Safety of Sea Transportation*, vol. 7–4, 2013, pp. 249–259.

[24] Kelly, O. R.; Aydin, H.; Zhao, B. "On partitioned scheduling of fixed-priority mixed-criticality task sets", *Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011*, vol. 0546244, 2011, pp. 1051–1059.

[25] Kritikakou, A.; Baldellon, O.; Pagetti, C.; Rochange, C.; Roy, M.; Vargas, F. "Monitoring on-line timing information to support mixed-critical workloads", *2013 IEEE Real Time Systems Symposium*, –December, 2013, pp. 9–10.

[26] Lakshmanan, K.; De Niz, D.; Rajkumar, R.; Moreno, G. "Resource allocation in distributed mixed-criticality cyber-physical systems". In: Proceedings - International Conference on Distributed Computing Systems, 2010, pp. 169–178.

[27] Leon, T.; Mips, D.; Gpl, G. N. U. "Interrupts LEON3 CPU", *Source*, 2010.

[28] Li, H.; Baruah, S. "An algorithm for scheduling certifiable mixed-criticality sporadic task systems", *Proceedings - Real-Time Systems Symposium*, 2010, pp. 183–192.

[29] Li, H.; Baruah, S. "Global mixed-criticality scheduling on multiprocessors", *Proceedings - Euromicro Conference on Real-Time Systems*, 2012, pp. 166–175.

[30] Mollison, M. S.; Erickson, J. P.; Anderson, J. H.; Baruah, S. K.; Scoredos, J. A. "Mixed-criticality real-time scheduling for multicore systems", *Proceedings - 10th IEEE International Conference on Computer and Information Technology, CIT-2010, 7th IEEE International Conference on Embedded Software and Systems, ICESS-2010, ScalCom-2010*, –Cit, 2010, pp. 1864–1871.

[31] Pasricha, S.; Dutt, N. "On-Chip Communication Architectures". Morgan Kaufmann Publishers, 2008.

[32] Pellizzoni, R.; Meredith, P.; Nam, M.-y.; Sun, M.; Caccamo, M.; Sha, L. "Handling Mixed-Criticality in SoC-based Real-Time Embedded Systems", 2009.

[33] Pellizzoni, R.; Schranzhofer, A.; Chen, J. J.; Caccamo, M.; Thiele, L. "Worst case delay analysis for memory interference in multicore systems", *Proceedings -Design, Automation and Test in Europe, DATE*, 2010, pp. 741–746.

[34] Rochange, C.; Uhrig, S.; Sainrat, P. "Time-Predictable Architectures". 2014, vol. 9781848215.

[35] Ros, J.; Andrei, A.; Eles, P.; Peng, Z. "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip".

[36] Schranzhofer, A.; Chen, J. J.; Thiele, L. "Timing analysis for TDMA arbitration in resource sharing systems", *Real-Time Technology and Applications - Proceedings*, 2010, pp. 215–224.

[37] Socci, D.; Poplavko, P.; Bensalem, S. "Time-Triggered Mixed-Critical Scheduler on Single- and Multi-processor Time-Triggered Mixed-Critical Scheduler on Single- and Multi-processor Platforms ( Revised Version ) Verimag Research Report n o TR-2015-8", –January, 2016.

[38] Socci, D.; Poplavko, P.; Bensalem, S.; Bozga, M. "Time-Triggered Mixed-Critical Scheduler".

[39] Socci, D.; Poplavko, P.; Bensalem, S.; Bozga, M. "Mixed critical earliest deadline first", *Proceedings - Euromicro Conference on Real-Time Systems*, –October 2015, 2013, pp. 93–102.

[40] Standard, A. A.; Systems, P. "ARINC 653 An Avionics Standard for Safe , Partitioned Agenda • Aerospace Trends", *Seminar*, 2008, pp. 1–30.

[41] Tan, L.; Tan, L.; Timing, K.; Timing, K.; Worst, A.; Worst, A.; Execution, C.; Execution, C. "The Worst Case Execution Time Tool Challenge 2006: Technical Report for the External Test", *In Proc. 2nd International Symposium on Leveraging Applications of Formal Methods (ISOLA'06)*, 2006.

[42] Validation, T.; Systems, R.-t. "aiT Worst-Case Execution Time Analyzer Timing Validation for Real-Time Systems", pp. 0–1.

[43] Version, B. "GRLIB IP Library User ' s Manual", *Design*, –May, 2010, pp. 1–76.

[44] Vestal, S. "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance", *Proceedings - Real-Time Systems Symposium*, 2007, pp. 239–243.

[45] Wilhelm, R.; Mitra, T.; Mueller, F.; Puaut, I.; Puschner, P.; Staschulat, J.; Stenström, P.; Engblom, J.; Ermedahl, A.; Holsti, N.; Thesing, S.; Whalley, D.; Bernat, G.; Ferdinand, C.; Heckmann, R. "The worst-case execution-time problem—overview of methods and survey of tools", *ACM Transactions on Embedded Computing Systems*, vol. 7–3, 2008, pp. 1–53.