Software Parametrization of Feasible Reconfigurable Real-time Systems under Energy and Dependency Constraints

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften der Naturwissenschaftlich-Technischen Fakultät der Universität des Saarlandes
&
Tunisia Polytechnic School
University of Carthage

von

Aicha GOUBAA

Saarbrücken
2022.

Tag des Kolloquiums: 14.12.2022

Dekan: Prof. Dr. Jörn Walter

Vorsitz: Prof. Dr. Dietrich Klakow
Berichterstatter: Prof. Dr. Georg Frey
Prof. Dr. Mohamed Khalgui
Prof. Dr. Luis Gomez
Akademischer Beisitzer: Dr. Klaus Schumacher
Weitere Mitglieder: Prof. Dr. Sofiene Ouni

I dedicate this work to My dear parents Mohamed and Saida,
My two brothers Ayoub and Chokri,
My two sweets sisters, Amel and Besma,
My beloved husband Fethi,
My two young men, I loved more than any other, my sons Haroun and Yaakoub,
My friends and family, for their endless love, support and encouragement.

Acknowledgement

With immense pleasure and deep sense of gratitude, I would like to thank my supervisor Mr. Mohamed KHALGUI, professor at INSAT-University of Carthage, who guided my first steps on the difficult road of research, for his motivation and continuous encouragement in the course of the elaboration of my thesis.

I am also very grateful to my co-supervisor Mr. Georg FREY, Professor at Saarland University, for his constant support, availability, and constructive suggestions, which were determinants for the accomplishment of the work presented in this thesis.

Abstract

Enforcing temporal constraints is necessary to maintain the correctness of a real-time system. However, a real-time system may be enclosed by many factors and constraints that lead to different challenges to overcome. In other words, to achieve the real-time aspects, these systems face various challenges particularly in terms of architecture, reconfiguration property, energy consumption, and dependency constraints. Unfortunately, the characterization of real-time task deadlines is a relatively unexplored problem in the real-time community. Most of the literature seems to consider that the deadlines are somehow provided as hard assumptions, this can generate high costs relative to the development time if these deadlines are violated at runtime. In this context, the main aim of this thesis is to determine the effective temporal properties that will certainly be met at runtime under well-defined constraints. We went to overcome these challenges in a step-wise manner. Each time, we elected a well-defined subset of challenges to be solved. This thesis deals with reconfigurable real-time systems in mono-core and multi-core architectures. First, we propose a new scheduling strategy based on configuring feasible scheduling of software tasks of various types (periodic, sporadic, and aperiodic) and constraints (hard and soft) mono-core architecture. Then, the second contribution deals with reconfigurable real-time systems in mono-core under energy and resource sharing constraints. Finally, the main objective of the multi-core architecture is achieved in a third contribution.

Kurzfassung

Das Erzwingen zeitlicher Beschränkungen ist notwendig, um die Korrektheit eines Echtzeitsystems aufrechtzuerhalten. Ein Echtzeitsystem kann jedoch von vielen Faktoren und Beschränkungen umgeben sein, die zu unterschiedlichen Herausforderungen führen, die es zu bewältigen gilt. Mit anderen Worten, um die zeitlichen Aspekte zu erreichen, können diese Systeme verschiedenen Herausforderungen gegenüberstehen, einschliesslich Architektur, Rekonfigurationseigenschaft, Energie und Abhängigkeitsbeschränkungen. Leider ist die Charakterisierung von Echtzeit-Aufgabenterminen ein relativ unerforschtes Problem in der Echtzeit-Community. Der grösste Teil der Literatur geht davon aus, dass die Fristen (Deadlines) irgendwie als harte Annahmen bereitgestellt werden, was im Verhältnis zur Entwicklungszeit hohe Kosten verursachen kann, wenn diese Fristen zur Laufzeit verletzt werden. In diesem Zusammenhang ist das Hauptziel dieser Arbeit, die effektiven zeitlichen Eigenschaften zu bestimmen, die zur Laufzeit unter wohldefinierten Randbedingungen mit Sicherheit erfüllt werden. Wir haben diese Herausforderungen schrittweise gemeistert. Jedes Mal haben wir eine wohldefinierte Teilmenge von Herausforderungen ausgewählt, die es zu lösen gilt. Zunächst schlagen wir eine neue Scheduling-Strategie vor, die auf der Konfiguration eines durchführbaren Scheduling von Software-Tasks verschiedener Typen (periodisch, sporadisch und aperiodisch) und Beschränkungen (hart und weich) einer Mono-Core-Architektur basiert. Der zweite Beitrag befasst sich dann mit rekonfigurierbaren Echtzeitsystemen in Mono-Core unter Energie und Ressourcenteilungsbeschränkungen. Abschliessend wird in einem dritten Beitrag das Verfahren auf Multi-Core-Architekturen erweitert.

Résumé

L'application de contraintes temporelles est nécessaire pour maintenir l'exactitude d'un système temps réel. Cependant, un système temps réel peut être entouré par des nombreux facteurs et contraintes conduisant à différents défis à surmonter. En d'autres termes, pour atteindre les aspects temporels, ces systèmes peuvent faire face à divers défis, notamment en termes d'architecture, de propriété de reconfiguration, de consommation d'énergie et de contraintes de dépendance. Malheureusement, la caractérisation des échéances des tâches temps réel est un problème relativement inexploré dans la communauté du temps réel. La plupart de la littérature semble considérer que les échéances sont en quelque sorte fournis comme des hypothèses dures, cela peut générer des coûts élevés par rapport au temps de développement si ces échéances sont violées à l'exécution. Dans ce contexte, l'objectif principal de cette thèse est de déterminer les propriétés temporelles effectives qui seront certainement respectées à l'exécution sous des contraintes bien définies. Nous avons choisi de surmonter ces défis de manière progressive. Tout d'abord, nous proposons une nouvelle stratégie d'ordonnancement basée sur la configuration d'un ordonnancement faisable de tâches temps réel de différents types (périodiques, sporadiques et apériodiques) et contraintes (hard et soft) sur une architecture mono-coeur. Ensuite, la deuxième contribution traite des systèmes temps réel reconfigurables en mono-coeur sous contraintes de d'énergie et de partage des ressources. Enfin, l'objectif principal de l'architecture multi-coeur est atteint dans une troisième contribution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Context

Nowaday, real-time systems have an increasingly important in many fields of application, such as automotive, robotics, industry or telecommunications [12, 11, 61]. A real-time system is a system for which the correction depends not only on the accuracy of the results, but also on the time at which these results are provided [67, 95]. The main software part of a real-time system consists of computer processes called tasks. Consequently, to achieve its functionalities, a real-time system may execute tasks of different types: periodic, aperiodic and sporadic tasks. These tasks can be independent or dependent where they need to cooperate to achieve their missions. Each task has to be completed in an interval of certain length, the beginning of the interval is called release time and the end is called deadline [93], it represents the time at which specific task has to be completed. A deadline can be (i) hard where its non-respect is considered as a complete system failure and may cause catastrophic consequences on the system under control, or (ii) soft where its non-respect decreases the utility of the system result without causing a complete system failure [68].

In order to verify whether the temporal correctness of a real-time system is valid or not (whether task executions always complete before their specified deadlines), it is necessary to perform schedulability analysis. However, a real-time system often needs to execute jointly the hard periodic tasks set mixed with the dynamically occurring soft aperiodic tasks [30]. Thus the scheduling problem of mixed task set has been an essential issue in the design of real-time computing systems. In fact, the task scheduling algorithms in such systems must satisfy the real-time tasks hard deadlines and improve response times for soft ones [53]. The main solution to schedule mixed task set, presented in the literature, is the use of a periodic server, accounted as a periodic task, to manage aperiodic tasks during its service time. Periodic tasks, including the server, are scheduled to meet their deadlines.

Further, The main entity to perform a real-time schedule is the scheduler [94, 70]. Thid entity relies on a scheduling algorithm to select the task or job to be executed. A scheduling algorithm can be: (i) mono-core when it is executed to schedule tasks on mono-core processor [87], where only one task can be executed at the same time, or (ii) multi-core when it is executed to schedule tasks on multi-core processor [87], where multiple tasks can be executed at the same time. In multi-core architecture, a scheduling algorithms can be classified based on task migration. An algorithm which denies any task migration is called partitioned scheduling algorithm. On the other side, an algorithm which allows tasks to migrate at any point during the runtime is called the global scheduling algorithm.

Such systems need to operate continuously without missing the available energy. Thus, the management of energy consumption for real-time systems must be considered. Many solutions were proposed in the literature to solve power supply issues for real-time systems while powering those real-time systems from a renewable energy source [103, 2, 79]. Energy harvesting is the technique for collecting energy from various environmental sources such as solar, wind, etc. However, this type of energy is often weak and unstable. Thus, this constraint should be considered as well the temporal correctness of real-time systems, so that the violation of one of them will lead to system failure. On the other hand, to reduce the energy consumption of real-time systems, many techniques were applied in the literature[101], Dynamic Power Management (DPM)[14] and Dynamic voltage and frequency scaling (DVFS)[62].

Real-time systems often have to interact with their environment that may be changed over time. Thus, real-time systems must constantly be adapted to their environment evolution while providing reconfiguration techniques. Hence, the notion of a reconfigurable real-time system. A reconfiguration is an operation allowing the system to transform its working process in order to adapt to the environment changes. More precisely, reconfigurability means the capability of the system to be adapted to the evolution of its environment while executing an adequate reconfiguration scenario. In fact, reconfiguration scenario is any operation that consists in adding, removing or updating hardware or software components [111]. Based on this definition, a reconfigurable real-time system is considered as a set of implementations where each implementation is executed by the system after a particular reconfiguration scenario.

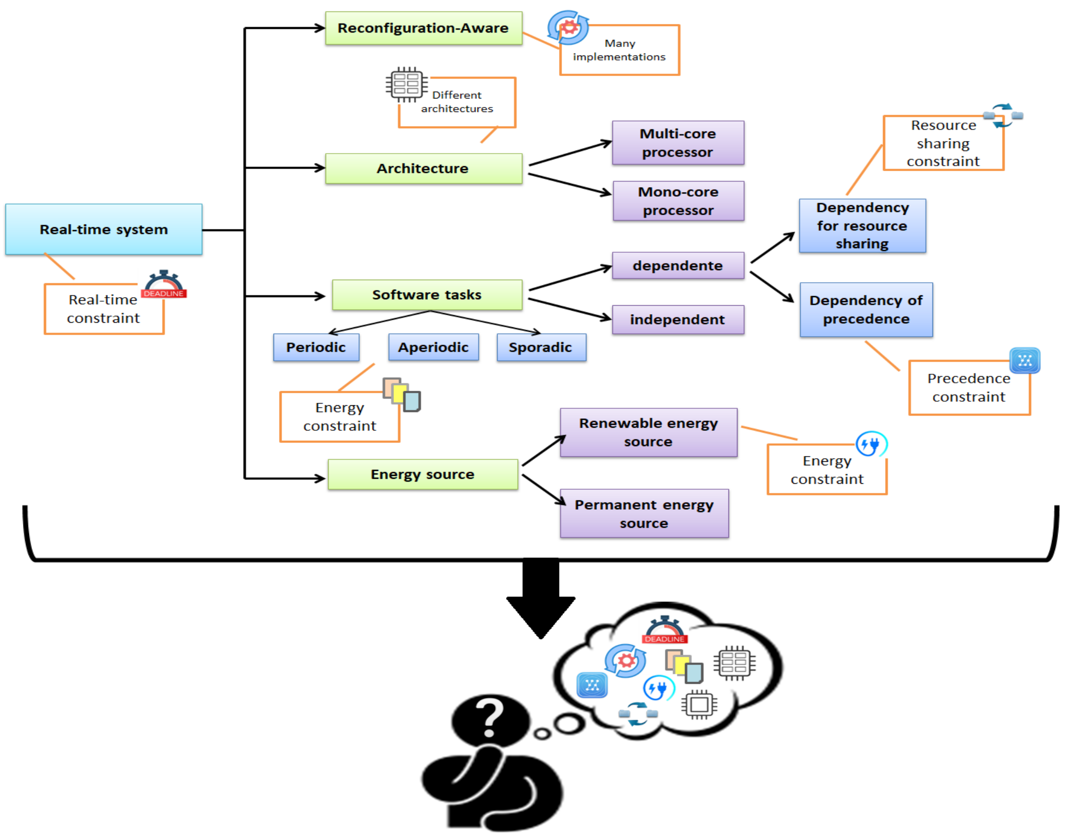


Figure 1.1: Real-time tasks characterization.

As a fianl point, the real-time correctness is the most prominent part of real-time systems feasibility. However, such system may be surrounded and influenced by several factors as presented in figure 1.1 : i) factors outside the real-time system, i.e., external interruptions, source of energy, and ii) factors inside the real-time system,i.e., type of architecture, type of tasks and their dependencies, reconfigurability aware.

1.2 Thesis Problems

As mentioned previously, a real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation, but also upon the time at which the results are produced. Thus, the design of a real-time system must not only incorporate means to tackle functional complexity, but also means to analyze and predict temporal properties. Hence, enforcing timeliness constraints is necessary to maintain the correctness of a real-time system. However, as indicated previously, a real-time system may be enclosed by many factors that lead to different constraints. These constraints may affect the execution of the tasks and may cause some delays in such a way tasks may exceed their deadlines. i.e., may cause a system failure. Hence, predictability of the system behavior and factors is an important concern in these systems. The factors that will be considered in this thesis are:

- Type of tasks: scheduling both periodic, sporadic and aperiodic tasks in real-time systems is a much more difficult than scheduling a single type of tasks. In fact, as presented in figure 1.2, the invocation of an aperiodic task may cause the violation of hard periodic deadlines. Thus, it is a key challenge to address how to calculate the effective deadlines (hard and soft) of the different mixed tasks to guarantee that all tasks, with hard deadlines, will always meet their deadlines.

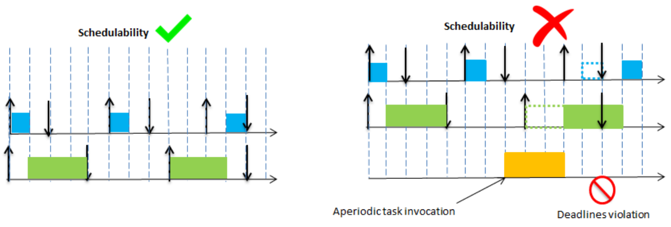


Figure 1.2: Mixed tasks problem.

- Reconfigurability: Real-time system are employed to assist humans by performing tasks that require an interaction with the physical world. In fact, these systems need to modify thier functions to be adapted to environment changes by adding/removing/updating a hardware/software component. In fact, as presented in figure 1.3, if the system need to add a periodic task to complete its treatment, the real-timle corcetness may be violated. Thus, it is a key challenge to address how calculate effective deadlines to guarantee system's feasibility under any reconfiguration secenario(adding/removing/updating task).

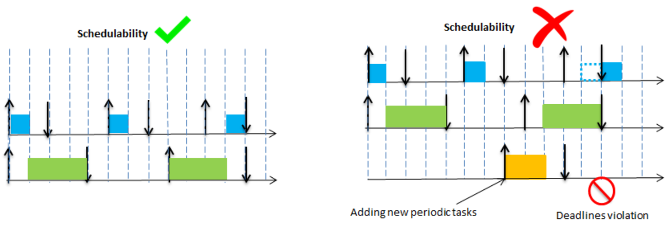


Figure 1.3: Reconfigurability problem.

- Energy availability: Typically, the harvested energy availability varies with time, depending on environmental conditions. The uncertain energy harvesting availability makes the problem of task scheduling more challenging, and may cause the violation of the energy constraints, i.e., when a real-time software task execution is incomplete because the required energy necessary to process its job by its deadline is not available, as presented in figure 1.4.

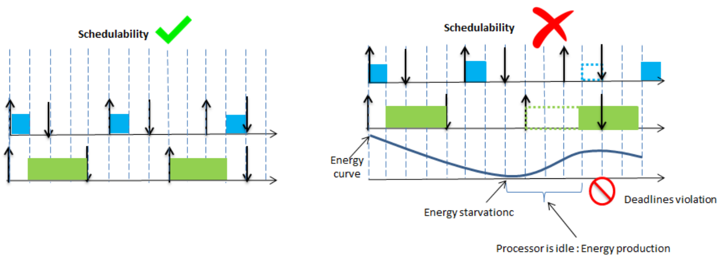


Figure 1.4: Energy availability problem.

- Task dependency: as presented in figure 1.5, a task may violate its deadline as it is blocked due to: i) precedence constraint (it waits message from its successor) or ii) resources sharing (it waits that the resource is unlocked).

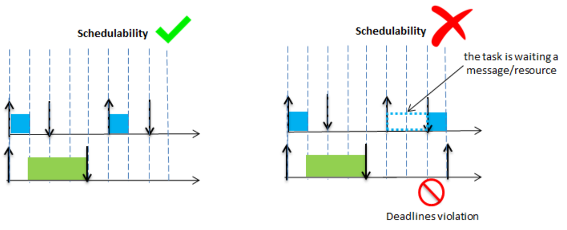


Figure 1.5: Task dependency problem.

Thus, In order to ensure a required real-time performance, the designer should predict the behavior of a real-time system by ensuring that all tasks meet their

hard deadlines while considering the different factors that could infect the real-time system correctness (architecture, type of tasks, reconfigurability, energy, and tasks dependency. Unfortunately, the characterization of real-time tasks deadlines is a relatively unexplored problem in the literature. Moreover, the techniques to calculate systems' deadlines are very seldom presented, and they provide limited support to the designer to forecast real-time properties. Thus, It is a primordial need to have an effective solution that can provide designers with real-time properties that should maintain real-time system correctness.
As presented in figure 1.6, we can recapitalize our problems with the fact of how parameterizing feasible real-time system under all predicted constraints that can affect the system execution.

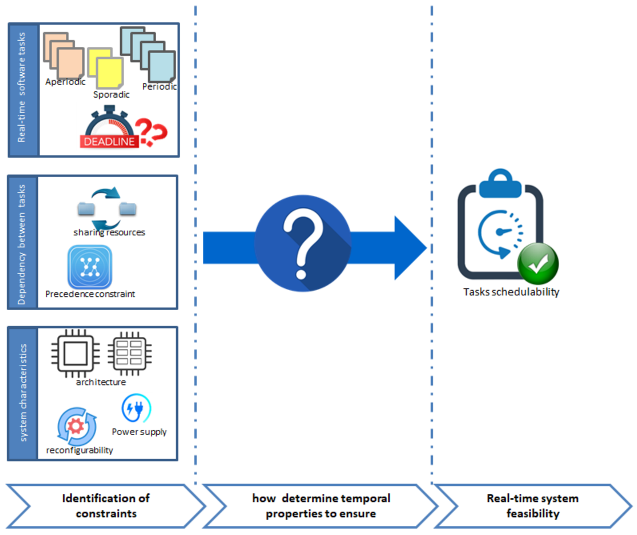


Figure 1.6: The main problem.

1.3 Thesis Contribution

Fom the design phase of real-time systems, it would be necessary to be able to predict the potential of the design retained with regard to the satisfaction of the temporal requirements. However, there is no tooled engineering methodology to identify the temporal properties of systems. In fact, the characterization of deadlines is a relatively unexposed problem in the real-time community. Most of the literature seems to consider that the deadlines are somehow provided as hard assumptions, and the designer have to verify if these deadlines are correct or no by performing many scheduling tests. This may take a lot of time especially with a big number of tasks and of constants, and generates high costs relative to the development time, which could be lengthened if it is belatedly realized that the chosen deadlines may be violated at runtime. Thus, to facilitate the task of a designer, we propose a new solutions base on identifying the constraints that may affect the real-time system, and then determining the effective temporal properties that will surely be met at runtime under different constraints that may affect its correctness. In fact, we went to overcome these constraints in a step-wise manner. Each time, we elected a well-defined subset of challenges to be solved and then we parametrize a feasible real-time system under the given constraints. As a consequence, we propose in this thesis three contributions, each of which is done offline and works around a subset of possible constraints that may affect any real-time system. These contributions are presented below:

- The first contribution focuses on mixed tasks set with hard and soft temporal constraints executing on mono-core architecture. The proposed approach, presented in figure1.7, deals with a combination of mixed sets of tasks. First, this approach builds a New Periodic Server noted NPS to serve aperiodic tasks, it allows meeting real-time constraints of aperiodic tasks. Second, it computes deadlines to meet hard deadlines of periodic and sporadic tasks while considering aperiodic tasks invocations. The proposed approach greatly improves response times for soft deadline aperiodic tasks and computes hard deadlines for both periodic and sporadic tasks that ensure system feasibility.

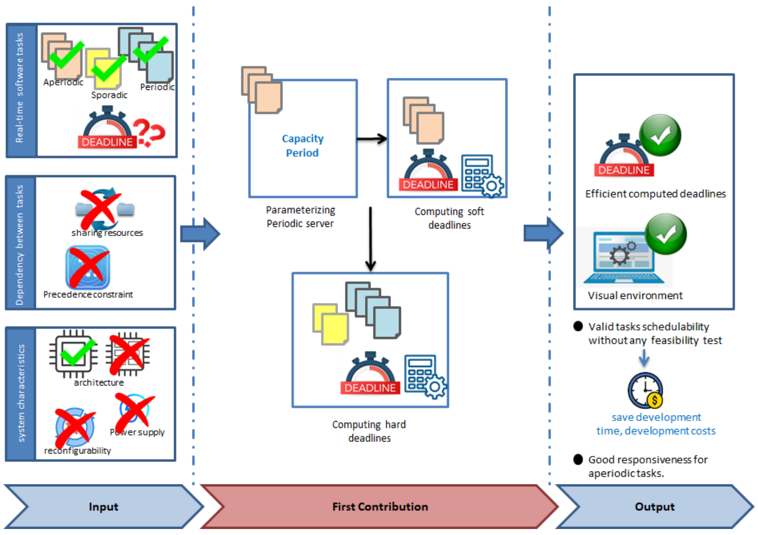


Figure 1.7: First contribution.

- The second contribution focuses on reconfiguration aware, renewable energy, and resource sharing constraints under mono-core architecture. The proposed method, presented in figure1.8, consists of three solutions to calculate the deadlines of tasks. The first serves to compute the deadlines ensuring the real-time system feasibility and also minimizes the number of context switches by assigning the highest priority to the task with the smallest maximum deadline. The second computes the deadlines ensuring the respect of energy constraints, and the third computes the deadlines ensuring the respect of resource sharing constraints. These three solutions calculate the possible deadlines of each task in the hyper-period of the corresponding implementations. In fact, we assume that the reconfigurable real-time system is defined as a set of implementations, each of which is encoded by real-time periodic software tasks and executed under well-defined conditions to adapt the system to any related environment evolution. We develop a new simulator called DEADCALC that integrates a new tool called RANDOM-TASK for applying and evaluating the proposed solutions.

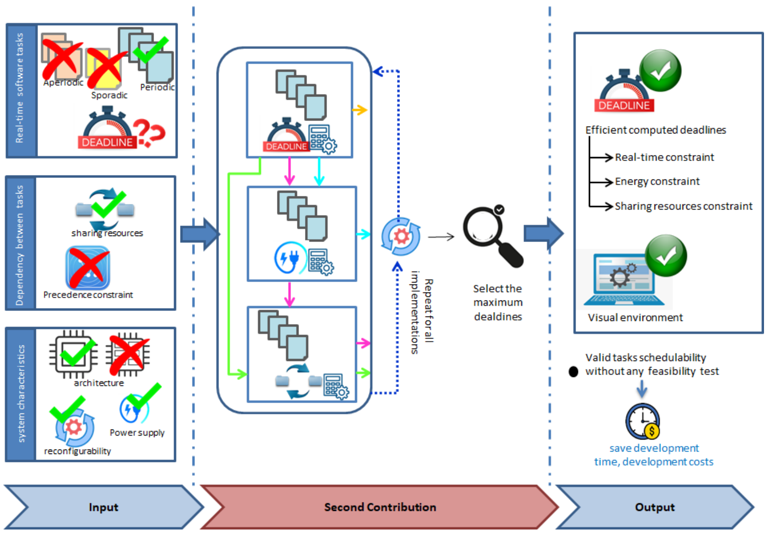


Figure 1.8: Second contribution.

- The third contribution focuses on mixed tasks, reconfiguration aware, renewable energy, and precedence constraints under multi-core architecture. The proposed approach, presented in figure1.9, serves to effectively compute deadlines allowing for tasks and messages to meet related constraints. This method consists of two phases:

  – The first one defines different operating modes each of which is characterized by energy and frequency parameters to cope with the energy availability issue. In each operating mode, the execution times of tasks are updated according to the defined processor's frequency. In other words, a given implementation is executed by the system after an updating reconfiguration scenario.

  – The second one calculates the deadlines ensuring real-time system feasibility by considering the invocation of aperiodic task execution and task precedence constraints while considering the communication messages between cores.

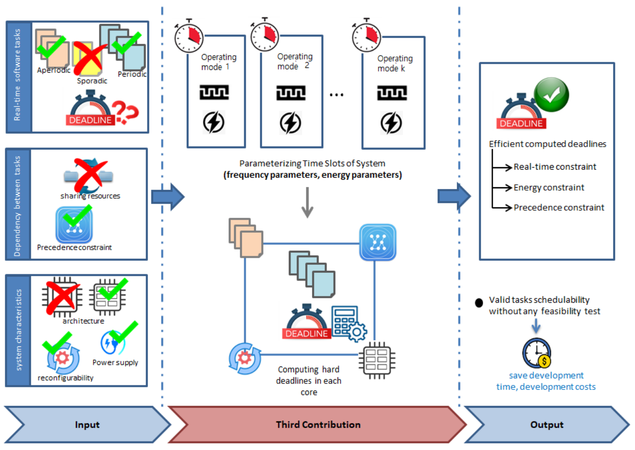


Figure 1.9: Third contribution.

As mentioned above, a reconfiguration scenario consists in the addition, removal or update of hardware and software components.In this thesis, we focus on the scenarios of adding removing or updating software tasks. Furthermore, we address this consideration in two parts:

- First, the reconfiguration is based on adding/removing tasks, as in the case of the second contribution. Each implementation is encoded by a subset of real-time software tasks. These tasks always keep their initial parameters.(See figure 1.10)

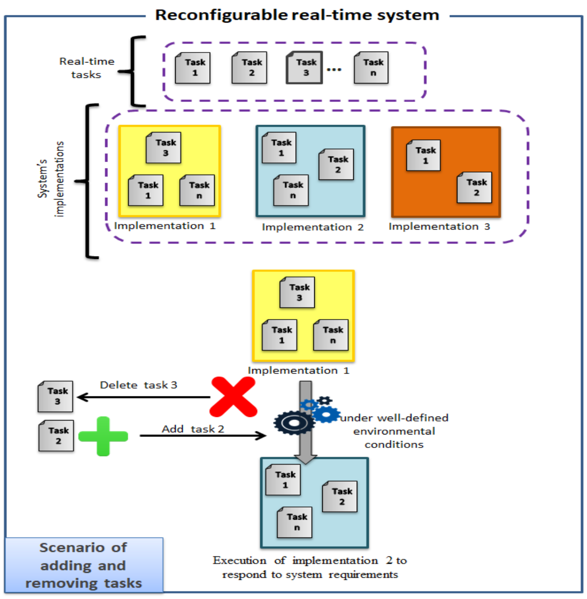

Reconfigurable real-time system
Real-time tasks
Task 1
Task 2
Task 3
Task n
System's implementations
Implementation 1
Implementation 2
Implementation 3
Delete task 3
Add task 2
under well-defined environmental conditions
Execution of implementation 2 to respond to system requirements
Scenario of adding and removing tasks

Figure 1.10: Reconfiguration scenario: adding or removing tasks.

- Second, the reconfiguration is based on updating tasks (execution time) as in the case of the third contribution. Each implementation gathers all tasks of the system. However, the tasks' execution times differ from one implementation to another.(See figure 1.11)

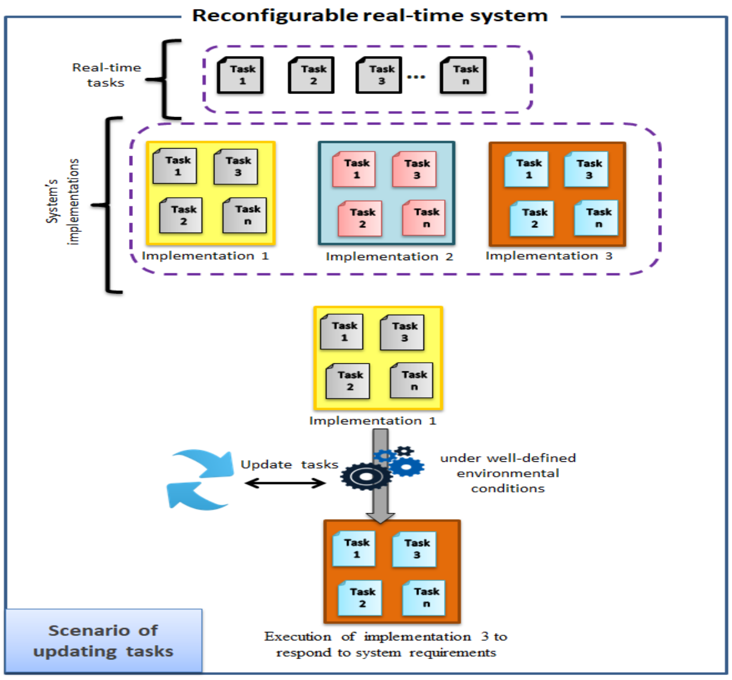

Reconfigurable real-time system
Real-time tasks
Task 1
Task 2
Task 3
Task n
System's implementations
Implementation 1
Implementation 2
Implementation 3
Update tasks
under well-defined environmental conditions
Execution of implementation 3 to respond to system requirements
Scenario of updating tasks

Figure 1.11: Reconfiguration scenario: updating tasks' execution times.

1.4 Thesis Output Tools:

During this thesis, we have developed 3 tools each of which has a specific function:

- GIGTHI-TOOL: is a visual environment that applies the services of the first proposed methodology to compute and display effective deadlines with few clicks in arranged tables and short time. For more details concerning the framework please visit our website https://projects-lisi-lab.wixsite.com/gigthistool.

- DEAD-CALC: is a visual environment serves to apply and evaluate the second contribution, it allows computing efficiently the deadlines of reconfigurable real-time devices to run possibly under energy and resource sharing constraints in all the system implementations. For more details concerning the framework please visit our website https://projects-lisi-lab.wixsite.com/deadcalc.

- RANDOM-TASK: is a random tasks generator tool, i.e, that allows the user to fill in the desired number of tasks and then generates randomly a set of software real-time tasks.

1.5 Thesis Organization:

In chapter 1, we present the general context,the problems and the contributions of the thesis.

In chapter 2, we present the state of the art to describe the current knowledge about the work throughout this thesis. we introduce several generalities related to the real-time systems. In addition, the related works on reconfigurable real-time systems are exposed in both mono-core and multi-core architecture.

In chapter 3, we propose a new approach that configures feasible scheduling of software tasks of various types (periodic, sporadic and aperiodic) and constraints (hard and soft) in the context of dynamic priority, preemptive, monocore scheduling. This approach allows to provide good average response times for soft aperiodic tasks without compromising the hard deadlines of the periodic ones.

In chapter 4, we propose a new methodology for parametrizing feasible reconfigurable systems under real-time, energy and resource sharing constraints. The system consists of a set of tasks implementing the applicative functions that we assume periodic and dependent if they share resources. This methodology serves to assign to each task an effective relative deadline that it will meet in all its implementations.

In chapter 5, we present a new approach that serves to configure feasible scheduling of software tasks with precedence constraints upon a multi-core processor powered by renewable energy harvested from the environment. This approach ensures that each core in the multi-core platform satisfies the real-time constraints, in which tasks must meet their deadlines and have the required energy for their execution.

In chapter 6, we present a summary of our thesis while and restating the research contributions. We introduce the future improvements to enhance the work proposed in this thesis.

1.6 List of Publications:

Journal papers:

- Aicha Goubaa, Mohamed Kahlgui, Frey Georg, Zhiwu Li, MengChu Zhou. (2020). Scheduling periodic and aperiodic tasks with time, energy harvesting and precedence constraints on multi-core systems. Information Sciences 520 : 86-104 . **IF= 5,91**

- Aicha Goubaa, Mohamed Kahlgui, Frey Georg, Zhiwu Li, Abdulrahman Al-Ahmari. (2020). On Parametrizing Feasible Reconfigurable Systems Under Real-Time, Energy, and Resource Sharing Constraints. IEEE Transactions on Automation Science and Engineering 18(3): 1492-1504. **IF= 5.13**

International Conference:

- Aicha Goubaa, Mohamed Kahlgui, Frey Georg, Zhiwu Li: New Approach for Deadline Calculation of Periodic, Sporadic and Aperiodic Real-time Software Tasks. ICSOFT 2020, 452-460. **(B ranked)**

Book Chapter:

- Aicha Goubaa, Mohamed Kahlgui, Frey Georg, Zhiwu Li: Efficient Scheduling of Periodic, Aperiodic, and Sporadic Real-Time Tasks with Deadline Constraints. ICSOFT 2020, 25-43. **(Selected paper)**

Chapter 2

State of the Art

2.1 Introduction

This chapter is devoted to present several generalities related to the real-time systems while discussing some basic concepts like task models, scheduling algorithms, reconfiguration, energy consumption in both mono-processor and multi-processor architecture. We start by presenting the definition of real-time systems and their classifications, and we finish this chapter by giving an overview about the existing approaches and methods in the literature for the synthesis of reconfigurable real-time systems' problem.

2.2 Real-time Systems: Generalities and Definitions

2.2.1 Definition of Real-time Systems

Real-time systems are reactive computing systems that react to input events by performing a correct information processing within a specific time interval, the beginning of the interval is called release time and the end is called deadline [93]. Therefore, successful completion of a real-time operation depends not only on the value of the logical result, but also on the response time to which the result is produced [67, 95]. Moreover, real-time systems are subjected to different environmental and resource constraints like power consumption, cost, resources and memories, etc [72, 45].

As presented in figure 2.1, a real-time system is in permanent interaction with its external environment, generally represented by a physical process, in order to control its evolving behavior over time [13]. Such system must be capable of detecting and reacting to changes in the state of the environment, carrying out processing operations that make it possible to produce a change of state at the output within a limited time interval. The interaction of the system with the environment is obtained by acquiring information from sensors in the form of interruptions, and then reacting to these interruptions by sending adequate commands via actuators [18, 66].

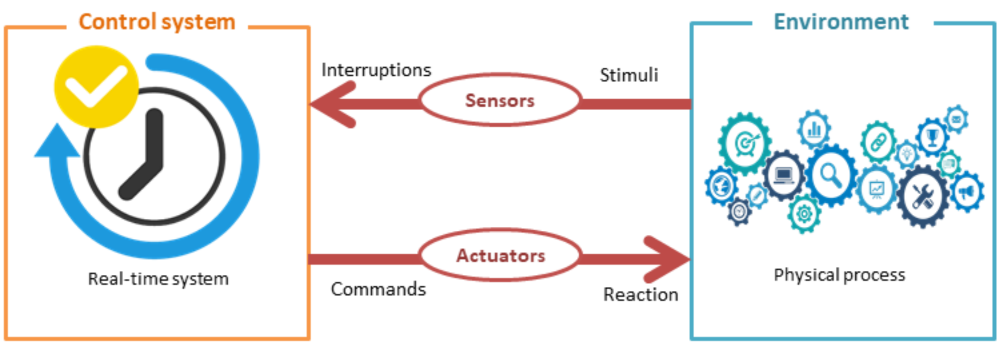


Figure 2.1: Real-time system structure.

2.2.2 Classification of Real-time systems

Depending on the consequences that may occur because of a missed deadline, three categories of real-time system can be distinguished [71, 74]:

- **Soft real-time system**: Such a system considers that missing a time constraint (deadline) is acceptable but undesirable [48]. In other words, it is a system whose operation is degrade if results are not produce according to the specified timing requirement but without causing serious consequences (see figure 2.2 curve a). Example: Console hockey game.
- **Firm real-time system**: Such a system considers that missing a few deadlines will not lead to total failure, but missing more than a few can lead to catastrophic system failure [31, 50] (see figure2.2 curve b). Example: Multimedia applications.
- **Hard real-time system**: Such a system considers that producing results after a given deadline as a system failure [109, 69], i.e. it is necessary for the system to respond within the specified delay, otherwise this could lead to serious consequences (see figure 2.2 curve c). Example: Aircraft systems. Consequently, a hard real-time system should be predictable, deterministic, and reliable:
  - Predictability: The performance of the real-time application must be defined in all possible cases in such a way as to ensure compliance with time constraints in the worst case.
  - Determinism: There is no uncertainty about the behavior of the system. In other words, for a given context, the behavior of the system is always the same.
  - Reliability: In real-time systems, reliability concerns compliance with real-time constraints.

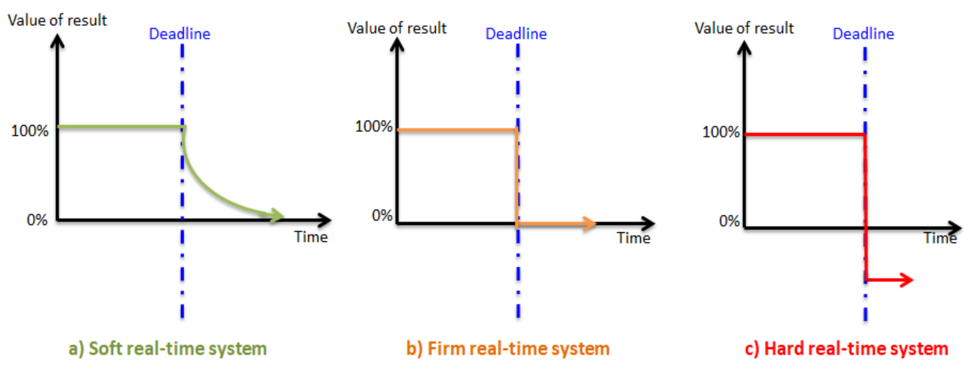


Figure 2.2: A schematic visualization of the value of a late computational result under different real-time systems.

2.2.3 Architecture of Real-time Systems

A real-time system is a combination of computer hardware coupled with proposed software that operates under time constraints to meet performance criteria [84] (see figure 2.3).

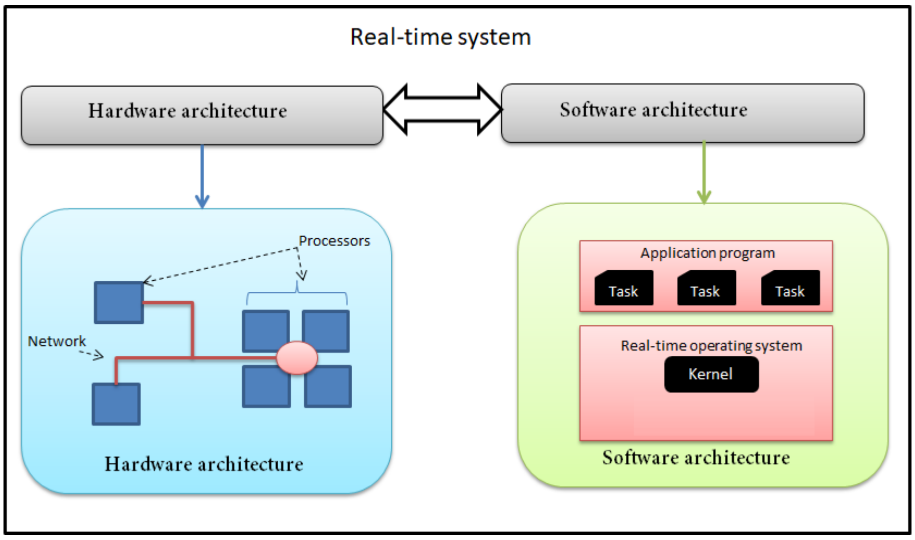


Figure 2.3: Real-time system architecture.

2.2.3.1 Hardware Architecture

The hardware architecture integrats several complex and heterogeneous components such as processors (CPUs), communication networks, storage components, memories, input/output peripherals, etc. As schown in figure 3.2, these different hardware components interacts together under different manner which leads to three categories of architecture [89] :

- Monoprocessor architecture: a single processor works to perform system tasks.

- Multiprocessor architecture: multiple processors operate in parallel while sharing memory, bus, peripherals, computer clock, etc.

- Distributed architecture: A set of nodes each of which is represented as a mono-processor or a multi-processor architecture and which communicate via networks and work in parallel without sharing memory.

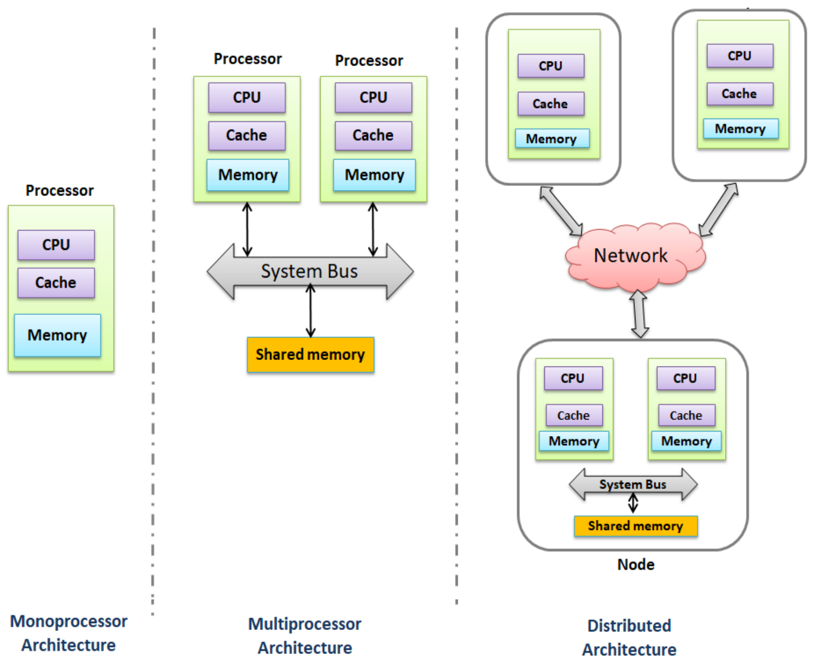


Figure 2.4: The categories of hardware architecture.

On the other hand, a processor itself can be manufactured using different technologies. Thus, it can be a single-core processor [100] or a multi-core processor [21], as presented in the table 2.1:

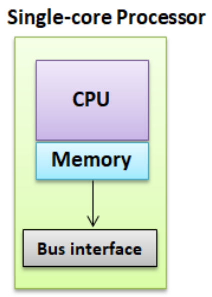
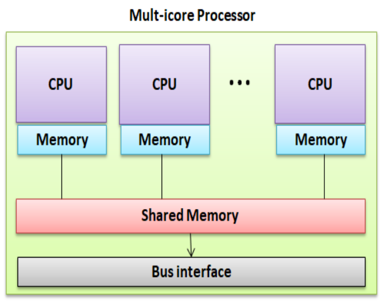
2.2.3.2 Software Architecture

The software architecture incorporate two main parts as shown in figure2.3:

- Real-time operating system(RTOS): it is an operating system that is used in real-time applications to obtain real-time output without buffer delay. It consists of a real-time kernel, which is the vital and central component of a RTOS. The kernel is responsible for: task management, task scheduling, task Synchronization, memory management, etc.

- The application program: this software is devoted to execute various functions to perform a specific treatment. The application program is divided into a set of tasks, each of which ensures a sequence of instructions to respond to system needs..

Table 2.1: Processor architectures

Parameter	**Single-core processor**	**Multi-core processor**
Number of cores	It has just one core inside(see figure below. 	It has two or more cores embedded into one integrated circuit called die (see figure below).  Mult-icore Processor CPU CPU CPU Memory Memory Memory Shared Memory Bus interface
Performance	Single Core performance is important in older applications that are not programmed efficiently to use multiple cores.	Multicore processing can increase performance by running multiple applications concurrently.
Memory	In single core processor memory is utilized by itself.	In multi-core processor memory is shared.
Multitasking	Multitasking is not allowed, as a single-core processor can execute a single instruction at a time.	Multitasking is allowed, as a multi-core processor can execute multiple instructions by using multiple cores, i.e., run two or more processes at the same time.
Reliability	The software is assigned to a single core, thus it is unable to resist faults.	The software is always assigned to different cores. When one piece of software fails, the others remain unaffected.

First, this thesis deals with mono-core processors. Then its contribution is extended to support multi-core architecture.

2.3 Real-time Scheduling: Context and Analysis

A real-time application is composed of a set of software tasks executed by processors to achieve specific treatments. Therefore, it is necessary to organize the execution of tasks over time and manage their competition on the processor. This process is called real-time scheduling which is performed by a scheduler [94, 70]. In fact, according to the performance criteria, the scheduler orders for every time interval the execution of tasks on processor.
In order to perform real-time scheduling analysis, it is important to construct an analytical model describing the timing behavior of the analyzed system.

2.3.1 Real-time Tasks: Model and Classification

The active entities of a real-time system are their tasks. In fact, a real-time task is a computation code, i.e. sequence of given operations, that has to be executed by the processor in order to perform one or many system functions. Each task consist of an infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate.

At a given moment, a task can be in different states as shown in the following diagram presented in figure 2.5:

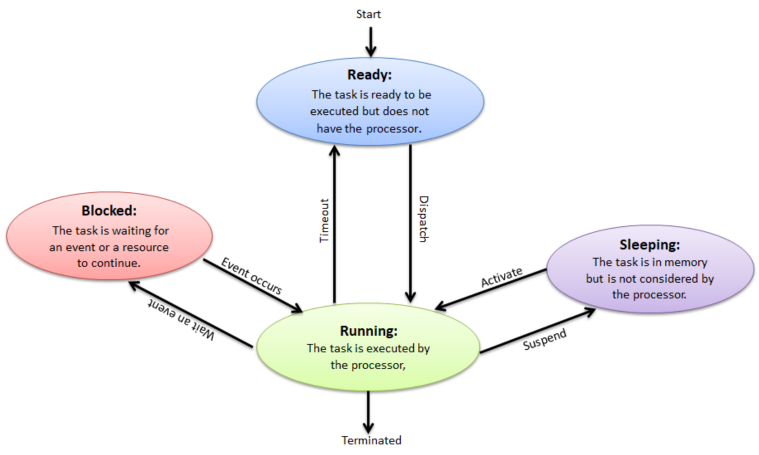

Start
Ready:
The task is ready to be executed but does not have the processor.
Timeout
Dispatch
Blocked:
The task is waiting for an event or a resource to continue.
Event occurs
Wait an event
Sleeping:
The task is in memory but is not considered by the processor.
Activate
Suspend
Running:
The task is executed by the processor,
Terminated

Figure 2.5: Different states of a real-time task.

2.3.1.1 Tasks Model

In the literature, a task τ_i is often represented by a time-line diagram, as shown in figure 2.6, that contains a part of most used properties [63]. Each task τ_i is assumed to generate one or more identical instances which are called jobs denoted $\tau_i j$. :

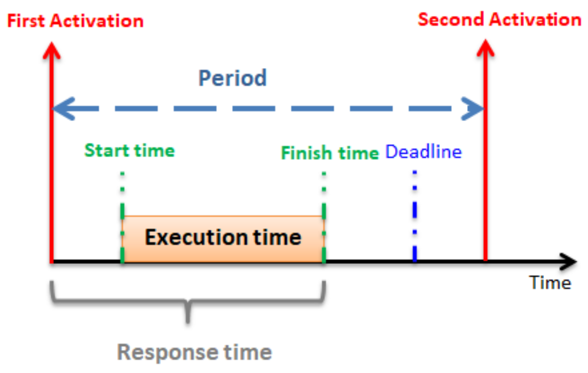


Figure 2.6: Real-time tasks characterization.

- Release time R_i: it corresponds to the activation date of task τ_i after its creation, i.e., the activation instant of the first job of task τ_i. A task is concrete if its release time is known. Otherwise, the task is non-concrete.
- Start time S_i: The date when the task τ_i starts its executing on the processor,
- Finish time F_i: The date when the task τ_i finishes its execution.
- Execution time C_i: The time needed for executing task τ_i on a processor. In the majority of real-time scheduling works, this parameter is considered as the worst-case execution time (WCET) which is the upper bound of all possible execution times of any job.
- Response time RT_i: duration spent by task τ_i to produce its results.
- Deadline D_i: The time at which task τ_i must finish its execution,
- Period P_i: The execution frequency of task τ_i.

2.3.1.2 Real-time Tasks Classification

As mentioned previously, a real-time system must react to events from the controlled environment while executing specific tasks that can be classified according to various parameters: periodicity, dependency, synchronization and activation as presented in figure 2.7.

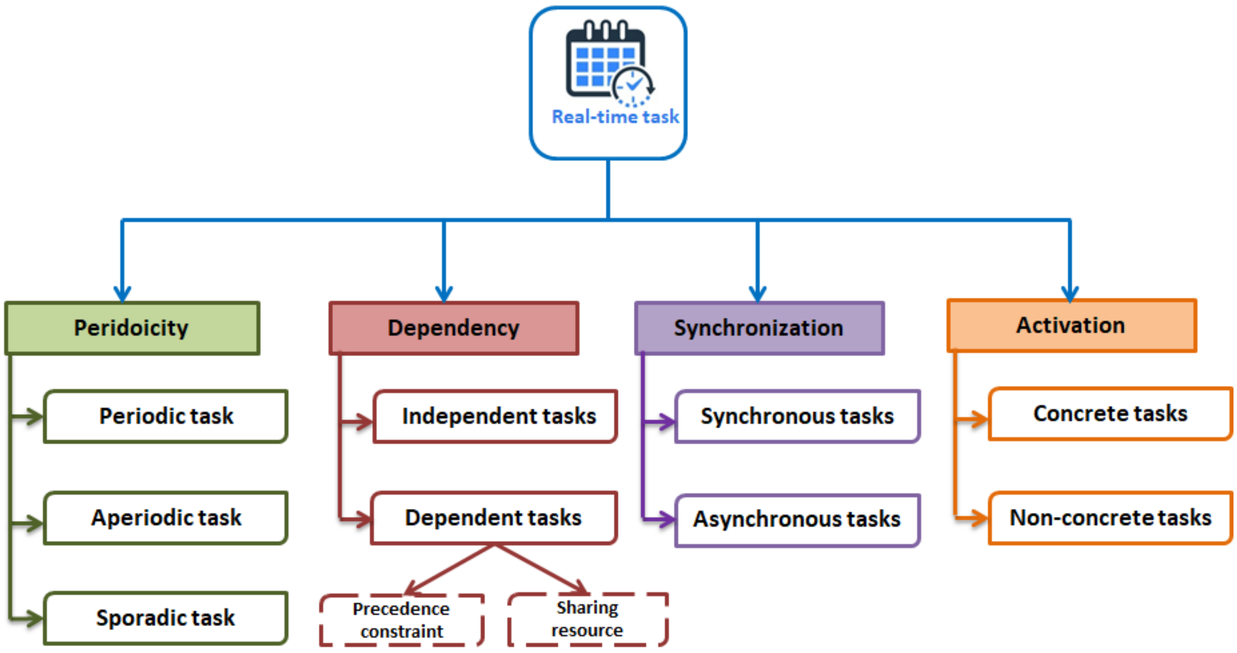


Figure 2.7: Real-time Tasks Classification.

- **Periodicity:**
  Real-time tasks are classified according to their trigger mode, then a task can be:
  - Periodic task: It is activated on a regular cycle, where its instances are separated by a fixed time interval (period) [52].
  - Aperiodic task: It is activated randomly to cope with external events. Its arrival time and the minimum separation duration between two consecutive instances are unknown at design time [41].
  - Sporadic task: It arrives to the system at arbitrary points in time where consecutive jobs are separated by a minimum inter-arrival time [38].

    In this thesis, we use the following notation:
    - emph$\tau_i^0$is the notation of a periodic task, and it can be written as a 4-tuple $(R_i^0, C_i^0, P_i^0, D_i^0)$.
    - emphτ_e^1 is the notation of a sporadic task, and it can be written as a 4-tuple $(R_e^1, C_e^1, P_e^1, D_e^1)$.
    - emphτ_l^2 is the notation of an aperiodic task, and it can be written as a 3-tuple $(C_l^2,\ D_l^2)$.
- **Dependency:**
  In a real-time system, if tasks need to work together to perform a specific functionality, then they are dependent tasks, else they are independent:
  - Independent task set: All tasks are independent from each other, in such a way they can be executed in any order [22].
  - Dependent task set: Dependency between two tasks can be of two types:
    - Precedence dependency: A task need the result provided by another one before its execution [73]. Generally, the precedence dependency of a task-set is represented by a directed acyclic graph DAG where a task τ_i precedes τ_k is represented by $\tau_i \prec \tau_k$, and task τ_i is a predecessor of task τ_k (i.e., τ_k is a successor of task τ_i). In this case, τ_k cannot be executed before τ_j is finished. In a DAG, a task is represented as a node and a communication message among two tasks is represented as an edge [34].
    - Shared resource dependency: When two or more tasks need to advance their executions sharing the same software or hardware resource by using the mutual exclusion algorithm, i.e., this resource cannot be used by more than one task simultaneously [43]:
      - Deadlock: two or more tasks are waiting for each other to be completed, but neither ever does because each task locks the resources required by the other task [77]. Example: let us consider two tasks τ_1 and τ_2 with priority from low to high are 1 and 2. These tasks are completely preemptive, and share two resources R_1 and R_2. As shown in figure 2.8, we have:

1. At instant t_0, τ_1 runs and locks R_1.
2. At instant t_1, τ_2 preempts τ_1 and locks R_2, and then attempts to lock R_1. Since R_1 is already locked, τ_2 enters the waiting state.
3. At instant t_2, τ_1 resumes execution and attempts to lock R_2. Since R_2 is already locked, τ_1 enters the waiting state.Thus, τ_1 and τ_2 can never continue to execute.

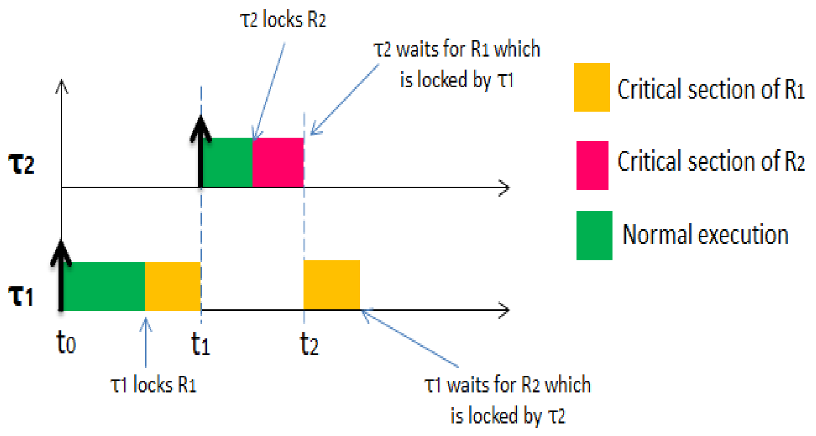


Figure 2.8: Example of deadlock.

· Priority inversion: a low priority task preempts a high priority task for an unbounded time when the shared resource is locked by the low priority task [5, 76]. Example: let us consider three tasks τ_1, τ_2, and τ_3, with priority from low to high are 1, 2, 3. These tasks are completely preemptive, and tasks τ_1 and τ_3 share resource a R. As shown in figure 2.9, we have:

1. At instant t_0, τ_1 runs and locks resource R.
2. At instantt_1, τ_3 is activated to preempt task τ_1 and attempts to lock resource R.As R is locked by τ_1, τ_3 enters the waiting state and τ_1 recovers from the preempted instant.
3. At instant instant t_2, τ_2 is activated and preempts τ_1 as τ_2 is have a higer priority than τ_1, and τ_2 executes successfully. We observe that τ_2 has a lower priority than τ_3 but actually preempts τ_3. This is the priority inversion.
4. At instant instant t_3, τ_1 resumes execution and releases R, and it completes.
5. At instant instant t_4, τ_3 resumes and locks resource R and completes execution.

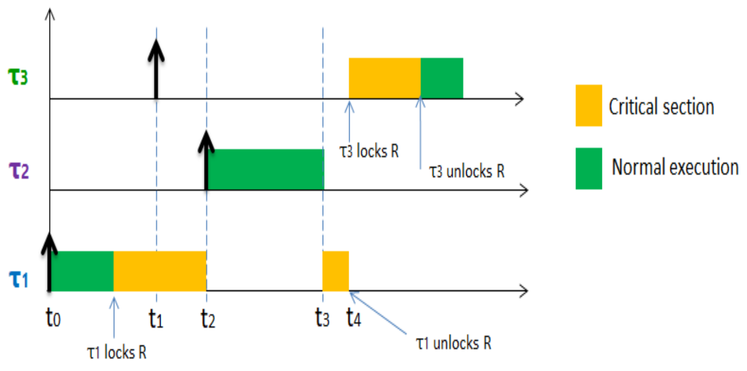


Figure 2.9: Example of priority inversion.

To avoid these problems, several synchronization protocols have been proposed that will be studied later in this chapter.

At different stages in this thesis, we consider independent tasks, dependent tasks by resource sharing, and dependent tasks by precedence constraint.

- **Synchronization:**
  Synchronous tasks are activated at the same time, contrary to the asynchronous ones, where each task has its own release time.

- **Activation:**
  If the system is event-driven, the first activation dates of tasks are unknown, then tasks are considered non-concrete. Otherwise, if the system is time-driven, an activation scenario is imposed, then tasks are considered concrete.

In this thesis, we assume that all tasks are concrete and synchronous.

2.3.2 Real-time Scheduling and Analysis

The real-time scheduling refers essentially to the process of deciding how to order the execution of tasks by the processor [1]. The real-time scheduling theory is classically based on a set of rules that determines the order in which tasks are executed, called an algorithm[16].

Based on various criteria, scheduling algorithms can be classified as presented in table 2.2.

Table 2.2: Scheduling algorithms classification

Preemptive	**Non-preemptive**
The running task can be interrupted at any time to assign the processor to another high priority task ready for execution.	Once started, a task is executed by the processor until the end of its execution. In other words, interruption by another task is allowed and all scheduling decisions must be made after the task has finished executing.
Static	**Dynamic**
Static priority algorithms are those in which priorities are considered fixed parameters, and which are assigned to tasks before they are activated.	Dynamic priority algorithms are those in which the priorities are calculated during the execution time of the system.
Off-line	**Online**
A scheduling algorithm is off-line if it assumes prior knowledge about arrival times, execution times, and deadlines to construct the sequence on the entire task set before the system starts operation.	A scheduling algorithm is online if scheduling decisions are is done at run-time based on the information about the tasks arrived so far.
Heurestic	**Optimal**
An algorithm is heuristic if it solves the feasibility problem, giving a good enough scheduling, rather than finding the best possible one.	A scheduling algorithm is optimal with respect to a system if it is able to find a feasible schedule, if one exists.

The main aim of scheduling analysis is to provide conditions and tests that help designers to determine if the temporal correctness of a real-time system is valid or not prior to run-time. We define below some properties needed to understand of the schedulability analysis:

A schedule consists of a list of times at which a set of ready jobs are intended to take place in the processor.

A scheduler is the entity that is responsible for making a particular schedule[96].

Feasible schedule if every job completes by its deadline while satisfying all other constraints like resource constraints, precedence constraints, etc.

Feasible task set: a task set is considered feasible if exist an algorithml that generates a feasible schedule.

Schedulable task set: a task set is considered schedulable with an algorithm A if A generates a feasible schedule.

Feasibility analysis: determines whether there exists a feasible schedule of the system where all jobs meet their deadlines irrespective to the used scheduling algorithm.

Schedulability analysis: determines whether a given scheduling algorithm will always meet all jobs deadlines.

A feasibility analysis is more general than a schedulability one. As presentend in figure, a task set at is schedulable according to a specific algorithm, is necessarily feasible.

The worst-case scenario of task τ_i: is the configuration of the system to lead to the Worst Case Response Time of τ_i denoted $WCRT_i$. A schedulability test can be obtained by checking if the WCRT of each task is lower than its relative deadline.

Processor demand: over an interval $[t_1, t_2]$, the processor demand is defined as the total time needed for completing all jobs having arrival times at or after time-instant t_1, and deadlines at or before time-instant t_2.

Task processor utilization: the processor utilization of task τ_i denoted U_i is the ratio of time spent in the execution of τ_i's jobs. It can be obtained as follows:

$$U_i = \frac{C_i}{P_i} \tag{2.1}$$

System processor utilization: denoted U is the ratio of time spent in the execution of the whole task set. It is the sum of all tasks of processor utilization. It can be obtained as follows:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{2.2}$$

Hyper-Period (HP): the hyper-period of a task set is the smallest interval of time after which the global periodic jobs of all the tasks are repeated [83]. It is defined as the Least Common Multiple (LCM) of the periods of all the tasks of the system.

2.3.2.1 Priority-based scheduling algorithms

Many scheduling algorithms are priority-based, they are used to define a priority assignment strategy, i.e., define an order of ready tasks. Thus, a task with the highest priority will be selected by the scheduler to be executed.

We present below the priority-based scheduling algorithms most often encountered in the literature.

Static priority algorithms:

- Rate Monotonic(RM): This algorithm was introduced by Liu and Layland in 1973 [64]. It is a preemptive scheduling algorithm that applies to periodic independent tasks. The priority of a task is proportional to its period, i.e. the smaller the period of a task, the higher its priority. This algorithm is optimal in the class of algorithms with fixed priorities for independent preemptive tasks. A sufficient condition for the schedulability of the algorithm RM for a set of n periodic tasks is given by:

$$U \leq n(2^{\frac{1}{n}} - 1) \tag{2.3}$$

  where U is the processor utilization.

- Deadline Monotonic(DM): Although RM is optimal in the class of periodic tasks with deadlines are equal to periods, this is no longer the case when the deadlines are less than the periods. To resolve this issue, an algorithm named Deadline Monotonic (DM) was proposed by Leung and Whitehead in [56]. Indeed, the priority of a task is inversely proportional to its deadline, i.e., the smaller the deadline of a task, the higher its priority. This algorithm is optimal in the class of preemptive algorithms with fixed priorities for independent preemptive tasks with deadlines are less than the periods. A sufficient condition for the schedulability of the algorithm DM for a set of n periodic tasks is given by:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1) \tag{2.4}$$

Dynamic priority algorithms:

- Earliest Deadline Firs (EDF): it is a dynamic priority scheduling algorithm for real time systems [64]. Earliest deadline first selects a task according to its deadline such that a task with earliest deadline has higher priority than others,i.e., priority of a task is inversely proportional to its absolute deadline. Since absolute deadline of a task depends on the current instant of time so every instant is a scheduling event in EDF as deadline of task changes with time,i.e., priorities are reviewed, if necessary, over time. A sufficient condition for the schedulability of the algorithm EDF for a set of n periodic tasks is given by:

$$U \leq 1 \tag{2.5}$$

- Least-Laxity First (LLF): This algorithm selects the task with lowest laxity to be executed [9]. A sufficient condition for the schedulability of the algorithm LLF for a set of n periodic tasks is given by:

$$U \leq 1 \tag{2.6}$$

In our work, we have chosen the EDF algorithm as the scheduling algorithm, view of its optimality and performance.

2.3.2.2 Migration-based algorithms

In general, multi-core scheduling is more complex than mono-core scheduling, as shown in figure 2.10. In fact, in mono-core scheduling, it is enough to satisfy the temporal problem, i.e. when to start, interrupt and resume each task. However, in addition to temporal issue, the multicore scheduling has to solve the spatial one, i.e., on which core a task will be executed.

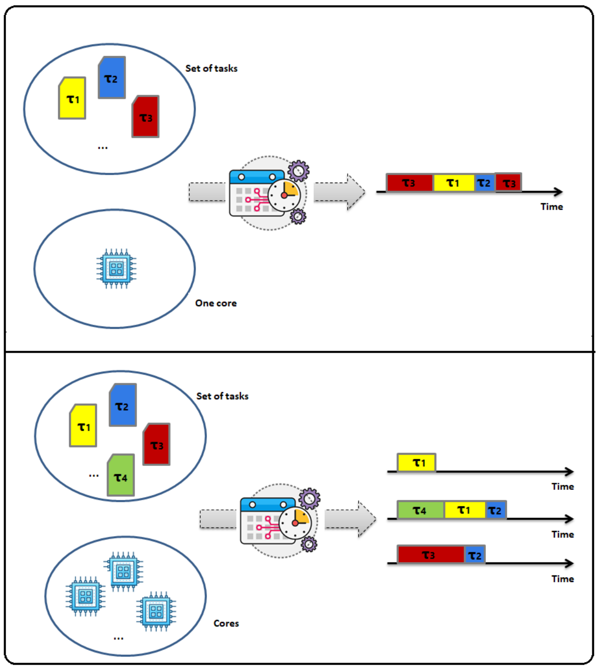


Figure 2.10: mono-core and multicore scheduling

In the case of multicore systems, the scheduling algorithms are classified according to the allocation issue. The authors in [19] proposed the following classification:

- No migration: This approach considers that tasks are not allowed to migrate from one core to another. In fact, it statically partitions tasks into disjoint subsets, each of which is handled on a specific core by a scheduling policy that can be different from the others.

- Restricted Migration: This approach allows to different task jobs to run on different core with the only constraint, that each job only runs on one CPU.
- Full migration: This approach allows to each job to migrate and run on a different core, but without any parallel execution of the job.

In this thesis, we do not allow task migration and each core receives a subset statically.

2.3.3 Synchronization Protocols

As mentioned previously, the concurrent accesses of tasks to shared resources may cause some problems like the deadlock and the priority inversion. In order to avoid these issues, several synchronization protocols have been proposed [80]. The following protocols are widely used:

- The Priority Ceiling Protocol (PCP)[81]: is a method for eliminating deadlock issue. The basic idea of this protocol is to assign a priority ceiling to each shared resource. This priority is equal to the highest priority of any task that may lock the resource. Thus, a task can only take a resource if this one is free and if its priority is greater than the ceilings of all resources currently taken.
- The Priority Inheritance Protocol (PIP)[86]: is a protocol for eliminating the priority inversion. Thus, if a job blocks one or more high-priority jobs, it would temporarily inherit the priority of the blocked task (the one with higher priority) to prevent the execution of intermediate priority tasks while the blocking tasks owns the critical section. Then, the priority of the blocking tasks returns to its nominal value when it left the critical section.
- StackResourcePolicy(SRP) [6]:similar to PCP, this protocol uses the concept of priority ceiling. The SRP protocol allows a task to start executing only when its own preemption level is higher than the one of the executing task and also higher than the ceilings of all the resources.

2.3.4 Real-time Scheduling of Mixed Task Sets

A real-time system often needs to execute both periodic and aperiodic tasks to achieve its functionalities [30]. The scheduling problem for mixed task set (aperiodic and periodic tasks) is more difficult than the scheduling problem for periodic tasks [53]. In fact, scheduling algorithms for mixed task set must be able to provide good average response times for soft aperiodic tasks without compromising the hard deadlines of the periodic ones. Thus, it is a primordial challenge to work out a solution that allow to execute aperiodic tasks with a fast average response times while guaranteeing the hard deadlines of periodic tasks.
A simple way to accomplish this aim is the background processing [55]. In fact, background execution serves aperiodic tasks as lowest-priority tasks [92]. In other words, aperiodic tasks can be scheduled and executed only whenever the processor is

idle, i.e. if there are no pending periodic or sporadic tasks. Figure 2.11 illustrates an example of background execution for tow periodic tasks τ_1(period=4, WCET=2), τ_2 (period=10, WCET=2) and an aperiodic task Ap_1 (WCET=1). We assume that the periodic tasks are scheduled according to the RM algorithm, yielding a higher priority for τ_1.

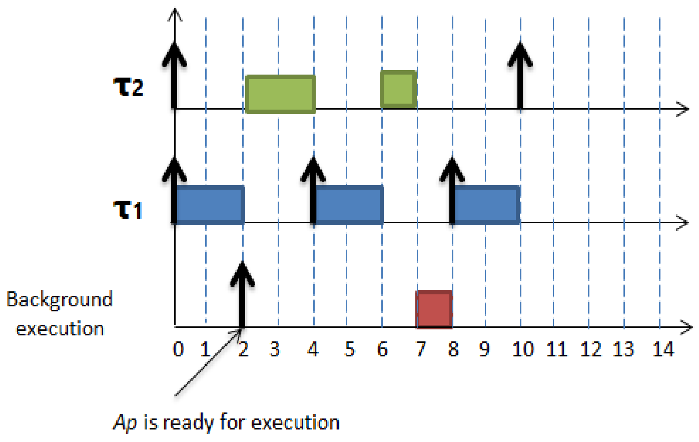


Figure 2.11: Illustration of the background processing

Since background execution occurs when the processor is idle, we observe that the aperiodic task cannot begin until time = 7. Thus, the response time of the aperiodic task is long (equal to 6 units), even it needs only 1 unit of time to be completed.

To overcome these limitations, several important approaches are proposed in the literature. The basic idea is to serve aperiodic tasks by periodically invoked servers, which are accounted for in periodic task schedulability analysis [53]. A periodic server is characterized by a period P_s and a capacity C_s (budget), and it serves aperiodic tasks until budget expires. As shown in figure 2.12, aperiodic tasks are put into a queue for the periodic server, then the server takes tasks from its queue for execution while using its available capacity. The server is scheduled as any periodic task. We will discuss some aperiodic server scheduling algorithms in what follows.

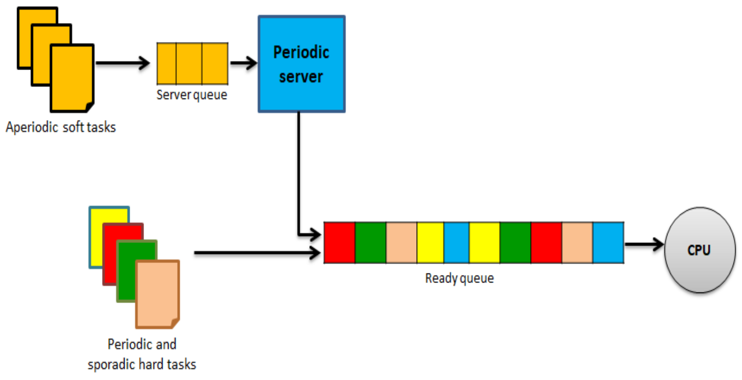


Figure 2.12: Illustration of the periodic server

2.3.4.1 Polling Server

This server is used under fixed priority assignments, it consists of creating a periodic task for servicing any pending aperiodic tasks [54]. At regular intervals equal to its period P_s, the polling server is instantiated, then it examines the aperiodic tasks queue to execute the pending aperiodic requests for up to C_s time units when it has the highest current priority. However, if the aperiodic queue is empty, the polling server suspends itself until its next period P_s, and gives up its budget. The server capacity is replenished every period. The major disadvantage of this server lies in its limited performance if an aperiodic task arrives just after the server has suspended. In this case, the aperiodic task has to wait until the next period, which causes a long response time. Figure 2.13 presents an example of polling server execution for a periodic task τ_1(period=4, WCET=2) and a list of aperiodic requests. We suppose that all aperiodic tasks have execution time equal to 1, and the periodic server parameters are: $P_s = 5$ and $C_s = 2$.

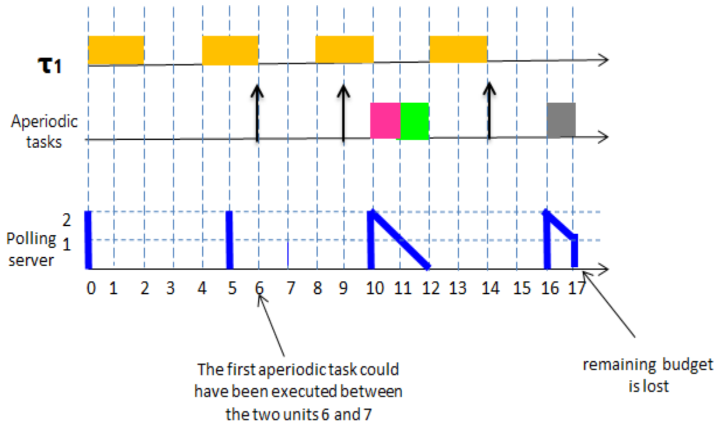


Figure 2.13: Illustration of a polling server execution

The interference by a server task is the same as the one introduced by an equivalent periodic task in rate monotonic fixed priority scheduling. Thus, a sufficient condition for the schedulability of n periodic tasks and a server task is given by:

$$\frac{C_s}{P_s} + \sum_{i=1}^{n} \frac{C_i}{P_i} \leq (n+1)(2^{\frac{1}{n+1}} - 1) \tag{2.7}$$

Where $\frac{C_s}{P_s}$ is equal to the server utilization factor denoted by U_s.

2.3.4.2 Deferrable server

Unlike PS, the deferrable server DS Keeps the balance of the budget until the end of the period [97]. In other words, aperiodic tasks can be serviced at the server's high priority at anytime as long as the server's capacity for the current period has not been consumed. Thus, the DF can improve average response times for aperiodic tasks. Figure 2.14 shows the illustration of the deferrable server execution of the previous example (done with polling server).

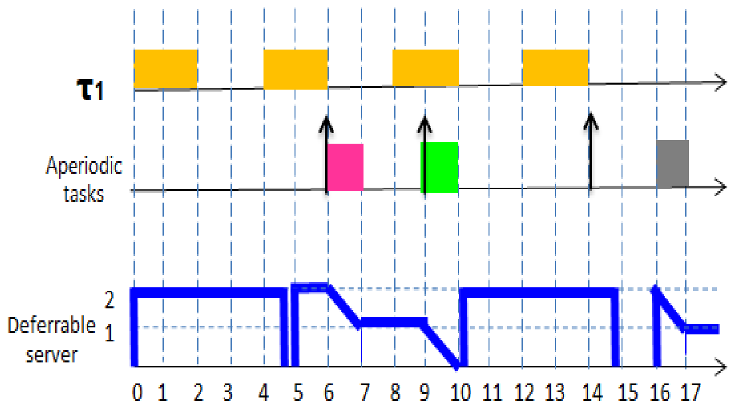


Figure 2.14: Illustration of a deferrable server execution

A sufficient condition for the schedulability of the algorithm RM for a set of n periodic tasks and a server task is given by:

$$U \leq \ln \frac{U_s + 2}{2U_s + 1} \tag{2.8}$$

2.3.4.3 Total Bandwidth Server

The total bandwidth server TBS is a dynamic priority server used with EDF[17]. The main aim of this server is to execute the aperiodic while achieving good aperiodic responsiveness. When an aperiodic task arrives at time $t = r_k$, it receives a deadline d_k. Once a deadline is assigned, the aperiodic task is inserted in the ready queue of the processor and scheduled together with the periodic hard tasks. The main disadvantage of TBS is that after starting executiion, a task takes more time than the one declared, which can cause the violation of hard deadlines for periodic tasks.

$$d_k = max(r_k, d_{k-1}) + \frac{C_k}{U_s} \tag{2.9}$$

where C_k is the execution time of the aperiodic task and U_s is the server utilization factor. By definition, $d_0 = 0$.

A sufficient condition for the schedulability of the algorithm EDF for a set of n periodic tasks and a server task is given by:

$$\frac{C_s}{P_s} + \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{2.10}$$

2.4 Energy Consumption of Real-time Systems

To perform its tasks correctly, a real-time system needs to consume the adequate amount of energy for its execution [35, 36, 46]. Hence, monitoring energy consumption is of critical importance in real-time systems [110, 82]. Exists environmental energy sources that can be exploited to supply real-time systems with the necessary energy [113, 10], such energy is defined as energy generated from natural resources like sunlight, wind, tides, geothermal heat, etc., that are naturally replenished. As a consequence, energy harvesting technology is required [78, 33]. In fact, this technology consists of converting energy from the environment into electrical energy and storing it in an energy storage unit[8]. Such an energy storage unit is needed since a real-time system have to operate continuously without missing the available energy [24, 26]. However, using renewable energy sources to power these systems makes it more difficult to schedule real-time tasks and can lead to energy starvation due to its availability that typically varies with time, depending on environmental conditions. Thus, the energy consumption constraint must be considered and added to temporal constraints where scheduling tasks in real-time systems [102]. Therefore, the violation of one of them will conduct to system failure.
In this case, a job misses its deadline when:

- It reaches its deadline at time t, its execution is incomplete because the time needed to complete the job by its deadline is insufficient.

- It reaches its deadline at time t, its execution is incomplete because the energy necessary to process the job by its deadline is not available.

Usually, energy-harvesting technology is composed of three main parts, as described in figure 2.15,

- The energy harvester is the part in charge of collecting the ambient energy and converting it into electrical energy.

- The energy storage unit, such as rechargeable batteries or super-capacitors, is a device used to store the harvested energy.
- The computing unit is the energy consumer which can be an embedded application or a mission part of the system, it is typically composed of data sensors, a processing unit and a data transmission device.
  In this thesis, we assume that the computing unit represents the execution support of the real-time tasks.

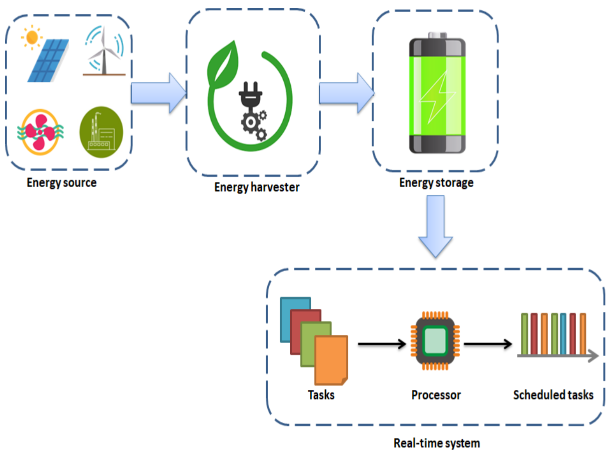


Figure 2.15: Energy-harvesting components

2.4.1 Environmental Energy Sources

The main aim of this subsection is to present some environmental energy sources [44].

Solar energy: it is simply the light and heat that come from the sun. The power is collected using photovoltaic cells that transform sunlight into electricity and can supply different devices.

Wind energy: it describes the process that used to generate electricity from the wind. The power is collected using wind turbines that convert the kinetic energy in the wind into electrical energy.

Hydroelectric energy: it works similarly to wind power in that it is used to spin a generator's turbine blades to create electricity. This energy is based on fast moving water in rivers to spin the turbine blades.

Biomass energy: it uses organic material from animals and plants, including trees, waste wood, and corps. This biomass is used to create heat, that powers a steam turbine and generates electricity.

Geothermal energy: it is the heat that comes from the sub-surface of the earth. This energy source can be made greener by pumping the steam and hot water back into the earth, thereby lowering emissions.

In this thesis, we work on real-time reconfigurable systems powered by renewable energy harvested from the environment.

2.4.2 Energy Storage Devices

As mentioned previously, the production of renewable energies strongly depends on the variable environment conditions (sunlight, wind speed, water availabilit, etc.)[20]. Thus, it seems crucial to store the harvested energy for future use. In fact, there are two widely used alternatives for energy storage [88] Rechargeable Batteries and supercapacitors.

Batteries :
They provide a way to power electric devices that cannot be plugged into a mains supply at all times. Batteries are classified into two categories based on their ability to be reused: non-rechargeable (primary) and rechargeable (secondary) batteries. The main difference between them is that rechargeable batteries can be charged and reused again after have been fully discharged, while non-rechargeable batteries cannot be charged again once they discharge fully.

Supercapacitors :
They are high-capacity capacitors with a capacitance value much higher than other capacitors, but with lower voltage limits, that bridges the gap between electrolytic capacitors and rechargeable batteries. It typically stores 10 to 100 times more energy per unit volume or mass than electrolytic capacitors, can accept and deliver charge much faster than batteries, and tolerates many more charge and discharge cycles than rechargeable batteries.

2.4.3 Approaches for Minimizing Energy Consumption

Energy consumption is a key challenge in the design of real-time systems[101]. Therefore, there are two popular techniques are applied to reduce energy consumption of a real-time system presented in figure2.16 Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVFS).

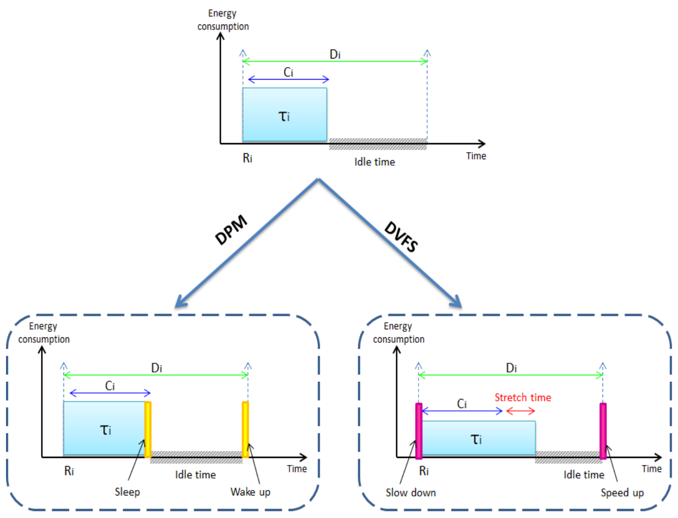


Figure 2.16: Minimizing Energy Consumption with DPM and DVFS.

Dynamic power management (DPM) is a control strategy aimed to dynamically adapt the operating mode of a real-time system to its workload in order to minimize the power consumption under given performance constraints[14]. In other words, this technique serves to switch from idle mode to active mode and vice versa according to system activity. Therefore, the processor is temporarily turned off whenever necessary. For example, at a given instant not all tasks of a real-time system participate in performing different functions, so it is useful to switch the unused tasks to the sleep state to reduce power consumption. However, this technique incurs performance loss because of the overhead associated with shutdowns and wake-ups. An effective DPM policy must therefore seek to maximize power savings while keeping performance degradation within acceptable limits [62].

Dynamic Voltage and Frequency Scaling(DVFS) serves to dynamically change the frequency of the processor when necessary to manage energy consumption[98]. In fact, energy consumption is a quadratic function of frequency, thus the reduction in frequency value has a significant impact on minimizing energy consumption. This technique should be applied to processors that allow varying the supply voltage and thus the operating frequency. However, using this technique may violate real-time constraint. In fact, the execution time of a task is equal to its number of execution cycles divided by the operating frequency, thus if the frequency is reduced, the running job will take longer time to be executed and may violate its deadline. For example, if the frequency is reduced to half, the task will take twice as long to execute. An effective DVFS policy must therefore save energy while controlling the expense of stretching the execution times, so that the temporal constraint can still be met [3].

In this thesis, we apply the frequency variation technique when necessary to manage the energy consumption.

2.5 Reconfigurable Real-time Systems

Real-time software systems need to modify its behavior or its architecture according to the evolution of the requirements of its context of use and the variation of the constraints of its execution environment, hence the notion of reconfigurable real-time system. With the increasing complexity of these systems and the autonomy required to manage them, the reconfigurability is more important than ever.

2.5.1 Reconfigurability Definition

Real-time systems are highly coupled to the external world and are increasingly facing changing environmental conditions. That is, during its lifetime, a real-time system should be adapted to changing environmental conditions and parameters such as available battery power, varying communication bandwidth, available memory or faults in software components in order to preserve desired application-level quality of service while continuously meeting all tasks deadlines. The reconfiguration technology presents a powerful solution to adapt a real-time system to face environmental changes and to perform autonomous behavior. Reconfigurability means the capability to adapt the system behavior to the evolution of its environment by applying a reconfiguration scenario by adding, removing or updating hardware or software components [111]. In other words, reconfiguration is a runtime flexible scenario that adapts the current system's implementation to any related environment evolution under well-defined conditions. More precisely, at a given moment and under well-defined environmental conditions, only a subset of the system's tasks will be executed to achieve the required functionalities.

2.5.2 Type of Reconfigurations

In literature, two policies are defined for the reconfiguration:

- Static vs Dynamic: Static reconfiguration is applied off-line to configure the system before starting its execution. However, dynamic reconfiguration is applied on-line when the system is under operation.
- Manual vs Automatic: Manual reconfiguration is needs to be done manually by user to achieve the system functionality. Automatic reconfiguration is applied by automatic program or by intelligent agents without manual intervention.
- Hardware vs Software: A hardware reconfiguration is assumed to be any addition and/or removal of hardware components required to ensure the exibility of the system. The software reconfiguration is an execution scenario that allows moving from one implementation to another by adding/removing/updating software tasks.

2.6 Analysis of Related Works and Discussion

The related and relevant research works are presented in this section. These works related to our contributions range from static analysis techniques to energy-aware scheduling of multi-core reconfigurable real-time systems with a mixed set of dependent software tasks, and are further detailed as follows.

Several studies have been done in recent years focusing on real-time systems such as those reported in [7, 99, 105, 37, 49]. They neither deal with energy and dependency constraints nor with multi-core architectures. Some of them reported in [7] aim to minimize deadlines on a mono-core processor without considering energy and dependency constraints. Moreover the rate of deadlines reduction can be improved compared to the one delivered in the work presented in this thesis. On other hand, in [28], the authors consider multicore architecture where the tasks are independent and their deadlines are known.

Task scheduling process is one of the most important objectives for the designing of the real-time systems as it determines the appropriate and optimal resource utilization and overall quality of the system [65, 57]. In [51] the authors work on scheduling independent task with hard real-time constraints and multicore systems, where the processors can be manipulated to change the clock cycle speed and power levels. Zhou et al. [114] considered heterogeneous multiprocessors systems with deadline constraints and reliability. In fact, the authors tried to develop a earliest finish-time based algorithm for heterogeneous multiprocessor systems to maximize reliability. However, it didn't take energy consumption management into account. Some research contributions have been dedicated to working on reconfigurable real-time systems in various areas [12, 11, 104]. In [12], the authors proposed a methodology to design safe reconfigurable medical robotic systems without considering energy and dependency constraints.

The energy-aware task scheduling is of major importance for the energy-aware real-time systems. One of the main approaches here is to change the processor's scheduling and DVFS frequency in order to ensure energy requirements. Abdel-Basset et al. [2] proposed an approach to reduce the energy consumption with DVFS technique while satisfying memory capacity constraints. The work reported in [79] deals with the minimization of energy consumption on heterogeneous processors. In fact, it focuses only on energy constraint while calculating the desired minimum energy. Furthermore, the work reported in [103] considers mixed time criticality levels for different energy criticality modes, i.e. the energy-constraint may surpass the temporal constraint when this latter is considered less critical. However, this approach may cause a catastrophic event in hard real-time systems as they operate within the confines of a stringent deadline. In addition, the studies in [108, 25, 42, 91, 112] considered energy-awre scheduling without considering temporal constraints analysis. In [108, 25, 24, 32] the authors only seek to schedule tasks to respect energy constraints since deadlines are given beforehand. In these researches, the input is a defined system with all parameters, and in the output, they verify if the tasks can be scheduled with the given deadlines for the purpose of energy consumption or not.

On the other hand, there has been a large body of works trying to optimize energy as well as guarantee dependency constraints. The works presented on [23] and [29] investigated the problem of minimizing energy consumption for task scheduling on heterogeneous multiprocessor systems while meeting the precedence constraints of these tasks. These researches have received extensive attention in the literature. However, they considered neither the hard time constraint of tasks nor their types. Thus, the proposed solutions can affect the reliability of a real-time system with hard temporal constraints. Moreover, Huang et al. [47] considered dynamic scheduling of tasks modeled by directed acyclic graphs without considering the time constraint.

Table 2.3 describes the comparison of the work developed in this thesis with previous related work. Even if the research results of each topic are separately rich, to our best knowledge, none of these solutions simultaneously considers real-time, energy, dependency constraints, multi-core architecture, communication messages, mixed tasks set, and reconfiguration property. The originality of this work is that it is the first that computes effective deadlines, ensuring the feasibility of real-time reconfigurable systems while considering:

- The multi-core architecture which deliver higher throughput at energy consumption than mono-core architecture,
- The reconfiguration-aware property to ensure system adaptation to changing environmental conditions,
- The ability to adjust the frequency of the processor to manage energy consumption,
- The characterization of tasks and messages' deadlines which will be certainly respected online,
- The dependency between tasks (by sharing resources or by precedence constraints),
- The energy harvested from the environment to power systems to alleviate the burden of periodic battery replacement.

Table 2.3: Related work overview

Work	Mixed Tasks Set	Deadline Calcutation	Reconfiguration Property	Multi-core Architecture	Energy Constraint	Dependency Constraint	Communication Messages Constraint
[28]	-	-	✓	✓	✓	-	-
[24]	✓	-	-	-	✓	-	-
[7]	-	✓	-	-	-	-	-
[49]	-	✓	✓	✓	✓	-	-
[99]	✓	✓	-	-	-	-	-
[32]	✓	-	-	-	✓	-	-
[114]	-	✓	-	✓	-	✓	✓
[51]	✓	-	-	✓	✓	-	-
[29]	-	-	-	✓	✓	✓	✓
Our thesis	✓	✓	✓	✓	✓	✓	✓

2.7 Conclusion

For a better understanding to the presented works in this dissertation, the state of the art is introduced. We explained the methods that have been used in the literature to solve the problem of real-time tasks scheduling. We have introduced the real-time systems, their different classes and their software and hardware architectures. We talked about the concept of tasks and the related constraints such as real-time, energy and dependency constraints. Furthermore, we presented the problem of real-time scheduling under the different constraints on a reconfigurable architecture. The existing researches and approaches in the literature are studied as complete as possible. The next chapters will detail our contributions, which are dedicated to mono and multi-core architectures.

Chapter 3

Efficient Real-time Scheduling Algorithm for Mixed Task Set on Mono-core Architecture

3.1 Introduction

The previous chapter discussed various generalities about real-time systems related to reconfigurability analysis, type of architecture, energy consumption etc. In addition, it gave an overview of existing methodologies in the literature working on reconfigurable real-time systems. In this chapter, we cope with three main factors, which are: mixed tasks set, hard and soft temporal constraints, and the mono-core architecture. In other words, we propose a new approach that serves to schedule a set of software tasks with various types (periodic, aperiodic, and sporadic) and constraints (soft and hard) on mono-core real-time systems.
The outline of the chapter is organized as follows: First, we present the motivation section. Then, the computational model and the assumptions considered are defined. After that, we explain in detail our contribution. Furthermore, a formal case study is also proposed in order to pinpoint the main problem studied in this chapter.Lastly, the discussion section is presented to analyze and interpret this chapter's findings.
Note that this chapter has been published in the International Conference on Software Technologies[39, 38].

3.2 Motivation

Due to temporal perspective, real-time systems design is inherently different from other forms of systems design. In other words, such design must not only integrate ways to address functional complexity, but also ways to analyze and predict temporal constraints.

Real-time systems are found in diverse application areas, including critical control applications where time-critical control activities should be implemented as hard periodic tasks. In fact, with hard timing constraints, all task instances must be guaranteed to complete within their deadlines to deny any catastrophic consequence on the controlled system. However, hard real-time tasks usually coexist with soft real-time activities. In fact, these applications, need to serve aperiodic user requests while executing soft aperiodic tasks that do not need to be guaranteed, but if they are completed as early as possible, they provide good performance. For example, in an aircraft control system, turning on/off autopilot mode is an aperiodic event that demands the closest attention and should be completed as soon as possible. Therefore, handling soft aperiodic simultaneously with hard periodic and sporadic tasks is an important aspect in real-time scheduling.

Definitely, working on scheduling different types of software real-time tasks simultaneously is more difficult than considering a single type of tasks. Consequently, a designer given an application composed of mixed tasks and constraints has to predict the behavior of a real-time system by ensuring its feasibility while considering three main factors:

- the system architecture.

- the different type of tasks.
- the type of temporal constraints: hard or soft.

Therefore, the problem to be treated in this chapter is how parameterizing feasible scheduling of real-time tasks with various types and constraints in the context of dynamic-priority, preemptive, mono-core scheduling. Thus, the two main goals to be achieved are

- an aperiodic task must be executed as early as possible without jeopardize the schedulability of periodic and sporadic tasks.
- a periodic or sporadic job has to meet its deadline using the EDF scheduling algorithm even in worst-case conditions,i.e., .

In order to solve these issues, we propose a new offline approach that:

- addresses initially the mono-core architecture,
- deals with real-time tasks of various types and constraints simultaneously,
- parameterizes periodic server to execute aperiodic tasks,
- calculates soft deadlines of aperiodic tasks,
- calculates periodic and sporadic tasks hard deadlines which will be certainly respected online,
- improves response times of aperiodic tasks which can lead to a significant improvement of the system performance,
- presents new tool called GIGTHIS-TOOL to evaluate the proposed solution.

3.3 Formalization

In this section, we aim to explicit and formalize a real-time system having hybrid task set. Therefore, we introduce the mathematical expressions that represents the characteristics of each type of task.

3.3.1 System Model

Let us consider that a real-time system denoted Π is defined as having three task sets as presented in figure 5.1:

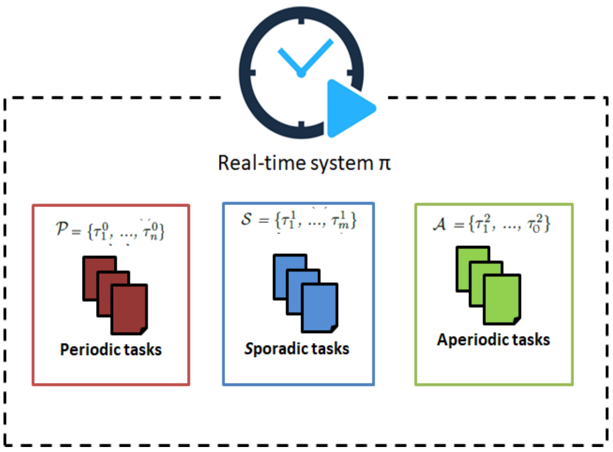

Real-time system π
$\mathcal{P} = \{\tau_1^0, \dots, \tau_n^0\}$
Periodic tasks
$\mathcal{S} = \{\tau_1^1, \dots, \tau_m^1\}$
Sporadic tasks
$\mathcal{A} = \{\tau_1^2, \dots, \tau_o^2\}$
Aperiodic tasks

Figure 3.1: Π's tasks sets.

- Set of periodic tasks: denoted $\mathcal{P}$, containing n periodic software tasks, i.e., $\mathcal{P} = \{\tau_1^0, \dots, \tau_n^0\}$. We suppose that all these tasks are activated at $t = 0$.
- Set of sporadic tasks: denoted $\mathcal{S}$, containing m sporadic software tasks, i.e., $\mathcal{S} = \{\tau_1^1, \dots, \tau_m^1\}$.
- Set of aperiodic tasks: denoted $\mathcal{A}$, containing o aperiodic software tasks, i.e., $\mathcal{A} = \{\tau_1^2, \dots, \tau_o^2\}$.

3.3.2 Task Model

The main aim of this section is to present and describe the parameters and characteristics of each type of task.

- Each periodic task τ_i^0, $i \in [1, \dots, n]$, in $\mathcal{P}$ is a 5-tuple $(R_i^0, C_i^0, P_i^0, D_i^0, Dmax_i^0)$ where:
  - R_i^0 is the release time at which τ_i^0 becomes ready for execution,
  - C_i^0 is the worst-case execution time (WCET),
  - P_i^0 is the period,
  - D_i^0 is the relative deadline to be calculated,
  - $Dmax_i^0$ is the maximum relative deadline.

Each periodic task τ_i^0 produces an infinite sequence of jobs τ_{ij}^0. Each job τ_{ij}^0 is described by:

- a release time r_{ij}^0,
- a relative deadline d_{ij}^0,
- end execution time E_{ij}^0.

- Each sporadic task τ_e^1, $e \in [1, ..., m]$, in $\mathcal{S}$ is a 5-tuple (R_e^1, C_e^1, P_e^1, D_e^1, $Dmax_e^1$) where:
  - R_e^1 is the release time at which τ_i^1 becomes ready for execution,
  - C_e^1 is the worst-case execution time (WCET),
  - P_e^1 is the period which measures the minimum interval between the arrival of two successive instances of a task τ_e^1,
  - D_e^1 is the relative deadline to be calculated,
  - $Dmax_e^1$ is the maximum relative deadline defined by user.

Each sporadic task τ_e^1 produces an infinite sequence of jobs τ_{ef}^1. Each job τ_{ef}^1 is described by:

- a release time r_{ef}^1,
- a relative deadline d_{ef}^1,
- end execution time E_{ef}^1.

- Each aperiodic task τ_l^2, $l \in [1, ..., o]$, in $\mathcal{A}$ is a 2-tuple (C_l^2, D_l^2) where:
  - R_l^2 is the arrival time which is unknown at design time.,
  - C_l^2 is the worst-case execution time (WCET),
  - D_l^2 is the relative soft deadline to be calculated.

  Each aperiodic task τ_l^2 consists of an infinite sequence of identical jobs τ_{lp}^2; however, their activations are not regularly interleaved. Each job τ_{lp}^2 is described by:

  - a release time r_{lp}^2,
  - a relative soft deadline d_{lp}^2,
  - end execution time E_{lp}^2.

In a given interval of time, an aperiodic task can arrive in a completely random way. Thus, we model this number by the Poisson distribution with a parameter λ which is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time. We note by OC the maximum number of aperiodic tasks' occurrences estimated on the hyper-period.

Let us denoted by HP the hyper-period of a set of periodic tasks. It is the least common multiple of periods of all the tasks in that set.

$$HP = LCM\{P_i^0\} \tag{3.1}$$

For example, two periodic tasks τ_1^0 and τ_2^0 having periods $P_1^0 = 4$ and $P_2^0 = 5$ respectively will have a hyper-period, $HP = lcm(P_1^0, P_2^0) = lcm(4, 5) = 20$.

This section starts with an overview of the proposed methodology. Then, it presents in detail the solutions involved in this chapter.

3.4 Feasible Scheduling of Periodic, Aperiodic, and Sporadic Real-time Tasks with Hard and Soft Deadline Constraints:

In this section, we present our contribution that we propose to compute effective deadlines of mixed tasks set having both hard and soft constraints in order to ensure system feasibility.

3.4.1 Overview of the proposed methodology

The fundamental purpose of this contribution is to ensure a feasible scheduling for a set of software real-time tasks having various types (periodic, aperidoci and sporadic) and constraints (hrad and soft) simultaneously.
This methodology is composed of two main steps as shown in figure 3.2:

- First, this approach starts by parameterizing a New Periodic Server NPS server which is a service task, with a period P^s and a capacity C^s. This server is invoked periodically to execute aperiodic tasks. then, this approach calculates soft deadlines of aperiodic tasks while supposing that an aperiodic task, with the smallest WCET, gets the highest priority.
  The NPS can provide a substantial reduction in the average response time of aperiodic tasks.

- Second, The proposed approach move to characterize hard deadlines for periodic and sporadic tasks. It starts by calculating jobs'deadlines. In fact, for each periodic/sporadic task, it calculates the deadlines of its jobs that occur on the hyper-period based on the maximum cumulative execution time requested by:

  – other periodic/sporadic jobs that will occur before the considered periodic/sporadic job on the hyperperiod based on the degree of criticality,

  – aperiodic tasks that may occur before periodic/sporadic job on the hyperperiod.

  Then, for each periodic/sporadic task, its deadline will be equal to the maximum of its jobs' deadlines. Thus, at runtime, even if an aperiodic task occurs, this methodology ensures certainly real-time system feasibility of periodic and sporadic tasks.

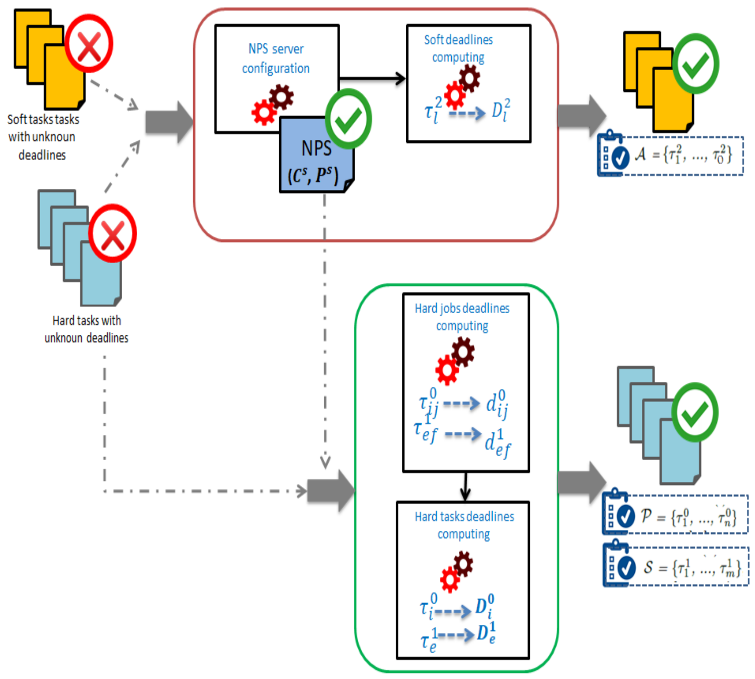

Soft tasks tasks with unknown deadlines
NPS server configuration
NPS (C^s, P^s)
Soft deadlines computing
$\tau_i^2 \dashrightarrow D_i^2$
$\mathcal{A} = \{\tau_1^2, \ldots, \tau_0^2\}$
Hard tasks with unknoun deadlines
Hard jobs deadlines computing
$\tau_{ij}^0 \dashrightarrow d_{ij}^0$
$\tau_{ef}^1 \dashrightarrow d_{ef}^1$
Hard tasks deadlines computing
$\tau_i^0 \dashrightarrow D_i^0$
$\tau_e^1 \dashrightarrow D_e^1$
$\mathcal{P} = \{\tau_1^0, \ldots, \tau_n^0\}$
$\mathcal{S} = \{\tau_1^1, \ldots, \tau_m^1\}$

Figure 3.2: The two main steps for the first contribution.

3.4.2 New Periodic Server for Serving Soft Aperiodic Tasks:

The main aim of this subsection is to present the steps for serving tasks with soft constraints. First we start by parametrize the NPS server, then we move to aperiodic tasks soft deadlines calculation.

3.4.2.1 NPS Server Configuration:

As mentioned previously, aperiodic tasks will be run periodically by the periodic server NPS. This server behaves much like a periodic task with a period P^s, and a capacity C^s. The server can serve aperiodic tasks until its capacity expires; then it can be replenished every period. The server's parameters will be calculated to meet time requirements. **P^s computing:**
P^s is calculated in such a way that the periodic execution of the aperiodic server is repeated as many times as the maximum number of aperiodic tasks occurrences in the hyper-period. As, OC is the maximum number of aperiodic tasks' occurrences estimated on the hyper-period, then, NPS must be activated OC times to serve all possible activations of aperiodic tasks that may occur. Thus, its period is calculated as below:

$$P^s = \lfloor \frac{HP}{OC} \rfloor \tag{3.2}$$

Example 1: let us consider a real-time system having two periodic tasks τ_1^0 and τ_2^0 with $P_1^0 = 3$ and $P_2^0 = 7$ are their periods respectively, and an aperiodic one τ_1^2 which at most will operate 3 times in the hyper-period HP:i.e., $OC = 3$ occurrences. Thus, we have

$$HP = lcm(3, 7) = 21 \tag{3.3}$$

then,

$$P^s = \lfloor \frac{HP}{OC} \rfloor = \lfloor \frac{21}{3} \rfloor = 7 \tag{3.4}$$

C^s computing:

C^s is calculated based on unused processing time by a given set of periodic and sporadic tasks in the hyper-period in such way aperiodic task execution should not jeopardize schedulability of periodic and sporadic tasks.

Moreover, aperiodic tasks are scheduled by utilizing unused processing time by a given set of periodic and spordic tasks in the hyper-period. Thus, the capacity of server is calculated as follows:

- First, we calculate the unused time by subtracting the maximum cumulative execution time requested by periodic and sporadic jobs from HP. Let Q be the maximum cumulative execution time requested by periodic and sporadic jobs on the hyper-period HP.

$$Q = (\sum_{\tau_i^0 \in \mathcal{P}} (C_i^0 \times \frac{HP}{P_i^0})) + (\sum_{\tau_e^1 \in \mathcal{S}} (C_e^1 \times \lceil \frac{HP}{P_e^1} \rceil)) \tag{3.5}$$

- Second we divide the obtained result by OC, i.e., the possible activation number, to affirm that in each period the same amount of execution time will be executed. Thus, the server capacity value is calculated as below:

$$C^s = \lceil \frac{HP - Q}{OC} \rceil \tag{3.6}$$

Example 2: Let us take the previous example, we suppose that $C_1^0 = 2$ and $C_2^0 = 1$ are the WCET for τ_1^0 and τ_2^0 respectively. The unused processing time in the hyper-period, denoted t, is calculated as below:

$$t = HP - (C_1^0 x \frac{HP}{P_1^0} + C_2^0 x \frac{HP}{P_2^0}) = 7 \tag{3.7}$$

$$C^s = \lfloor \frac{t}{OC} \rfloor = \lfloor \frac{7}{3} \rfloor = 2 \tag{3.8}$$

3.4.2.2 Computing Aperiodic Tasks Soft Deadlines:

After having parameterize the server, it only remains to know how the aperiodic tasks will be performed on the server. Thus, a soft deadline will be affected to each aperiodic task to guide the schedulability.

We suppose that the aperiodic task with the smallest C_l^2 gets the highest priority, we calculate the deadlines D_l^2 as following:

$$D_l^2 = \sum_{x=1}^{x=k} C_x^2 \times \alpha_x \tag{3.9}$$

where,

$$\alpha_x = \begin{cases} 1 \text{ if } (C_l^2 > C_x^2) \text{ or } (C_l^2 = C_x^2 \text{ and } l \geq x), \\ 0 \text{ else.} \end{cases} \tag{3.10}$$

3.4.3 Hard Real-time Constraints Characterization:

As mentioned previously, this contribution works on real-time constraints with various types: soft deadlines for aperiodic tasks and hard ones for periodic and sporadic tasks. Thus, this contribution aims to provide a feasible schedule for periodic and sporadic tasks. In fact, it assigns for each task a relative hard deadline so that these tasks meet their deadlines while using the EDF scheduling algorithm.

As the hyper-period is the smallest interval of time after which the periodic patterns of all the tasks are repeated, the computing of hard deadlines for each periodic or sporadic task is performed on the hyper-period.

A real-time system must behave in a way that can be predicted mathematically. In this context, the main idea of this contribution for effective characterization of tasks' deadlines is based on predicting mathematically the maximum cumulative amount of time that has to be executed before a given task. However, the response time of each job instance from the same task is likely to differ. This is due to the difference in the amount of work to be done between the instant when the given job is ready for execution, and the instant when it starts its execution (see Example 3 below).

Example 3: Let us consider three periodic tasks where deadlines are equal to periods: $\tau_1^0(0,1,5,5)$, $\tau_2^0(0,2,4,4)$, and $\tau_3^0(0,3,7,7)$. We assume that these tasks are scheduled according to the EDF algorithm.
Figure 3.3 illustrates the execution of these tasks. We observe that the amount of work executed before jobs of the same task differs from one to another. For example, before executing τ_{11}^0 the total amount of time that the processor executes is 2, while before executing the second job τ_{12}^0 the total amount of time that the processor executes is equal to 3, etc.

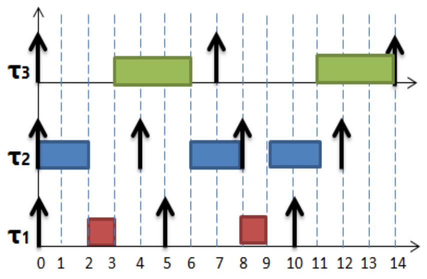


Figure 3.3: Tasks' execution.

Thus, the first step is determining the deadlines of each task jobs while computing:

- the maximum cumulative execution time requested by other periodic and sporadic jobs that have to be executed before the considered job on the hyper-period,
- the maximum cumulative execution time requested by aperiodic tasks that may occur before this job.

computing hard periodic tasks deadlines:
As aforementioned, we need firstly to compute the maximum cumulative execution time requested by periodic and sporadic jobs that may be executed before a periodic job τ_{ij}^0, denoted Δ_{ij}.

Δ_{ij} includes three factors:

- Δ_{ij}^0: which is the maximum cumulative execution time requested by the other instances of task τ_i^0 that have to be executed before jthjob τ_{ij}^0. Thus, we need to determine the number of these instances and then multiply it by the execution time as presented in 4.4:

$$\Delta_{ij}^0 = (j-1) \times C_i^0 \tag{3.11}$$

- Δ_{ij}^1: which is maximum cumulative execution time requested by jobs having maximum absolute deadlines less than that of job τ_{ij}^0 or having maximum absolute deadlines equal to that of job τ_{ij}^0 and arrival times less than that of job τ_{ij}^0. Δ_{ij}^1 is given by:

$$\Delta_{ij}^1 = \sum_{\tau_l^0 \in \mathcal{P}} (C_l^0 \times \beta_{\tau_l^0}) + \sum_{\tau_l^1 \in \mathcal{S}} (C_l^1 \times \beta_{\tau_l^1}) \tag{3.12}$$

where,

$\beta_{\tau_l^0}$ is the number of jobs produced by a periodic task τ_l^0 having maximum absolute deadlines and arrival time equal to those of job τ_{ij}^0, represented as

$$\beta_{\tau_l^0} = \left\lceil \frac{(j-1)P_i^0 + Dmax_i^0 - Dmax_l^0}{P_l^0} \right\rceil \tag{3.13}$$

$\beta_{\tau_l^1}$ is the number of jobs produced by a sporadic task τ_l^1 having maximum absolute deadlines and arrival time equal to those of job τ_{ij}^0, represented as

$$\beta_{\tau_l^1} = \left\lceil \frac{(e-1)P_e^1 + Dmax_e^1 - Dmax_l^1}{P_l} \right\rceil \tag{3.14}$$

- Δ_{ij}^2: which is maximum cumulative execution time requested by the periodic and sporadic jobs having maximum absolute deadlines equal to that of job τ_{ij}^0 and arrival times equal to that of job τ_{ij}^0 and their indices are less than i, i.e., in case of equality we will refer to apply the strategy of first in first out (FIFO) by assuming that the task with the smallest index is executed at first. Δ_{ij}^2 is given by:

$$\Delta_{ij}^1 = \sum_{\tau_l^0 \in \mathcal{P}} (C_l^0 \times \beta_{\tau_l^0}) + \sum_{\tau_l^1 \in \mathcal{S}} (C_l^1 \times \beta_{\tau_l^1}) \tag{3.15}$$

where,

$\beta_{\tau_l^0}$ is the number of jobs produced by a periodic task τ_l^0 having maximum absolute deadlines equal to that of job τ_{ij}^0 and arrival times equal to that of job τ_{ij}^0 and their indices are less than i, represented as

$$\beta_{\tau_l^0} = \left\lceil \frac{(j-1)P_i^0 + Dmax_i^0 - Dmax_l^0}{P_l^0} \right\rceil + 1 \tag{3.16}$$

$\beta_{\tau_l^1}$ is the number of jobs produced by a sporadic task τ_l^1 to be executed before job τ_{ij}^0, having maximum absolute deadlines equal to that of job τ_{ij}^0 and arrival times equal to that of job τ_{ij}^0 and their indices are less than i, represented as

$$\beta_{\tau_l^1} = \left\lceil \frac{(e-1)P_e^1 + Dmax_e^1 - Dmax_l^1}{P_l} \right\rceil + 1 \tag{3.17}$$

Note: jobs having maximum absolute deadlines greater than that of job τ_{ij}^0 are executed after τ_{ij}^0, i.e., we do not consider their cumulative execution time.

Thus, the maximum cumulative execution time Δ_{ij} requested by other periodic and sporadic jobs, at runtime, that may be executed before τ_{ij}^0 is given by:

$$\Delta_{ij} = \Delta_{ij}^0 + \Delta_{ij}^1 + \Delta_{ij}^2 \tag{3.18}$$

Secondly, we need to compute the maximum cumulative execution time requested by aperiodic tasks that may occur before a periodic job τ_{ij}^0, denoted Δ_{ij}^s. In order to ensure that periodic tasks will certainly respect their deadlines even if an aperiodic task is executed at runtime, we assume that at each NPS server period, there is an aperiodic task to execute. Thus, Δ_{ij}^s is given by:

$$\Delta_{ij}^s = \sum_{\tau_l^2 \in \mathcal{A}} (C_l^2 \times \lceil \frac{Pi^0}{P^s} \rceil) \tag{3.19}$$

After computing the maximum cumulative execution time Δ_{ij} to be performed before the execution of τ_{ij}^0, we proceed to calculate its deadline d_{ij}^0. If Δ_{ij} is achieved before the activation of τ_{ij}^0, i.e., $\Delta_{ij} < r_{ij}^0$, then as soon as it is active, τ_{ij}^0 starts its execution. Otherwise, τ_{ij}^0 is executed after a delay equal to $\Delta_{ij} - r_{ij}^0$,i.e., the remained cumulative execution time of other jobs to be executed after r_{ij}^0. Thus, the value d_{ij}^0 that guarantees the feasibility of τ_{ij}^0 takes the form

$$d_{ij}^0 = \begin{cases} \Delta_{ij}^s + \Delta_{ij} - r_{ij}^0 C_i^0 \\ \text{if } \Delta_{ij} > r_{ij}^0, \\ \Delta_{ij}^s + C_i^0 \text{ else.} \end{cases} \tag{3.20}$$

For each periodic task, the maximum among its calculated jobs deadlines will be its relative deadline. Thus, the deadline D_i^0 of task τ_i^0 is expressed by

$$D_i^0 = max\{d_{ij}^0\} \tag{3.21}$$

Finally, D_i^0 is the fixed deadline for τ_i^0 that will surely be meeted at runtime. In other words, even if an aperiodic task occurs at runtime, the periodic and sporadic tasks will certainly respect their deadlines and the response time of aperiodic task is improved as the invocation of aperiodic task execution is considered when calculating hard deadlines.

computing hard sporadic tasks deadlines:

A sporadic task τ_e runs at most each P_e^1. In this case, we can estimate the value r_{ef}^1 of each job τ_{ef}^1. Therefore, to calculate the deadline of a sporadic task, we follow the same procedure of a periodic task deadline calculation. For that, the deadline d_{ef} of the sporadic job τ_{ef}^1 is given by:

$$d_{ef}^1 = \begin{cases} \Delta_{ef}^s + \Delta_{ef} - r_{ef}^1 C_e^1 + \\ \text{if } \Delta_{ef} > r_{ef}^1, \\ \Delta_{ef}^s + C_{e^1} \text{ else.} \end{cases} \tag{3.22}$$

where,

Δ_{ef} is the maximum cumulative execution time requested by periodic and sporadic jobs that may be executed before the sporadic job τ_{ef}^1, itis given by:

$$\Delta_{ef} = \Delta_{ef}^0 + \Delta_{ef}^1 + \Delta_{ef}^2 \tag{3.23}$$

where,

$$\Delta_{ef}^0 = (f-1) \times C_e^1 \tag{3.24}$$

and,

$$\Delta_{ef}^1 = \sum_{\tau_l^0 \in \mathcal{P}} (C_l^0 \times \beta_{\tau_l^0}) + \sum_{\tau_l^1 \in \mathcal{S}} (C_l^1 \times \beta_{\tau_l^1}) \tag{3.25}$$

where,

$$\beta_{\tau_l^0} = \left\lceil \frac{(f-1)P_e^1 + Dmax_e^1 - Dmax_l^0}{P_l^0} \right\rceil \tag{3.26}$$

and,

$$\beta_{\tau_l^1} = \left\lceil \frac{(f-1)P_e^1 + Dmax_e^1 - Dmax_l^1}{P_l^1} \right\rceil \tag{3.27}$$

and,

$$\Delta_{ef}^2 = \sum_{\tau_l^0 \in \mathcal{P}} (C_l^0 \times \beta_{\tau_l^0}) + \sum_{\tau_l^1 \in \mathcal{S}} (C_l^1 \times \beta_{\tau_l^1} \tag{3.28}$$

where,

$$\beta_{\tau_l^0} = \left\lceil \frac{(f-1)P_e^1 + Dmax_e^1 - Dmax_l^0}{P_l^0} \right\rceil + 1 \tag{3.29}$$

and,

$$\beta_{\tau_l^1} = \left\lceil \frac{(f-1)P_e^1 + Dmax_e^1 - Dmax_l^1}{P_l^1} \right\rceil) + 1 \tag{3.30}$$

and, Δ_{ef}^s the maximum cumulative execution time requested by aperiodic tasks that may occur before the sporadic job τ_{ef}^1, denoted Δ_{ef}^s.

$$\Delta_{ef}^s = \sum_{\tau_l^2 \in \mathcal{A}} (C_l^2 \times \lceil \frac{Pe^1}{P^s} \rceil) \tag{3.31}$$

Finally, D_e^1 is the fixed deadline for τ_e^1 that will surely be meeted at runtime.

$$D_e^1 = max\{d_{ef}^1\} \tag{3.32}$$

3.4.4 New Solution for Deadline Calculation of Periodic, Sporadic and Aperiodic Real-time Tasks

The algorithm below implements the proposed approach.

Algorithm 1 New method for deadline calculation

```
Require: $\mathcal{P}$, $\mathcal{S}$,$\mathcal{A}$, OC
Ensure: $D_i^0$, $D_e^1$, $D_o^2$
 1: function $NPS\_Parameters(HP, OC)$
 2:     $P^s = \lfloor \frac{HP}{OC} \rfloor$
 3:     $Q = (\sum_{\tau_i^0 \in \mathcal{P}} (C_i^0 \times \frac{HP}{P_i^0})) + (\sum_{\tau_e^1 \in \mathcal{S}} (C_e^1 \times \lceil \frac{HP}{P_e^1} \rceil))$
 4:     $C^s = \lceil \frac{HP - Q}{OC} \rceil$
 5: end function
 6: for all $\tau_o^2 \in \mathcal{A}$ do
 7:     $D_o^2 = \sum_{x=1}^{x=k} C_x^2 \times \alpha_x$
 8: end for
 9: function $H\_Dead\_Calc(\mathcal{P}, \mathcal{S})$
10:     for all $\tau_i^0 \in \mathcal{P}$ do
11:         $\Delta_{ij} = \Delta_{ij}^0 + \Delta_{ij}^1 + \Delta_{ij}^2$
12:         if $\Delta_{ij} > r_{ij}^0$ then
13:             $d_{ij}^0 = \Delta_{ij}^s + C_i^0 + \Delta_{ij} - r_{ij}^0$
14:         else
15:             $d_{ij}^0 = \Delta_{ij}^s + C_i^0$
16:         end if
17:     end for
18:     $D_i^0 = max\{d_{ij}^0\}$
19:     for all $\tau_e^2 \mathcal{S}$ do
20:         $\Delta_{ef} = \Delta_{ef}^0 + \Delta_{ef}^1 + \Delta_{ef}^2$
21:         if $\Delta_{ef} > r_{ef}^0$ then
22:             $d_{ef}^0 = \Delta_{ef}^s + C_e^2 + \Delta_{ef} - r_{ef}^2$
23:         else
24:             $d_{ef}^2 = \Delta_{ef}^s + C_e^2$
25:         end if
26:     end for
27:     $D_e^2 = max\{d_{ef}^2\}$
28: end function
```

It uses the following functions:

- $NPS_Parameters(HP, OC)$which is a function that returns the NPS server parameters.
- $H\_Dead\_Calc(\mathcal{P}, \mathcal{S})$ which a function that returns the periodic and sporadic

hard deadlines. This function starts by computing jobs deadlines and then for each periodic/sporadic task, it calculates its fixed deadline to be equal to the maximum of its jobs' deadlines.

3.5 Simulation and Conducted Experimentation

This section reveals the developed tool called GIGTHIS-TOOL. This simulator serves to implement the concepts of the proposed methodology. Furthermore, we present a case study to test and evaluate the proposed approach.

3.5.1 Developed Environment: GIGTHIS-TOOL

In order to bring out the importance of the proposed approach and to test it, we have created a visual software environment, called GIGTHIS-TOOL. This simulator is an open source environment that applies the services of the proposed methodology. GIGTHIS-TOOL serves to facilitate the task of a designer; the main actor interacting with the simulator. In fact, it can be simply used by designers to compute and display effective deadlines, with few clicks, in arranged tables and in short time. Hence, this project can be a future reference for industrial partners who will be focusing on various real-time applications design.

The simulator is a multi-panel interface application as presented in figure 3.4, each area of functionality appears in a specified panel. Panel number 1 allows the designer to define the initial parameters of the real-time system. i.e., allows user to fill in with the desired task parameters. After clicking on button validate, the designer has access to the second panel to:

- Compute the hyper-period.
- Generate the aperiodic tasks' occurrence number.
- Parameterize the NPS server: calculate it capacity and it period.
- Compute tasks deadlines by applying the formulas of the proposed methodology.

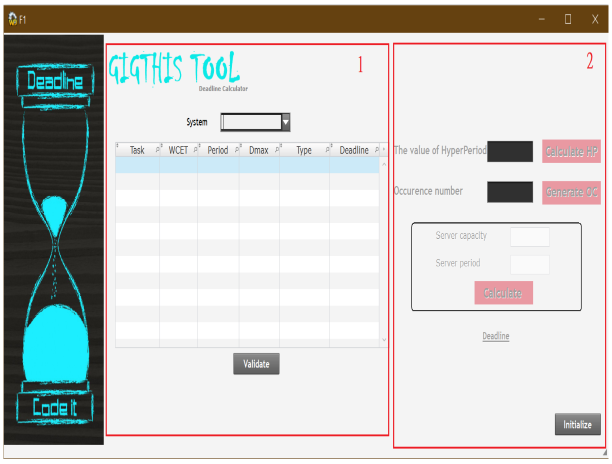


Figure 3.4: The main interface of GIGTHIS-TOOL.

3.5.2 Case Study

The considered case study is the anti-lock braking system (ABS)[90]. It is an active safety system designed to prevent a car's wheels from locking up and avoiding uncontrolled skidding. In fact, it allows the wheels of the vehicle to maintain tractive contact with the road surface according to driver inputs while braking. Figure3.5 illustrates a descriptive diagram of the ABS. The system consists of:

- a speed sensor for each wheel that detects the speed of the wheel and delivers this information to the calculator,
- a calculator that treats the information sent by the sensor and If the speed becomes too low and close to blocking, the computer alerts the hydraulic unit to reduce the pressure,
- a hydraulic system: regulates the braking pressure.

Thus, thanks to the speed sensor, the calculator and the hydraulic unit, the pressure is regulated when pressing the brake pedal to obtain the best braking efficiency without locking.

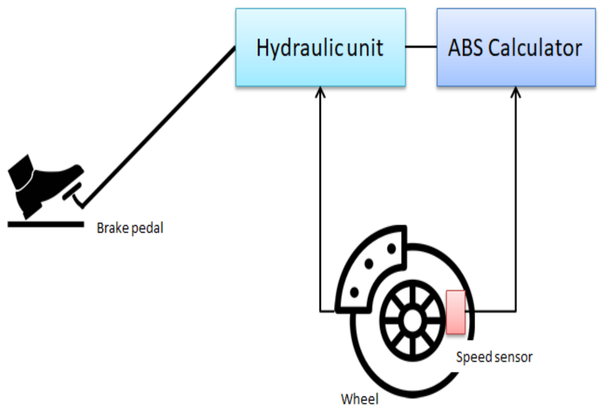


Figure 3.5: Case study modelisation.

We assume that the speed sensor reads the speed every 15s for two seconds, then it sends this information to the calculator in the same period. This latter takes 4s to evaluate the speed value every 20s. If the measured speed is not in the desired value, then the calculator sends an alert for 3s to the hydraulic unit, which is activated for 2s to correct this problem. We present in table3.1 the system's tasks parameters.

Table 3.1: System tasks

Task	Fonction	WCET	Period	Maximum dead-line
τ_1^0	detetct speed	2	15	10
τ_2^0	send speed value to the calculator	2	15	15
τ_3^0	trait the speed value	4	20	18
τ_1^1	alert the hydraulic unit	3	20	24
τ_1^2	adjust pressure	2		

Thus, Π is implemented by three sets: $\mathcal{P} = \{\tau_1^0, \tau_2^0, \tau_3^0\}$, $\mathcal{S} = \{\tau_1^1\}$ and $\mathcal{A} = \{\tau_1^2\}$.

We have, $HP = LCM\{15, 20\} = 60s$.

Let's suppose that the parameter λ of the Poisson distribution is equal to 1 occurences in 30 seconds. Thus, in the hyper-period we have $\frac{HP}{30} \times \lambda = \frac{60}{30} \times 1 = 2$ occurences, i.e., $OC = 2$.

3.5.2.1 Parameterizing the NPS server:

The periodic server parameters P^s and C^s are computed respectively as following:
According to Equation (3.2), $P^s = \lfloor \frac{60}{2} \rfloor = 30$
According to Equation (3.5), $Q = 2 \times 4 + 2 \times 4 + 4 \times 3 + 3 \times 3 = 37$
According to Equation (3.6), $C^s = \lfloor \frac{60 - 37}{3} \rfloor = 7$

After that, we calculate aperiodic tasks' deadlines. According to Equation (3.9)

$$\begin{aligned} D_1^1 &= C_1^1 \times \alpha_1 \\ &= 2 \times 1 = 2 \end{aligned}$$

3.5.2.2 Hard Real-time Constraints Characterization:

We move to periodic and sporadic tasks' deadlines calculation. As an example, we take the calculation of deadline D_3^0 for task τ_3^0. The number of jobs of task τ_3^0 in the hyper-period HP is $\frac{HP}{P_3^0} = \frac{60}{20} = 3$ jobs.

Job τ_{31}^0: First, we compute the maximum cumulative execution time requested by periodic and sporadic tasks that may occur before this job,i.e., Δ_{31}.
According to Equation (4.9), we have to calculate Δ_{31}^0, and Δ_{31}^1 and Δ_{31}^2:
According to Equation (4.4)
$\Delta_{31}^0 = (1 - 1) \times 4 = 0$
According to Equation (4.5)
$\Delta_{31}^1 = C_1^0 \times 1 + C_2^0 \times 1 = 4$

For the sporadic task τ_1^1, the first job τ_{11}^1 has an absolute deadline greater than that of τ_{31}^0. Therefore, this job is not considered when computing the deadline of τ_{31}^0.
According to Equation (4.7)
$\Delta_{31}^2 = 0$ because there is no periodic or sporadic job having a maximum absolute deadline and arrival time equal to that of job τ_{31}^0.
Thus, $\Delta_{31} = 4$.

Second, we compute the maximum cumulative execution time requested by aperiodic tasks that may occur before this job,i.e., Δ_{31}^s.

$$\begin{aligned} \Delta_{31}^s &= \sum_{\tau_l^2 \in \mathcal{A}} (C_l^2 \times \lceil \frac{P3^0}{P^s} \rceil) \\ &= C_1^2 \times \lceil \frac{P3^0}{P^s} \rceil \\ &= 2 \times \lceil \frac{20}{30} \rceil \\ &= 2 \end{aligned}$$

We have $r_{31}^0 = 0$, so $\Delta_{31} > r_{31}^0$. Thus, according to Equation (4.10),

$$\begin{aligned} d_{31}^0 &= \sum_{\tau_l^2 \in \mathcal{A}} (C_l^2 \times \lceil \frac{Pi_1}{P^s} \rceil) + C_3^0 + \Delta_{31} - r_{31} \\ &= 2 + 4 + 4 - 0 = 10 \end{aligned}$$

Job τ_{32}^0: First of all, we calculate the cumulative execution time Δ_{32}^0. According to Equation (4.9), we have to calculate Δ_{32}^0, and Δ_{32}^1 and Δ_{32}^2:

According to Equation (4.4)
$\Delta_{32}^0 = (2 - 1) \times 4 = 4$
According to Equation (4.5)
$\Delta_{31}^1 = C_1^0 \times 2 + C_2^0 \times 2 + C_1^1 \times 1 = 11$

For the sporadic task τ_1^1, the second job τ_{11}^1 has an absolute deadline less than that of τ_{32}^0. Therefore, this job is considered when computing the deadline of τ_{32}^0.
According to Equation (4.7)
$\Delta_{32}^2 = 0$ because there is no periodic or sporadic job having a maximum absolute deadline and arrival time equal to that of job τ_{32}^0 is equal to that of job τ_{32}^0.
Thus, $\Delta_{31} = 4 + 11 = 15$.

We have $r_{32}^0 = 20$, so $\Delta_{31} < r_{31}^0$.

Second, we compute the maximum cumulative execution time requested by aperiodic tasks that may occur before this job,i.e., Δ_{32}^s.

$$\begin{aligned} \Delta_{32}^s &= \sum_{\tau_l^2 \in \mathcal{A}} (C_l^2 \times \lceil \frac{P3^0}{P^s} \rceil) \\ &= C_1^2 \times \lceil \frac{P3^0}{P^s} \rceil \\ &= 2 \times \lceil \frac{20}{30} \rceil \\ &= 2 \end{aligned}$$

Thus, according to Equation (4.10),

$$\begin{aligned} d_{32}^0 &= \Delta_{32}^s + C_3^0 \\ &= 2 + 4 = 6 \end{aligned}$$

Job τ_{33}: First of all, we calculate the cumulative execution time Δ_{33}. According to Equation (4.9), we have to calculate Δ_{33}^0, and Δ_{33}^1 and Δ_{33}^2:
According to Equation (4.4)
$\Delta_{32}^0 = (3 - 1) \times 4 = 8$
According to Equation (4.5)

$\Delta_{31}^{1} = C_1^0 \times 4 + C_2^0 \times 4 + C_1^1 \times 2 = 22$

According to Equation (4.7)
$\Delta_{32}^{2} = 0$ because there is no periodic or sporadic job having a maximum absolute deadline and arrival time equal to that of job τ_{33}^{0} is equal to that of job τ_{32}^{0}.

Thus, $\Delta_{31} = 8 + 22 = 30$.

We have $r_{33}^{0} = 40$, so $\Delta_{31} < r_{33}^{0}$.

Second, we compute the maximum cumulative execution time requested by aperiodic tasks that may occur before this job,i.e., Δ_{32}^{s}.

$$\begin{aligned}\Delta_{32}^{s} &= \sum_{\tau_l^2 \in \mathcal{A}} (C_l^2 \times \lceil \frac{P3^0}{P^s} \rceil) \\ &= C_1^2 \times \lceil \frac{P3^0}{P^s} \rceil \\ &= 2 \times \lceil \frac{20}{30} \rceil \\ = 2\end{aligned}$$

Thus, according to Equation (4.10),

$$\begin{aligned}d_{33}^{0} &= \Delta_{33}^{s} + C_3^0 \\ &= 2 + 4 = 6\end{aligned}$$

Finally, according to Equation(4.11),
$D_3^0 = max\{d_{31}^0, d_{32}^0, d_{33}^0\} = max\{10, 6, 6\} = 10$

3.5.2.3 Case Study Results and Evaluation

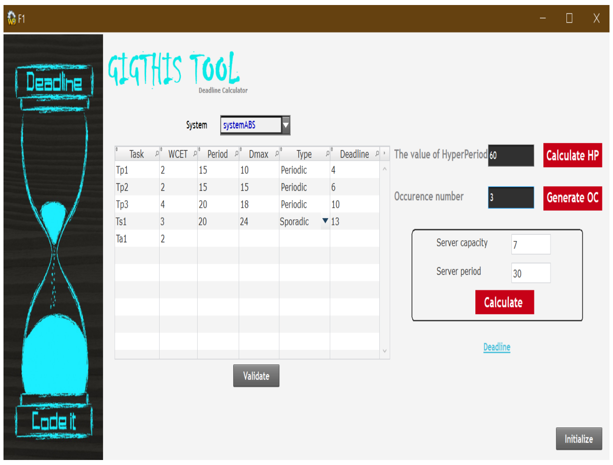
After completing the execution of the proposed approach, the calculated effective deadlines of the different tasks are given in table3.2.

Table 3.2: Tasks' calculated deadlines

Task	τ_1^0	τ_2^0	τ_3^0	τ_1^1	τ_1^2
Calculated Deadline	4	6	10	13	2

Figure3.6 illustrates the results of the case study by using GIGTHIS-TOOL[1].

[1]As writing index and exponent is not allowed in GIGTHIS-TOOL then the tasks notation becomes as follows: Tp_i for a periodic task, Ts_e for a sporadic task and Ta_o for an aperiodic one.



System: systemABS

Task	WCET	Period	Dmax	Type	Deadline
Tp1	2	15	10	Periodic	4
Tp2	2	15	15	Periodic	6
Tp3	4	20	18	Periodic	10
Ts1	3	20	24	Sporadic	13
Ta1	2				

The value of HyperPeriod: 60

Occurence number: 3

Server capacity: 7

Server period: 30

Figure 3.6: The calculation results from GIGTHIS-TOOL.

Figure3.7 shows the scheduling of tasks after the execution of the proposed approach. We note that the real-time constraints are respected by the proposed methodology, and the response time of the aperiodic task is equal to its execution time, i.e, it is executed with the best response time.

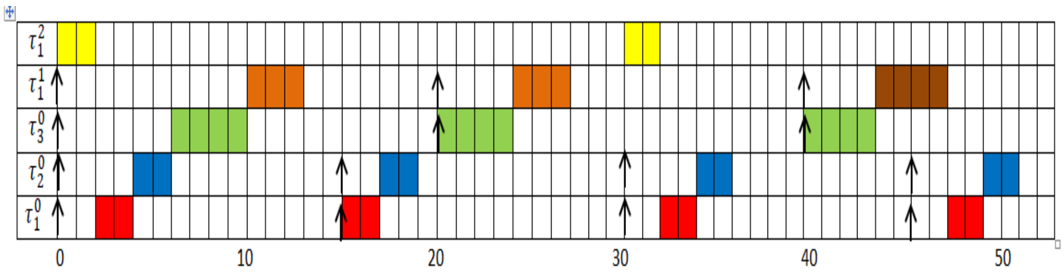


Figure 3.7: Scheduling of tasks after the execution of the proposed approach.

A second case study, using GIGTHIS-TOOL,is presented in our website https://projects-lisi-lab.wixsite.com/gigthistool.

3.6 Discussion

In this section we highlight the contribution and originality of the prposed approach through discussing the use of the proposed solution in comparison to other existing works.

We have randomly generated instances with 10 to 50 periodic and sporadic tasks. We compare the proposed approach with the work reported in [7], where the critical scaling factor (CSF) algorithm is developed. We focus on the reduction rate of the calculated deadlines compared to maximum deadlines.

Figure 3.8 shows that the reduction rates of deadlines by using [7] are smaller than those by using the proposed work. We conclude that the rate of reduction of deadlines in [7] can be improved. Hence, the gain is offered by the proposed approach. Moreover, the gain is more significant when increasing the number of tasks. If 10 tasks are considered, then the gain is equal to (0.31-0.2) = 0.11, and if 50 tasks are considered, then the gain is equal to (0.61-0.35) = 0.26.

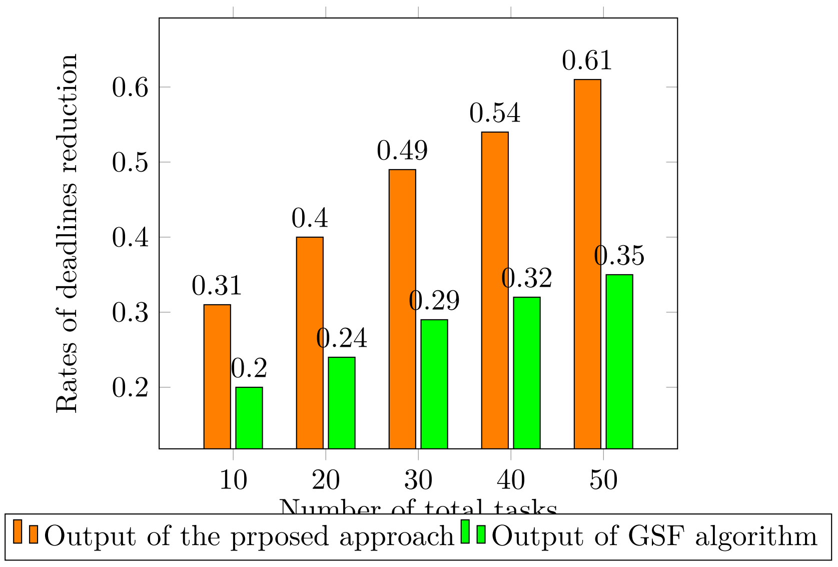


Figure 3.8: Rates of deadlines reduction in the case of the proposed approach and in the case of GSF algorithm.

Real-time systems are widely used in several fields where the development cycle takes several months, even several years. This can generate high costs relative to the development time, which could be lengthened if it is belatedly realized that the chosen deadlines do not allow the system feasibility. Thus, we propose in this chapter a new approach that parameterizes feasible scheduling of real-time tasks with various types and constraints in the context mono-core architecture. The numerical results show that this methodology reduces the development time by computing the deadlines for periodic and sporadic tasks to be certainly respected at runtime, and allows to minimize the response time for aperiodic tasks.

Despite its advantages, there is still some scopes that need to be improved in this contribution. For example, the software tool is still in a proof of concept level and needs to be more robust/mature in order to be used in more complex real-life systems. Moreover, many other factors that may affect the real-time correctness should be studied.

3.7 Conclusion

This chapter first discusses real-time systems executing periodic, sporadic, and aperiodic tasks on a mono-core architecture. The proposed work is to create the NPS server invoked periodically to run aperiodic tasks and then calculate the soft and hard deadlines for all the tasks. As the invocation of aperiodic task execution is considered when calculating hard deadlines, this methodology ensures certainly the real-time feasibility of periodic and sporadic tasks. A new visual simulator called GIGTHIS-TOOL is developed to apply the services of the proposed methodology.

As mentioned above, there are many other constraints that can affect real-time accuracy that need to be considered. Thus, the following chapter works on reconfiguration aware while dealing with real-time constraints, energy, and resource sharing.

Chapter 4

Configuring Feasible Reconfigurable Real-time System under Energy and Resource Sharing on Mono-core Architecture

4.1 Introduction

The previous chapter copes with three main factors, which are: mixed tasks set, hard and soft temporal constraints, and mono-core architecture. In this chapter, the main focus is solving additional constraints which are reconfiguration property, renewable energy and tasks dependency if one or more resources are shared by several tasks. Thus, the main aim is parametrizing a feasible reconfigurable real-time system that may be powered by a renewable energy source and their tasks may share resources on mono-core architecture.
The outline of the chapter is organized as follows: First, we present the motivation section. Then, we present the computational model and the assumptions and problem formulation. After that, we detail the main proposed contribution. A formal case study is proposed in order to pinpoint the main problem studied in this chapter. Finally, we present the discussion section to analyze and interpret this chapter's findings.
Note that this chapter has been published in the international journal IEEE Transactions on Automation Science and Engineering [40].

4.2 Motivation

Real-time correctness is the most prominent part of real-time systems' feasibility. However, such system may be surrounded and influenced by several factors. Thus, a real-time system should be adapted to their environment changes while applying the adequate reconfiguration scenario by adding/removing real-time tasks. In fact, a reconfigurable real-time system is a set of implementations, each of which is encoded by real-time periodic software tasks. At runtime, only one implementation is executed. The moving from one implementation to another results from adding/removing real-time tasks. These tasks may be independent or dependent when they use same resources. Moreover, each system needs an energy source to be in an operating mode. This source can be permanent, as the electricity sector provides a continuous energy load. Thus, there is no energy starvation. It can also be renewable as solar energy. Naturally, the amount of harvested renewable energy varies over time due to changing environmental conditions, like the angle of sunlight incidence, cloud density, etc. Thus, the system may have energy starvation. So in this situation, in addition to real-time and resource sharing constraints, the system must consider the renewable energy limitations.

To maintain the correctness of a real-time system, the designer should predict the behavior of a real-time system by ensuring that all the tasks meet their deadlines while having the required energy and resources. It is within this context that the problem, considered in this paper, is how to calculate the effective deadlines of the different periodic tasks in the different implementations under possibly the predicted renewable energy source and the sharing resource constraints.

The main contribution proposes is a new offline methodology for calculating effective deadlines under resource and energy constraints while minimizing the context switching on a mono-core real-time system. This methodology is divided into three solutions, each of which is dedicated to ensuring the respect of a given constraint. Thus, it is usable even if the system is powered by a permanent energy source (no energy requirement) and even if the system does not use shared resources (no dependence relation between tasks):

- The first solution serves to compute the deadlines ensuring the real-time system feasibility and also minimizes the number of context switches by assigning the highest priority to the task with the smallest maximum deadline.
- The second solution computes deadlines ensuring the respect of energy constraints
- The third solution compute deadlines ensuring the respect of resource sharing constraints.

This methodology provides deadlines without affecting neither the load nor the processor speed while reducing the calculation time. Moreover, the calculated deadlines will certainly be respected at runtime without wasting time doing the schedulability tests. We develop a new simulator called DEADCALC that integrates a new tool called RANDOM-TASK for applying and evaluating the proposed solutions. The conducted experimentation proves that this methodology provides deadlines with affecting neither the load nor the processor speed, while reducing the calculation time.

4.3 Formalization

In this section, we introduce the mathematical expressions that represent the characteristics of a reconfigurable real-time system in mono-core architecture. Moreover, we introduce a formal description of the energy model.

4.3.1 System Model

We focus in this chapter on periodic tasks. Thus, based on the formalization made in first contribution, we define a real-time reconfigurable system, denoted by Π, as a task set $\mathcal{P}$ containing n periodic software tasks, i.e., $\mathcal{P} = \{\tau_1, \ldots, \tau_n\}$, which is the set of all tasks that can implement Π. We suppose that all these tasks are activated at $t = 0$.

as mentioned above, one of the constraints to be considered in the chapter is the reconfiguration property. We assume that a reconfiguration scenario is an operation that allows the adding or removing software tasks. Consequently, we define a reconfigurable real-time system as a set of implementations, each of which is encoded by a subset of real-time software tasks, and is able to perform the system's intended function under stated conditions without failure for a given period of time. In other words, at a given time t, only one subset of $\mathcal{P}$ is executed after applying a reconfiguration scenario according to user requirements (and possibly an evolution of the environment). Thus, we define by $\mathcal{I}$ the set of system implementations,i.e., I_k, $k \in [1, ..., x]$, i.e., $\mathcal{I} = \{I_1,\ ...,\ I_x\}$. Each element I_k, $k \in [1, ..., x]$, subsets of $\mathcal{P}$.

4.3.2 Task Model

In the first contribution, a periodic task τ_i^0 in $\mathcal{P}$, $i \in [1, ..., n]$, is a 5-tuple (R_i^0, C_i^0, P_i^0, D_i^0, $Dmax_i^0$). However, in this contribution, there are two additional parameters for each task τ_i^0:

- E_i^0 is the required energy for τ_i^0 to be executed. In fact, the other additional constraint considered is the energy requirements.
- B_i^0 is the maximum blocking time that can delay τ_i^0. This blocking time results from taking into account the resource sharing constraint in this contribution. We denote by $\mathbb{R}$ the set of shared resources and $Rs$is a shared resource in $\mathbb{R}$.

Thus, each periodic task τ_i^0, $i \in [1, ..., n]$, in $\mathcal{P}$ is a 7-tuple (R_i^0, C_i^0, P_i^0, D_i^0, $Dmax_i^0$, E_i^0, B_i^0).

We assume that each task τ_i^0 in implementation I_k is denoted by τ_{ik}. It is characterized by a deadline d_{ik}^0 that guarantees the respect of real-time constraints by task τ_i^0 in implementation I_k) and by blocking time b_{ik}^0 that is the time that can delay τ_i^0 in implementation I_k.

For example, as shown in figure4.1, we consider a reconfigurable real-time system Π having four periodic tasks, $\mathcal{P} = \{\tau_1^0, \tau_2^0, \tau_3^0, , \tau_4^0\}$ and two implementations, $\mathcal{I} = \{I_1, I_2\}$, where $I_1 = \{\tau_{21}^0, \tau_{41}^0\}$, and $I_2 = \{\tau_{12}^0, \tau_{22}^0, \tau_{32}^0\}$. A reconfiguration scenario is applied at instant t_1, where the implementation before t_1 is I_2 and after t_1 is I_1. At t_1, I_2 I_1

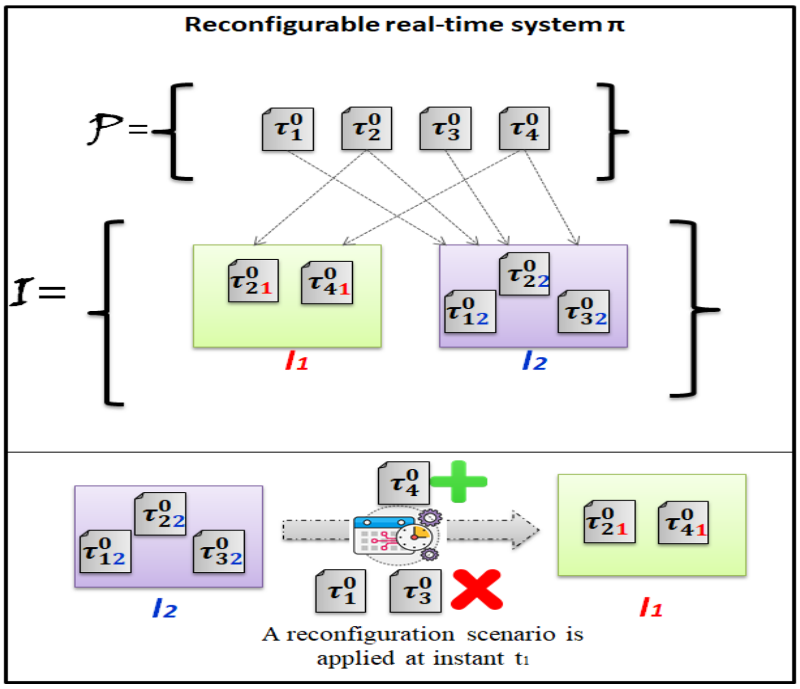

Reconfigurable real-time system π
$\mathcal{P}=\{\tau_1^0\ \tau_2^0\ \tau_3^0\ \tau_4^0\}$
$\mathcal{I}=$
τ_{21}^0 τ_{41}^0
I_1
τ_{22}^0 τ_{12}^0 τ_{32}^0
I_2
τ_4^0
τ_1^0 τ_3^0
A reconfiguration scenario is applied at instant t_1

Figure 4.1: Example of system model.

Each periodic task τ_{ik} in I_k produces an infinite sequence of jobs τ_{ijk}^0, where j is a positive integer. Each job τ_{ijk}^0 is described by a release time r_{ij}^0 and a deadline d_{ijk}^0. Also, each job τ_{ijk}^0 has two parameters s_{ijk}^0 and f_{ijk}^0 which represent the start and the end execution time, respectively.

Finally, we denote by HP_k the hyper-period which is the lowest common multiple (LCM) of the tasks'periods in I_k.

4.3.3 Energy Model

The chapter focus on real-time systems powered by renewable energy source. Energy harvesting[25] is the technique for collecting energy from various environmental sources such as solar, wind, etc. However, this type of energy is often weak and unstable. Thus, for their reliable operation, these systems require ensuring constraints in both execution time and energy consumption in order to guarantee the completion within given resource budgets.

As mentioned above, we assume that Π is alimented by a renewable energy source [106, 107]. For this, we propose an energy model, presented in 4.2, to be used to meet energy requirements. This model is characterized by:

- The amount of energy E_a estimated to be available in the battery at time $t = 0$.

- An instantaneous charging rate E^u that represents the worst case energy production per unit of time.
- The produced energy between $t = 0$ and the end execution time of job τ^0_{ijk}, denoted by $E^p_{ijk}(0, f^0_{ijk})$, where $E^p_{ijk}(0, f_{ijk}0) = \int_0^{f^0_{ijk}} E^u dt = E^u \times (f^0_{ijk} - 0)$.
- The energy consumed between $t = 0$ and the end execution time of job τ^0_{ijk}, denoted by $E^c_{ijk}(0, f^0_{ijk})$.

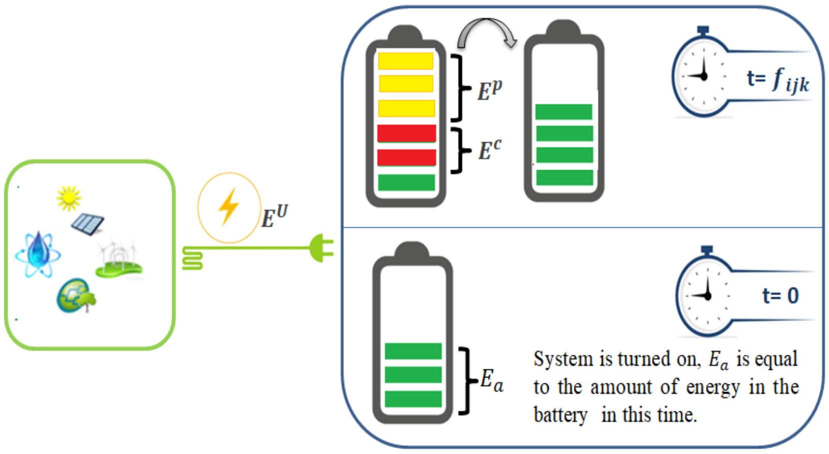

E^P
E^c
t= f_{ijk}
E^U
t= 0
E_a
System is turned on, E_a is equal to the amount of energy in the battery in this time.

Figure 4.2: Energy model schema.

4.3.4 Problems' Mathematical Formalization

To ensure that Π runs correctly, it is necessary solving the given constraints which are reconfiguration aware, renewable energy and resource sharing. Whatever the implementation, each job, even if it has been blocked because of resource sharing, should reach its deadline without energy starvation. These constraints are given as follows:

- Real-time constraint: all task's jobs has to be completed before its absolute deadline,i.e.,
$$\forall\ i,\ j \text{ and } k,\ f^0_{ijk} \leq r^0_{ij} + D^0_i \tag{4.1}$$
- Resource sharing constraint: each job, blocked because of resource sharing, has to be completed before its absolute deadline, i.e,
$$\forall\ i,\ j \text{ and } k,\ f^0_{ijk} \leq r^0_{ij} + D^0_i \tag{4.2}$$
- Energy constraint: each job has to obtain required energy for its execution to be finished before its absolute deadline, i.e.,
$$\forall\ i,\ j \text{ and } k,\ E_a(0) + E^p_{ijk}(0, f^0_{ijk}) - E^c_{ijk}(0, f^0_{ijk}) \geqslant 0 \tag{4.3}$$

4.4 Parametrizing Feasible Reconfigurable Systems Under Real-time, Energy and Resource Sharing Constraints

This section starts with an overview of the proposed methodology. Then, it presents in detail the solutions involved in this chapter.

4.4.1 Overview of the proposed methodology

The approach proposed in this chapter consists in configuring reconfigurable real-time systems implemented on a mono-core architecture. In addition to the temporal requirements, this approach ensures the energy and resource sharing constraints. Thud, this methodology presented in figure 4.3 is divided into three solutions, each of which guarantees respect of a given constraint.

- **Real-time based deadline computing solution**:it calculates effective deadlines, ensuring real-time constraints. This solution is achieved by two consecutive functions:
  - RT_S is executed to calculate the maximum of jobs' deadlines for each task in the hyper-period of any related implementation, i.e., an implementation to which the task belongs. The output is the deadline of the task in this implementation.
  - S^{RT} is executed to calculate the maximum of deadlines of each task in all related implementations.
- **Energy-based deadline computing with no starvation solution**:it is invoked when the system is powered by a renewable energy source to cope with the energy availability issue. In fact, this solution computes the portion of the time needed to produce the tasks' energy requirements and then adds it to the deadlines outputted by the first solution. The novel deadlines prevent any energy starvation as there is more time for energy production. This solution is ensured by two functions:
  - EN_S calculates the maximum portion of time needed by task jobs in the hyper-period of any related implementation to prevent energy starvation.
  - S^{EN} calculates the maximum portion of time needed by each task in all related implementations. This maximum time will be added to the deadline calculated by S^{RT} in the first solution. The novel deadlines guarantee the energy feasibility, as they give more time for energy production.
- **Resource sharing based deadline computing solution**: it is executed if the system uses shared resources to cope with the blocking time may delay a task'execution. In fact, the main aim of this solution is to determine the maximum blocking time that may delay a task execution and adds it its deadline

outputted by the second solution if the system is powered by a renewable energy source, or by the first solution otherwise. It is ensured by two functions:

– BT_S calculates the maximum blocking time of the different tasks in any related implementation.

– S^{BT} is executed to add the maximum blocking time of each task to its deadline, coming from S^{EN} if the system is powered by a renewable energy source or from SRT if not. These novel deadlines guarantee the resource sharing feasibility.

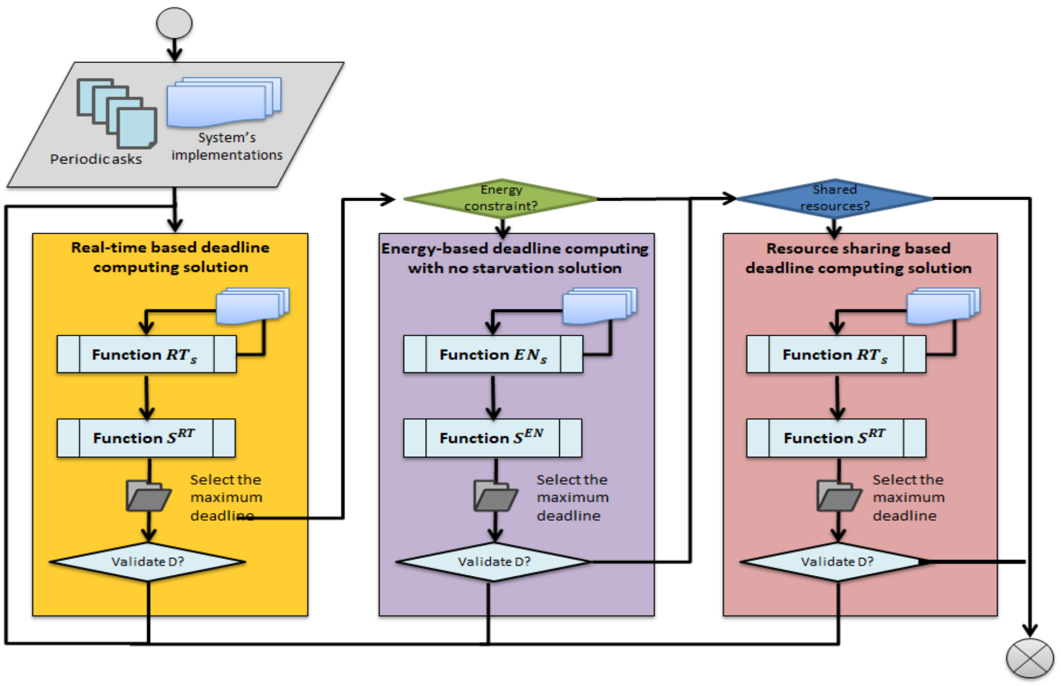

Periodic asks
System's implementations
Real-time based deadline computing solution
Function RT_s
Function S^{RT}
Select the maximum deadline
Validate D?
Energy constraint?
Energy-based deadline computing with no starvation solution
Function EN_s
Function S^{EN}
Select the maximum deadline
Validate D?
Shared resources?
Resource sharing based deadline computing solution
Function RT_s
Function S^{RT}
Select the maximum deadline
Validate D?

Figure 4.3: New methodology of effective deadlines calculation.

The fact that the proposed approach is divided into three solutions, each of which is dedicated to guarantee the respect of given constraints, that it can be applied even if there are no energy requirements or shared resources. Thus, the splitting of the solutions makes it possible to engage four branches, each of which is executed under well-defined constraints, as presented in figure 4.4. As the timing constraint is paramount to maintain the correctness of a real-time system, the real-time based deadline computing solution is invoked at the outset of each branch.

- Branch 1: it is executed to ensure the feasibility of the reconfigurable real-time system. In fact, it assumed that the system is powered by a permanent energy source, i.e., there are any energy requirements, and that tasks are independent. Thus, it is enough to execute the first solution.

- Branch 2: it is executed to ensure the feasibility of the reconfigurable real-time system powered by renewable energy and has independent tasks. Thus, it is necessary to execute the first solution to guarantee real-time correctness, and the second solution to prevent any energy starvation.

- Branch 3: it is executed to ensure the feasibility of the reconfigurable real-time system powered by a permanent energy source and their tasks share resources. Thus, it is necessary to execute the first solution and the third solution to guarantee real-time correctness even if there is a blocking time due to resource sharing.

- Branch 4: it is executed to ensure the feasibility of the reconfigurable real-time system powered by a renewable energy source, and it has one or more shared resources. In this case, it is necessary to execute successively the three solutions.

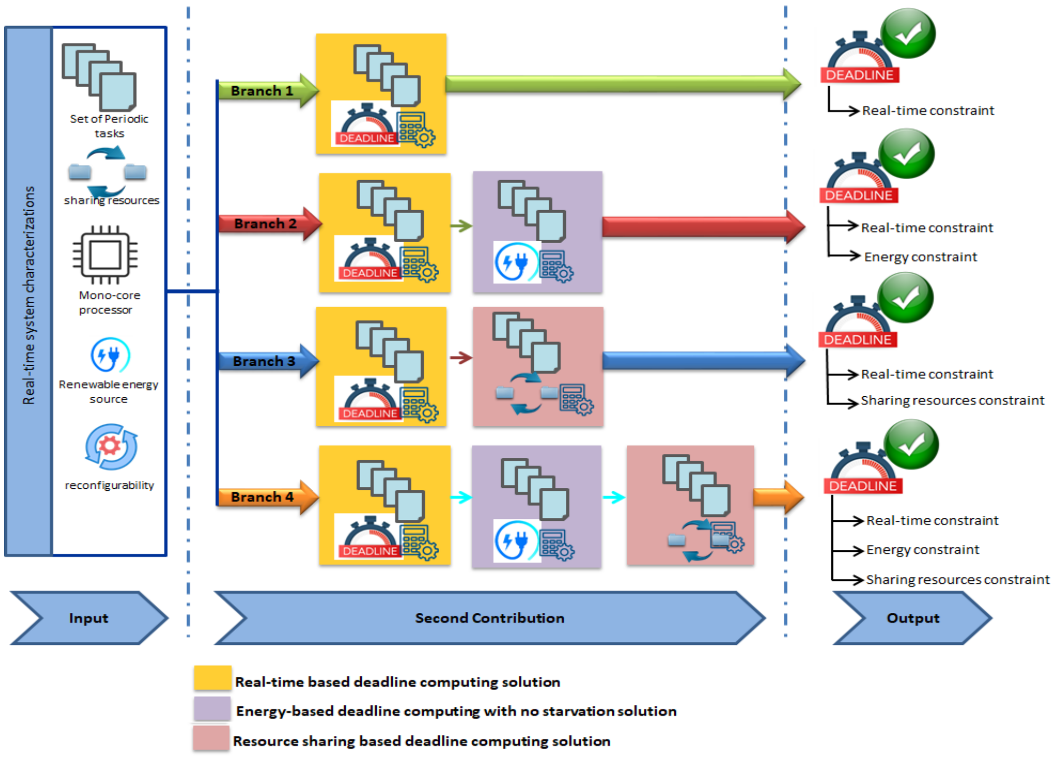

Real-time system characterizations
Set of Periodic tasks
sharing resources
Mono-core processor
Renewable energy source
reconfigurability
Branch 1
Branch 2
Branch 3
Branch 4
DEADLINE
Real-time constraint
Energy constraint
Sharing resources constraint
Input
Second Contribution
Output
Real-time based deadline computing solution
Energy-based deadline computing with no starvation solution
Resource sharing based deadline computing solution

Figure 4.4: The branches of the methodology.

4.4.2 Real-time Based Deadline Computing Solution

The main aim of this solution is the calculation of tasks' deadlines for a feasible behavior of the real-time system. As presented in figure 4.5, the first step is computing the deadlines of each task in all its related implementations. Then, the second step is selecting the maximum value from these calculated deadlines as the relative deadline of the considered task.

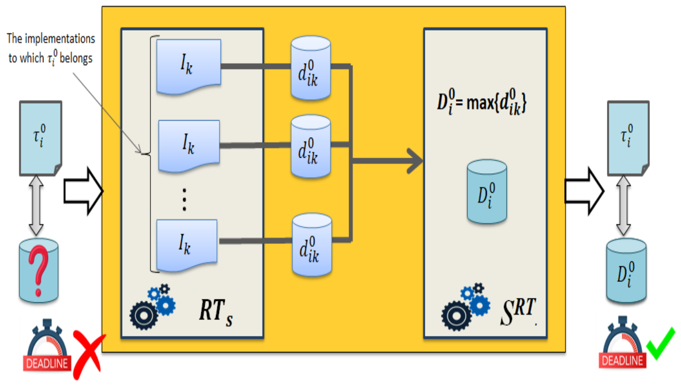

The implementations to which τ_i^0 belongs
τ_i^0
I_k
d_{ik}^0
RT_s
$D_i^0 = \max\{d_{ik}^0\}$
D_i^0
S^{RT}
DEADLINE

Figure 4.5: Real-time Based Deadline Computing Solution.

Function RT_S

The function serves to resolve the temporal constraint on reconfigurable real-time systems. We can denote that they are points in common with the previous contribution proposed in chapter 3. In fact, the second step of the first contribution consists of computing deadlines of periodic and sporadic tasks while considering aperiodic tasks' invocations. Its basic idea is predicting mathematically the maximum cumulative amount of time that has to be executed before a given task. However, this second contribution has different challenges, as it works on a reconfigurable real-time system, i.e., many implementations, and considers only periodic tasks. Thus, to compute the deadlines of a given task in its related implementations, we follow the same steps except that we consider only periodic tasks.

The first step is determining the deadlines of each task jobs τ_{ijk} in each related in implementation I_k. Thus, for each job τ_{ijk}, we compute the maximum cumulative execution time requested by jobs in implementation I_k that may be executed before it, denoted Δ_{ijk}.

Δ_{ijk} includes three factors:

- Δ_{ijk}^0: which is the maximum cumulative execution time requested by the other instances of task τ_{ik}^0 that have to be executed before jthjob τ_{ijk}^0 in the implementation I_k. Thus, we need to determine the number of these instances and then multiply it by the execution time as presented in 4.4:

$$\Delta_{ijk}^0 = (j-1) \times C_i^0 \tag{4.4}$$

- Δ_{ijk}^1: which is maximum cumulative execution time requested by jobs in the implementation I_k having maximum absolute deadlines less than that of job

τ^0_{ijk} or having maximum absolute deadlines equal to that of job τ^0_{ijk} and arrival times less than that of job τ^0_{ijk}. Δ^1_{ijk} is given by:

$$\Delta^1_{ijk} = \sum_{\tau^0_{lk} \in \mathcal{I}_{\|}} (C^0_l \times \beta_{\tau^0_{lk}}) \tag{4.5}$$

where,

$\beta_{\tau^0_{lk}}$ is the number of jobs produced by tasks τ^0_{lk} having maximum absolute deadlines and arrival time equal to those of job τ^0_{ijk}, represented as

$$\beta_{\tau^0_{lk}} = \left\lceil \frac{(j-1)P^0_i + Dmax^0_i - Dmax^0_l}{P^0_l} \right\rceil \tag{4.6}$$

- Δ^2_{ijk}: which is maximum cumulative execution time requested by jobs in the implementaion I_k having maximum absolute deadlines equal to that of job τ^0_{ijk} and arrival times equal to that of job τ^0_{ijk} and their indices are less than i, i.e., in case of equality we will refer to apply the strategy of first in first out (FIFO) by assuming that the task with the smallest index is executed at first. Δ^2_{ijk} is given by:

$$\Delta^1_{ijk} = \sum_{\tau^0_{lk} \in \mathcal{I}_{\|}} (C^0_l \times \beta_{\tau^0_{lk}}) \tag{4.7}$$

where,

$\beta_{\tau^0_{lk}}$ is the number of jobs produced by task τ^0_{lk}, in implementation I_k, having maximum absolute deadlines equal to that of job τ^0_{ijk} and arrival times equal to that of job τ^0_{ijk} and their indices are less than i, represented as

$$\beta_{\tau^0_{lk}} = \left\lceil \frac{(j-1)P^0_i + Dmax^0_i - Dmax^0_l}{P^0_l} \right\rceil + 1 \tag{4.8}$$

Note: jobs having maximum absolute deadlines greater than that of job τ^0_{ijk} are executed after τ^0_{ijk}, i.e., we do not consider their cumulative execution time.

Thus, the maximum cumulative execution time Δ_{ijk} requested by other jobsin implementation I_k, at runtime, that may be executed before τ^0_{ijk} is given by:

$$\Delta_{ijk} = \Delta^0_{ijk} + \Delta^1_{ijk} + \Delta^2_{ijk} \tag{4.9}$$

After computing the maximum cumulative execution time Δ_{ijk} to be performed before the execution of τ^0_{ijk} in the related implementation I_k, we proceed to calculate its deadline d^0_{ijk}. If Δ_{ijk} is achieved before the activation of τ^0_{ijk}, i.e., $\Delta_{ijk} < r^0_{ij}$, then as soon as it is active, τ^0_{ijk} starts its execution. Otherwise, τ^0_{ijk} is executed after a delay equal to $\Delta_{ij} - r^0_{ij}$,i.e., the remained cumulative execution time of other jobs to be executed after r^0_{ij}. Thus, the value d^0_{ijk} that guarantees the feasibility of τ^0_{ijk} takes the form

$$d^0_{ijk} = \begin{cases} \Delta_{ijk} - r^0_{ij} C^0_i + \\ \text{ if } \Delta_{ijk} > r^0_{ij}, \\ C^0_i \text{ else.} \end{cases} \tag{4.10}$$

For each task τ^0_{ik} and after calculating all the d^0_{ijk} of all jobs τ^0_{ijk} on the hyper-period HP_k of each implementation I_k, we propose the following formula to compute d^0_{ik} that guarantees the feasibility of τ^0_{ik} in the related implementation I_k.

$$d^0_{ik} = max\{d^0_{ijk}\} \tag{4.11}$$

Function S^{RT}

After executing RT_S, the function S^{RT} is executed to calculate D^0_i of task τ^0_i. This function is used to calculate D^0_i according to the solution executed before it. This function starts by assigning to each D^0_i the maximum value among d^0_{ik}, i.e.,

$$D^0_i = max\{d^0_{ik}\} \tag{4.12}$$

Finally, D^0_i is the fixed deadline for τ^0_i in all the related implementations.

4.4.3 Energy-Based Deadline Computing with no Starvation Solution

The main aim of this solution is to cope with energy requirements. As presented in figure 4.6, the solution takes as input the tasks with their deadlines outputted from the first solution, i.e., the deadlines ensuring real-time feasibility. The first step is computing the maximum portion of time needed by task jobs in the hyper-period of any related implementation to prevent energy starvation. Then, the second step selects the maximum value from these calculated potion of time to add it to the tasks' deadlines.

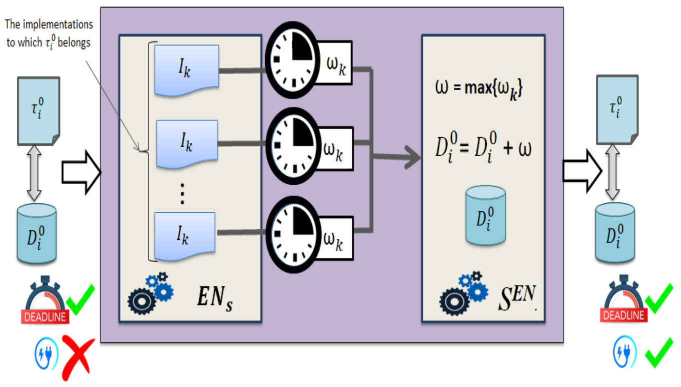


Figure 4.6: Energy-Based Deadline Computing with no Starvation.

To calculate of effective deadlines to be respected by tasks under energy constraints, this solution is based on predicting mathematically the maximum cumulative energy that has to be consumed by the jobs. It considers two functions, EN_S and S^{EN}.

Function EN_S
Let Δ^E_{ijk} be the maximum cumulative energy requested by the jobs in implementation I_k whose (i) absolute deadlines, calculated by Equation (4.12), are less than that of job τ^0_{ijk}. This condition is denoted by D_{L2}, or (ii) absolute deadlines, calculated in Equation (4.12), are equal to that of job τ^0_{ijk} and their arrival times are less than that of job τ^0_{ijk} or their indices are less than i, i.e., we apply the strategy of first in first out (FIFO) by assuming that the task with the smallest index is the one that comes at the beginning. This condition is denoted by D_{E2}, i.e.,

$$\Delta^E_{ijk} = \sum_{\tau^0_{lk} \in I_k} (E_l \times \delta^{ijk}_{lk}) \tag{4.13}$$

where δ^{ijk}_{lk} is the number of jobs produced by task τ^0_{lk} in implementation I_k that have to be executed before τ^0_{ijk} according to their calculated deadlines in RT_S. It verifies one of the following cases given by

$$\delta^{ijk}_{lk} = \begin{cases} \left\lceil \dfrac{(j-1)P^0_i + D^0_i - D^0_l}{P^0_l} \right\rceil \text{ if } (D_{L2} = \text{true} \\ \text{and } D_{E2} = \text{false},) \\ \\ \left\lceil \dfrac{(j-1)P^0_i + D^0_i - D^0_l}{P^0_l} \right\rceil + 1 \\ \text{if } (D_{L2} = \text{true and } D_{E2} = \text{true}), \\ \\ 1 \text{ if } (D_{L2} = \text{false and } D_{E2} = \text{true}), \\ \\ 0 \text{ else.} \end{cases} \tag{4.14}$$

To calculate $E^p_{ijk}(0, f^0_{ijk})$ and $E^c_{ijk}(0, f^0_{ijk})$, we use respectively the following formulas.

$$\begin{aligned} E^p_{ijk}(0, f^0_{ijk}) &= (E^u - \frac{E_a(0)}{HP_k}) \times f^0_{ijk} \\ &= (E^u - \frac{E_a(0)}{HP_k}) \times (C_i + \sum_{\tau_{lk} \in I_k} (C_l \times \delta^{ijk}_{lk})) \end{aligned} \tag{4.15}$$

In Eq. 4.15, $\frac{E_a(0)}{HP_k}$ has been subtracted from E^u to reserve the same amount of energy at the beginning of the next hyper-period.

$$E^c_{ijk}(0, f^0_{ijk}) = \Delta^E_{ijk} + E_i \tag{4.16}$$

Let ω_{ijk} be a portion of time to be added to job τ_{ijk} to prevent the energy starvation. During this portion of time, the processor will be idle, i.e., the energy is produced without any consumption. ω_{ijk} is given by

$$\omega_{ijk} = \begin{cases} \left\lceil \dfrac{E^c_{ijk}(0, f^0_{ijk}) - (E^p_{ijk}(0, f^0_{ijk}) + E_a(0))}{E^u - \frac{E_a(0)}{HP_k}} \right\rceil \\ \text{if } E^c_{ijk}(0, \phi_{ijk}) > E^p_{ijk}(0, f^0_{ijk}) + E_a(0), \\ 0 \text{ else.} \end{cases} \tag{4.17}$$

Then, we propose the following formula to compute ω_{ik} which is the maximum of ω_{ijk}, i.e., the portion of time needed to produce the energy requirements of task τ^0_{ik} in all its related implementations.

$$\omega_{ik} = max\{\omega_{ijk}\} \tag{4.18}$$

To reach the portion of time that prevent energy starvation in all implementations, we propose this formula.

$$\omega_k = max\{\omega_{ik}\} \tag{4.19}$$

Function S^{EN}
After executing EN_S, the function S^{EN} is executed to calculate D^0_i to be respected by tasks under energy constraints. This function starts by adding to each D^0_i the maximum value among ω_k, i.e.,

$$D^0_i := D^0_i + \omega \tag{4.20}$$

where $\omega = max\{\omega_k\}$.
Finally, D^0_i is the fixed task deadline in all the related implementations.

4.4.4 Resource Sharing Based Deadline Computing Solution

The main aim of this solution is to cope with the resources sharing constraint. As presented in figure 4.7, the solution takes as input the tasks with their deadlines outputted from the second solution if the system needs to cope with energy requirements or the first solution overview. The solution is based on computing the maximum blocking time that may delay a task due to the use of a shared resource and adding it to its deadline outputted.

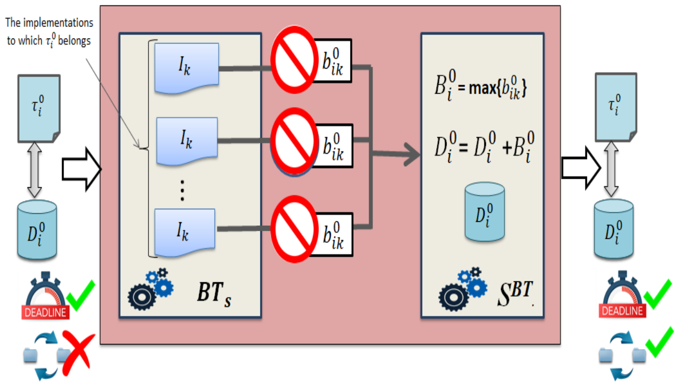


Figure 4.7: Resource Sharing Based Deadline Computing Solution.

This solution calculates the deadlines to be respected by tasks under resource sharing constraints while executing BT_S and S^{BT}.

Function BT_s

Let b^0_{ik} be the blocking time of task τ^0_{ik} resulting from the resource sharing in implementation I_k. It is given by

$$b^0_{ik} = \sum_{\tau^0_{lk} \in I_k} (C^0_l - 1) \times \sigma^{ik}_l \tag{4.21}$$

where

$$\sigma^{ik}_l = \begin{cases} 1 \text{ if } \tau^0_{ik} \text{ and } \tau^0_{lk} \text{ have a common resource and } i \neq l, \\ 0 \text{ else.} \end{cases} \tag{4.22}$$

Function S^{BT}

After executing BT_S, the function S^{BT} is executed to calculate D^0_i to be respected by task τ_i under resource sharing constraint. This function starts by adding to each D^0_i the maximum blocking time, i.e.,

$$D^0_i := D^0_i + B^0_i \tag{4.23}$$

where B^0_i is the blocking time of task τ^0_i,

$$B^0_i = max\{b^0_{ik}\} \tag{4.24}$$

Theorem: Let Π be a reconfigurable real-time system powered by a renewable energy source and uses shared resources, $\mathcal{P} = \{\tau^0_1, ..., \tau^0_n\}$ is the set of all periodic software tasks that can implement Π. By applying successively $(RT_S + S^{RT})$, $(EN_S + S^{EN})$ and $(BT_S + S^{BT})$, the inequality below will be verified

$$f^0_{ijk} \leq r^0_{ij} + D^0_i$$

i.e., each job τ^0_{ijk} completes before the related absolute deadline.

Proof. Let Q_{ijk} be the job quantity to be executed after r_{ij}^0 and before the execution of τ_{ijk}^0 expressed per unit of time, i.e., the cumulative execution time requested by jobs such that their priorities are higher than the priority of τ_{ijk}^0, and their execution will be performed after r_{ij}^0.

We assume that τ_{ik}^0 violates the related real-time constraint even we apply the proposed methodology, i.e., there exists job τ_{ijk}^0 such that $f_{ijk}^0 > r_{ij}^0 + d_{ik}^0$. In this case, f_{ijk}^0 is calculated as follows: $f_{ijk}^0 = r_{ij}^0 + Q_{ijk} + C_i^0$. Since $d_{ik}^0 = max\{d_{ijk}^0\}$, $r_{ij}^0 + Q_{ijk} + C_i^0 > r_{ij}^0 + d_{ijk}^0$ if

$$\boxed{C_i^0 + Q_{ijk} > d_{ijk}^0} \text{ (a)}$$

According to Eq. (4.10), if $\Delta_{ijk} > r_{ij}^0$, we have

$$\boxed{d_{ijk}^0 = C_i^0 + \Delta_{ijk}^0 - r_{ij}^0} \text{ (b)}$$

Since Q_{ijk} is the work quantity to be executed after r_{ij}^0 and before the execution of τ_{ijk}^0, and Δ_{ijk} is the job quantity to be executed from r_{i1}^0 until the instant before the execution of τ_{ijk}^0, $Q_{ijk} = \Delta_{ijk} - (r_{ij}^0 - r_{i1}^0)$. Since all the tasks are activated at $t = 0$, $r_{i1}^0 = 0$. Thus,

$$\boxed{Q_{ijk} = \Delta_{ijk} - r_{ij}^0} \text{ (c)}$$

By (a), (b) and (c), we have $C_i^0 + \Delta_{ijk} - r_{ij}^0 > C_i^0 + \Delta_{ijk} - r_{ij^0} \Leftrightarrow C_i^0 > C_i^0$, absurd. Therefore, $f_{ijk}^0 < r_{ij}^0 + d_{ik}^0$.
According to Eq. (4.10), if $\Delta_{ijk} < r_{ij}^0$, we have

$$\boxed{Q_{ijk} = 0} \text{ (d)}$$

and,

$$\boxed{d_{ijk}^0 = C_i^0} \text{ (e)}$$

By (a), (d) and (e), we have $C_i^0 + 0 > C_i^0 \Leftrightarrow C_i^0 > C_i^0$, absurd.
Then, $f_{ijk}^0 < r_{ij}^0 + d_{ik}^0$. We have $D_i^0 = max\{d_{ik}^0\}$, then $f_{ijk}^0 < r_{ij}^0 + D_i^0$.

We assume that τ_{ik}^0 violates the energy constraint, i.e., there exists job τ_{ijk}^0 such that $r_{ij}^0 + Q_{ijk} + C_i^0 + \omega > r_{ij}^0 + D_i^0$. In this case, f_{ijk}^0 is calculated as follows:

$$\boxed{f_{ijk}^0 = r_{ij}^0 + Q_{ijk} + C_i^0 + \omega} \text{ (f)}$$

Since D_i is the sum of D_i^0 of the first solution and ω,

$$\boxed{D_i^0 := D_i^0 + \omega} \text{ (g)}$$

We have (from the observations previously collected)

$$\boxed{r_{ij} + Q_{ijk} + C_i^0 < r_{ij}^0 + D_i^0} \text{ (h)}$$

By (g) and (h), we have

$$\boxed{r_{ij}^0 + Q_{ijk} + C_i^0 + \omega < r_{ij}^0 + D_i^0} \text{ (i)}$$

By (f) and (i), we have $f_{ijk}^0 < r_{ij}^0 + D_i^0$.

We assume that τ_{ik}^0 violates the resource sharing constraint, i.e., there exists job τ_{ijk}^0 such that $r_{ij}^0 + Q_{ijk} + C_i^0 + b_{ik}^0 > r_{ij}^0 + D_i^0$. In this case, f_{ijk}^0 is calculated as follows:

$$\boxed{f_{ijk}^0 = r_{ij}^0 + Q_{ijk} + C_i^0 + b_{ik}^0} \text{ (j)}$$

Since D_i^0 is the sum of D_i^0 of the first solution and B_i^0,

$$\boxed{D_i^0 := D_i^0 + B_i^0} \text{ (k)}$$

We have (from the observations previously collected)

$$\boxed{r_{ij}^0 + Q_{ijk} + C_i^0 < r_{ij}^0 + D_i^0} \text{ (l)}$$

By (k) and (l), we have $r_{ij}^0 + Q_{ijk} + C_i^0 + B_i^0 < r_{ij}^0 + D_i^0$
As $B_i^0 = max\{b_{ik}^0\}$, then

$$\boxed{r_{ij}^0 + Q_{ijk} + C_i^0 + b_{ik}^0 < r_{ij}^0 + D_i^0} \text{ (m)}$$

By (j) and (m), we have $f_{ijk}^0 < r_{ij}^0 + D_i^0$.

We conclude that $f_{ijk}^0 \leq r_{ij}^0 + D_i^0$. □

4.5 Simulation and Conducted Experimentation

4.5.1 Developed Environment

In order to bring out the importance of the proposed approach and to test it, we have implemented the proposed methodology in a graphical simulator named DEAD-CALC, presented in figure 4.8. This tool is an open source environment that supports and implements the three proposed solutions. More details about our tool are presented on this website https://projects-lisi-lab.wixsite.com/deadcalc.

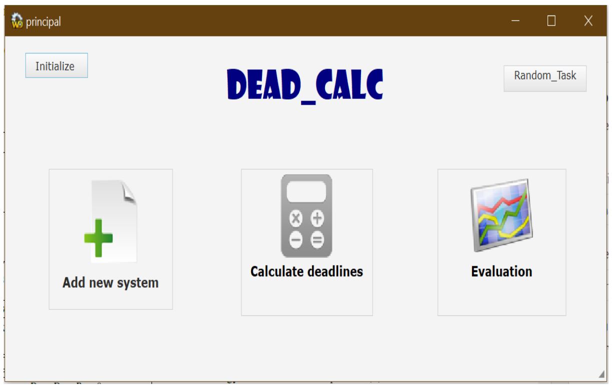
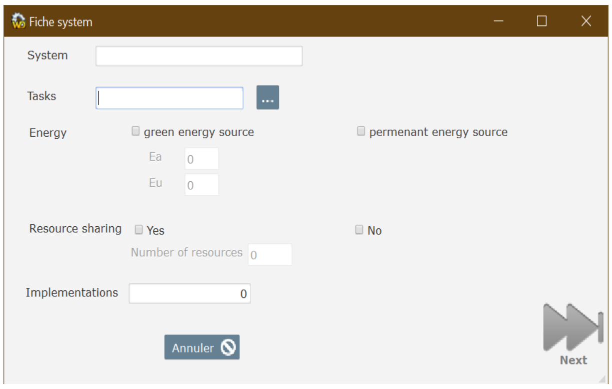


Figure 4.8: The main interface of DEAD-CALC.

Through the main interface, the user have to access to:

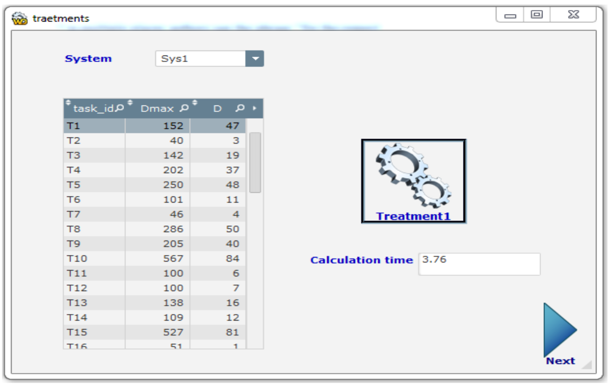
- The form, presented in figure 4.9, for adding a new system in which the user fills in the system parameters and constraints.



Ea 0
Eu 0
Number of resources 0
Implementations 0

Figure 4.9: The form to add new system.

- The interfaces that calculates the deadlines of tasks by applying the formulas of the proposed methodology. For example, figure 4.10 shows the interface of applying the first solution that serves to resolve real-time requirements. It shows the deadlines calculation time that reflects the speed of computing the deadlines.



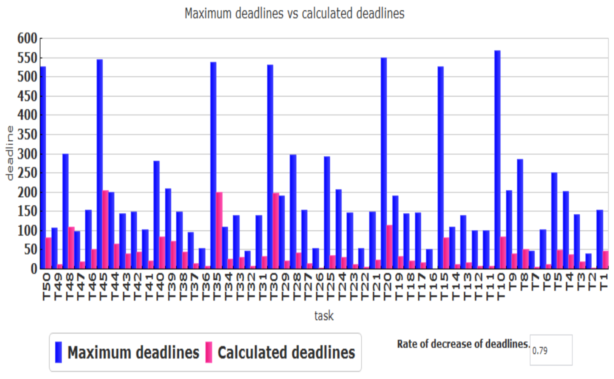
System Sys1

task_id	Dmax	D
T1	152	47
T2	40	3
T3	142	19
T4	202	37
T5	250	48
T6	101	11
T7	46	4
T8	286	50
T9	205	40
T10	567	84
T11	100	6
T12	100	7
T13	138	16
T14	109	12
T15	527	81
T16	51	1

Calculation time 3.76

Figure 4.10: Example of the first treatment.

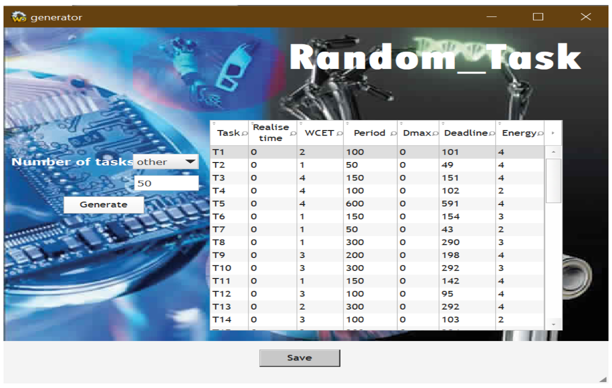
- The interface of evaluation. In fact, this tool allows to generate a histogram that reflects the difference between the maximum deadlines and the calculated effective deadlines, and shows the decreased rate of deadlines, as presented in figure 4.11.



Rate of decrease of deadlines: 0.79

Figure 4.11: Example of a generated histogram.

- The tool RANDOM-TASK, presented in figure 4.12, is a random tasks generator. It allows the designer to fill in the desired number of tasks and generates a random list of tasks after clicking on the button Generate. The list of tasks can be saved as Excel file by clicking the button Save.



Number of tasks: other

50

Task	Realise time	WCET	Period	Dmax	Deadline	Energy
T1	0	2	100	0	101	4
T2	0	1	50	0	49	4
T3	0	4	150	0	151	4
T4	0	4	100	0	102	2
T5	0	4	600	0	591	4
T6	0	1	150	0	154	3
T7	0	1	50	0	43	2
T8	0	1	300	0	290	3
T9	0	3	200	0	198	4
T10	0	3	300	0	292	3
T11	0	1	150	0	142	4
T12	0	3	100	0	95	4
T13	0	2	300	0	292	4
T14	0	3	100	0	103	2

Figure 4.12: RANDOM-TASK generator tool.

4.5.2 Case Study

We present in this section a case study to evalute and test the proposed contribution. We consider the chocolate production line system. We assume that the chocolate factory is powered by the energy collect by photovoltaic solar panels, as shown in figure 4.13. A second case study is detailed in our website https://projects-lisi-lab.wixsite.com/deadcalc/simple-case-study.

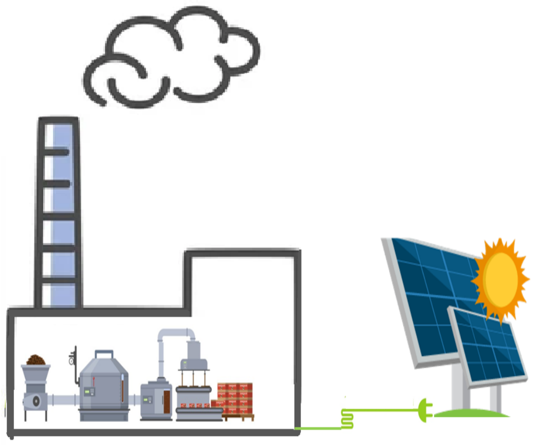


Figure 4.13: Case study modelisation.

The chocolate production line system, denoted Π, is implemented with the following tasks:

- τ_1^0 serves to dose the chocolate to have the right amount of chocolate in the molds.
- τ_1^0 serves to transfer the molds in moving frames to ensure continuous line production.
- τ_1^0 serves to control the amount of chocolate in the tank.

- τ_1^0 serves to fill the chocolate tank if the amount of chocolate in the tank falls below the threshold value.

Therefore,Π is implemented by four tasks presented in table 4.1, $\mathcal{P} = \{\tau_1^0, \tau_2^0, \tau_3^0, \tau_4^0\}$.

Table 4.1: System tasks

Task	R_i^0	C_i^0	P_i^0	$Dmax_i^0$	E_i^0
τ_1^0	0	4	20	18	1
τ_2^0	0	3	20	20	1
τ_3^0	0	1	10	8	1
τ_4^0	0	3	10	12	3

Moreover, this system has a shared memory Rs_1 used by τ_2^0 and τ_3^0 for data storage, i.e., $\mathbb{R} = \{SR_1\}$.

On the other hand, this system has two implementations, $\mathbb{I} = \{I_1, I_2\}$:

- I_1 is executed when the amount of chocolate in the tank is greater than the threshold value. Thus, there is no need to execute τ_4^0,i.e., $I_1 = \{\tau_{11}^0, \tau_{21}^0, \tau_{31}^0\}$.
- I_2 is executed when the amount of chocolate in the tank is less than the threshold value. In this case, there is a need to fill the tank with chocolate,i.e., $I_2 = \{\tau_{12}^0, \tau_{22}^0, \tau_{32}^0, \tau_{42}^0, \}$.

The factory is powred by energy solar with a lowest instantaneous charging rate is equal to $0, 6j$, i.e., $E^u = 0, 6j$. Moreover, we assume that the aivlebele energy in the battery when the system turned on is equel to $1j$,i.e, $E_a = 1j$.

4.5.2.1 Application of the Proposed Approach

In the proposed case study, the chocolate production line system is powered by solar energy, and it uses shared memory for data storage. In other words, this system deals with real-time, energy, and resource sharing constraints. Thus, to ensure the system's feasibility, we have to apply the proposed approach in the fourth branch. As presented in figure 4.14, we need to execute successively, Real-time based deadline computing solution, Energy-based deadline computing with no starvation solution, and Resource sharing based deadline computing solution.

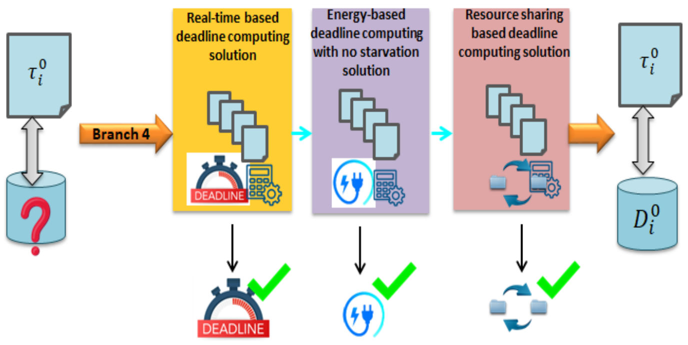

τ_i^0
Branch 4
Real-time based deadline computing solution
DEADLINE
Energy-based deadline computing with no starvation solution
Resource sharing based deadline computing solution
τ_i^0
D_i^0
DEADLINE

Figure 4.14: Steps of branch 4.

Real-time based deadline computing solution:
RT_S **execution:**
As first example, we present the deadline computing of task τ_3^0.
This task belongs I_1 (it will be noted τ_{31}), and I_2 (it will be noted τ_{32}).

We start with the first implementation, $I_1 = \tau_{11}, \tau_{21}, \tau_{31}$. We have $HP_1 = LCM 20, 20, 10 = 20$. Thus, τ_{31} has $HP_1/P_3^0 = 20/(10) = 2 jobs$ in the hyper-period HP_1.

Job τ_{311}^0: First, we compute the maximum cumulative execution time all jobs that may occur before this job,i.e., Δ_{311}.
According to Equation (4.9), we have to calculate Δ_{311}^0, and Δ_{311}^1 and Δ_{311}^2:
According to Equation (4.4): $\Delta_{311}^0 = (1-1) \times 1 = 0$
According to Equation (4.5): $\Delta_{311}^1 = 0$ because the jobs τ_{111}^0, and τ_{211}^0 have absolute deadlines greater than that of τ_{311}^0.
According to Equation (4.7): $\Delta_{311}^2 = 0$ because there is no job having a maximum absolute deadline and arrival time equal to that of job τ_{311}^0.
Thus, $\Delta_{311} = 0$.
We have $r_{311}^0 = 0$, so $\Delta_{311} = r_{311}^0$. Thus, according to Equation (4.10),

$$d_{311}^0 = C_3^0 = 1$$

Job τ_{321}^0: First, we compute the maximum cumulative execution time all jobs that may occur before this job,i.e., Δ_{321}.
According to Equation (4.9), we have to calculate Δ_{321}^0, and Δ_{321}^1 and Δ_{321}^2:
According to Equation (4.4): $\Delta_{321}^0 = (2-1) \times 1 = 1$
According to Equation (4.5): $\Delta_{321}^1 = 0$
According to Equation (4.7): $\Delta_{31}^2 = 0$ because there is no job having a maximum absolute deadline and arrival time equal to that of job τ_{321}^0.
Thus, $\Delta_{321} = 5$.
We have $r_{321}^0 = 10$, so $\Delta_{321} < r_{321}^0$. Thus, according to Equation (4.10),

$$d_{321}^0 = C_3^0 = 1$$

According to Equation (4.11), the deadline d^0_{31} that guarantees the feasibility of τ^0_{31} in the related implementation I_1 is equal to,

$$d^0_{31} = max\{d^0_{311}, d^0_{321}\} = 1$$

Then, in the second implementation, $I_2 = \tau_{12}, \tau_{22}, \tau_{32}, \tau_{42}$. We have $HP_2 = LCM 20, 20, 10, 10 = 20$. Thus, τ_{32} has $HP_2/P^0_3 = 20/(10) = 2 jobs$ in the hyper-period HP_2.

Job τ^0_{312}: First, we compute the maximum cumulative execution time all jobs that may occur before this job,i.e., Δ_{312}.
According to Equation (4.9), we have to calculate Δ^0_{312}, and Δ^1_{312} and Δ^2_{312}:
According to Equation (4.4): $\Delta^0_{312} = (1 - 1) \times 1 = 0$
According to Equation (4.5): $\Delta^1_{312} = 0$ because the job τ^0_{112}, τ^0_{212}, and τ^0_{412} have absolute deadlines greater than that of τ^0_{312}.
According to Equation (4.7): $\Delta^2_{312} = 0$ because there is no job having a maximum absolute deadline and arrival time equal to that of job τ^0_{312}.
Thus, $\Delta_{312} = 0$.
We have $r^0_{312} = 0$, so $\Delta_{312} = r^0_{312}$. Thus, according to Equation (4.10),

$$d^0_{312} = C^0_3 = 1$$

Job τ^0_{322}: First, we compute the maximum cumulative execution time all jobs that may occur before this job,i.e., Δ_{322}.
According to Equation (4.9), we have to calculate Δ^0_{322}, and Δ^1_{322} and Δ^2_{322}:
According to Equation (4.4): $\Delta^0_{322} = (2 - 1) \times 1 = 1$
According to Equation (4.5): $\Delta^1_{322} = C^0_4 \times 1 = 4$
According to Equation (4.7): $\Delta^2_{322} = 0$ because there is no job having a maximum absolute deadline and arrival time equal to that of job τ^0_{322}.
Thus, $\Delta_{322} = 5$.
We have $r^0_{322} = 10$, so $\Delta_{322} < r^0_{322}$. Thus, according to Equation (4.10),

$$d^0_{322} = C^0_3 = 1$$

According to Equation (4.11), the deadline d^0_{32} that guarantees the feasibility of τ^0_{32} in the related implementation I_2 is equal to,

$$d^0_{32} = max\{d^0_{312}, d^0_{322}\} = 1$$

As second example, we present the deadline computing of task τ^0_4.
This task belongs I_2 (it will be noted τ_{42}).
In the implementation, $I_2 = \tau_{12}, \tau_{22}, \tau_{32}, \tau_{42}$. We have $HP_2 = LCM 20, 20, 10, 10 = 20$. Thus, τ_{42} has $HP_2/P^0_4 = 20/(10) = 2 jobs$ in the hyper-period HP_2.

Job τ^0_{412}: First, we compute the maximum cumulative execution time all jobs that may occur before this job,i.e., Δ_{412}.
According to Equation (4.9), we have to calculate Δ^0_{412}, and Δ^1_{412} and Δ^2_{412}:
According to Equation (4.4): $\Delta^0_{412} = (1 - 1) \times 3 = 0$
According to Equation (4.5): $\Delta^1_{412} = C^0_3 \times 1 = 1$

According to Equation (4.7): $\Delta^2_{412} = 0$ because there is no job having a maximum absolute deadline and arrival time equal to that of job τ^0_{412}.
Thus, $\Delta_{412} = 1$.
We have $r^0_{412} = 0$, so $\Delta_{412} > r^0_{412}$. Thus, according to Equation (4.10),

$$d^0_{412} = C^0_3 + \Delta_{412} - r_{412} = 4$$

Job τ^0_{422}: First, we compute the maximum cumulative execution time all jobs that may occur before this job,i.e., Δ_{422}.
According to Equation (4.9), we have to calculate Δ^0_{422}, and Δ^1_{422} and Δ^2_{422}:
According to Equation (4.4): $\Delta^0_{322} = (2-1) \times 3 = 3$
According to Equation (4.5): $\Delta^1_{422} = C^0_1 \times 1 + C^0_2 \times + C^0_3 \times 2 = 9$
According to Equation (4.7): $\Delta^2_{422} = 0$
Thus, $\Delta_{422} = 12$.
We have $r^0_{422} = 10$, so $\Delta_{422} > r^0_{422}$. Thus, according to Equation (4.10),

$$d^0_{422} = C^0_4 + \Delta_{422} - r_{422} = 5$$

According to Equation (4.11), the deadline d^0_{42} that guarantees the feasibility of τ^0_{42} in the related implementation I_2 is equal to,

$$d^0_{42} = max\{d^0_{412}, d^0_{422}\} = 5$$

S^{RT} execution:
SRT execution: This function is executed to compute D^0_1, D^0_2, D^0_3, and D^0_4:

$$D^0_3 = max\{d^0_{31}, d^0_{32}\} = 1$$

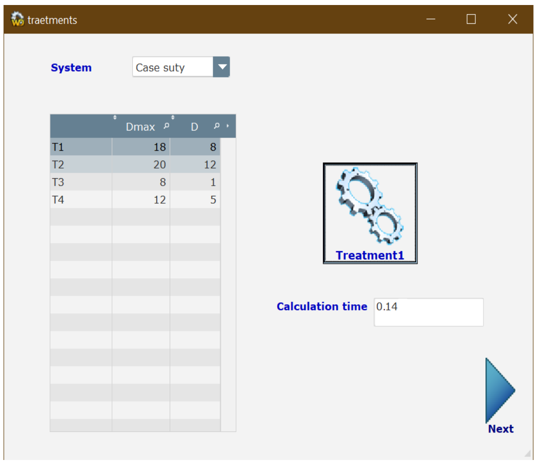
$$d^0_4 = max\{d_{42}\} = 5$$

While applying the same steps of RT_s function of τ^0_1, τ^0_2, we have:

$$D^0_1 = max\{d^0_{11}, d^0_{12}\} = 8$$

$$D^0_2 = max\{d^0_{21}, d^0_{22}\} = 12$$

Figure4.15 illustrates the results of the case study by using DEAD-CALC tool.



System Case suty

	Dmax	D
T1	18	8
T2	20	12
T3	8	1
T4	12	5

Calculation time 0.14

Figure 4.15: First solution results.

Energy-Based Deadline Computing with no Starvation Solution:
This solution is executed to cope with energy requirements. It takes as input the tasks with their deadlines outputted from the first solution.

EN_S **execution:**
This function serves to compute the maximum portion of time needed by task jobs in the hyper-period of any related implementation to prevent energy starvation.

We take as exemple task τ_{42}^0 in the second implementation I_2. We have $HP_2 = LCM 20, 20, 10, 10 = 20$. This task has two jobs in the hyper-period HP_2.
Job $\underline{\tau_{412}^0}$**:**
First of all, we calculate Δ_{412}^E. According to Eq.4.13,

$$\Delta_{412}^E = \sum_{\tau_{l2}^0 \in I_2} (E_l \times \delta_{l2}^{412})$$

Then, we calculate δ_{l2}^{412} as below (Eq.4.14)

$$\delta_{12}^{412} = 0$$

$$\delta_{22}^{412} = 0$$

$$\delta_{32}^{412} = 1$$

$$\delta_{42}^{412} = 0$$

Thus $\Delta_{412}^E = 1 \times 1 = 1$.
After that we calculte,according to Eq.4.15 and Eq.4.16, $E_{412}^p(0, f_{412}^0)$ and $E_{412}^c(0, f_{412}^0)$.

$$E_{412}^p(0, f_{412}^0) = 0,55 \times 4 = 2,2$$

and,

$$E_{412}^c(0, f_{412}^0) = 1 + 3 = 4$$

According to Eq.4.17, we calculate the portion of time to be added to job $\tau 412^0$ to satisfy its energy requirement. We have $E^p_{412}(0, f^0_{412}) + E_a(0) < E^c_{412}(0, f^0_{412})$, then, $\omega_{412} = \left\lceil \frac{4-3,2}{0,55} \right\rceil = 2$
We follow the same approach for the other tasks.

S^{EN} execution:
This function selects the maximum value from these calculated potion of time to add it to the tasks' deadlines.

$\omega = max\{\omega_1, \omega_2\} = 2$

According to (4.20), the tasks'deadlines ensuring real-time, and energy constraints are:

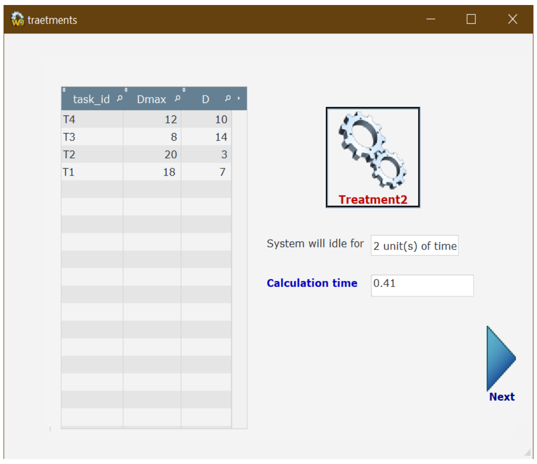
$$D^0_1 := D^0_1 + \omega = 10$$

$$D^0_2 := D^0_2 + \omega = 14$$

$$D^0_3 := D^0_3 + \omega = 3$$

$$D^0_4 := D^0_4 + \omega = 7$$

Figure4.16 illustrates the results of the case study by using DEAD-CALC tool.



task_id	Dmax	D
T4	12	10
T3	8	14
T2	20	3
T1	18	7

System will idle for 2 unit(s) of time

Calculation time 0.41

Figure 4.16: Second solution results.

Resource Sharing based deadline computing:
This solution calculates the deadlines to be respected by tasks under resource sharing constraints.

BT_S execution:
As the shared memory is used by τ^0_2 and τ^0_3, the computing of the blocking time will

be performed in the same way in l_1 and l_2.
According to (4.21),

$$b^0_{21} = b^0_{22} = (C^0_3 - 1) \times 1 = 0$$

$$b^0_{31} = b^0_{32} = (C^0_2 - 1) \times 1 = 2$$

S^{BT} **execution:**

After executing BT_S, this function adds to each deadline, outputted by the second solution, its maximum blocking time. As τ^0_1 and τ^0_4 do not use any shared resource, thus $B^0_1 = B^0_4 = 0$

According to (4.21),

$$B^0_2 = max\{b^0_{21}, b^0_{22}\} = 0$$

$$B^0_3 = max\{b^0_{31}, b^0_{32}\} = 2$$

According to (4.23), the tasks'deadlines ensuring real-time, energy, and resources sharing constraints are:

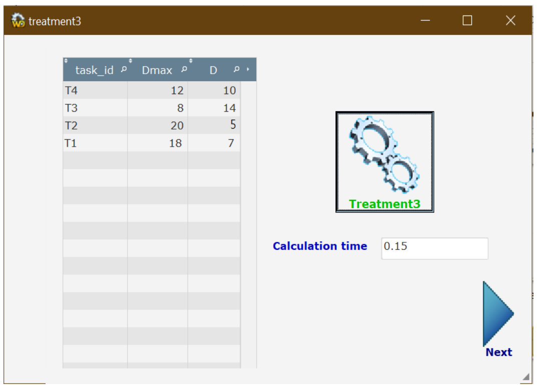
$$D^0_1 := D^0_1 + B^0_1 = 10 + 0 = 10$$

$$D^0_2 := D^0_2 + B^0_2 = 14 + 0 = 14$$

$$D^0_3 := D^0_3 + B^0_3 = 3 + 2 = 5$$

$$D^0_4 := D^0_4 + B^0_4 = 7 + 0 = 7$$

Figure4.17 illustrates the results of the case study by using DEAD-CALC tool.



task_id	Dmax	D
T4	12	10
T3	8	14
T2	20	5
T1	18	7

Calculation time 0.15

Figure 4.17: Third solution results.

4.5.2.2 Case Study Results Evaluation

In order to check the system feasibility with the computed deadlines a scheduling test with EDF algorithm is performed presented in figure 4.18. We note that the real-time constraints are respected by the proposed methodology. Moreover, we note that for each task, there is at least one of its jobs that ends exactly on its deadline. Hence, the efficiency and accuracy of calculated deadlines.

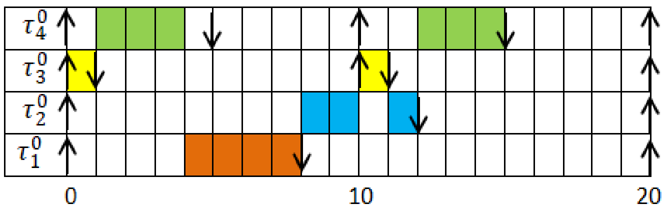


Figure 4.18: Scheduling of implementation I_2 after the execution of the first solution.

Figure 4.19 describes two schedules: (i)The first is done with the deadlines delivered by RT_S (i.e., before the execution of EN_S. We note that there is energy starvation (-1,8) at instant $t = 4$ of time. In fact, the energy necessary to process the job $\tau_4 12^0$ by its deadline is not available. (ii) The second is done with the deadlines delivered by EN_S. We note that From instant $t = 1$ to $t = 3$, the processor is idle for two units of time to allow energy production. Thus, there is any energy starvation and the real-tile correctness is also ensured. Hence, the effectiveness of the proposed approach in respecting both real-time and energy constraints.

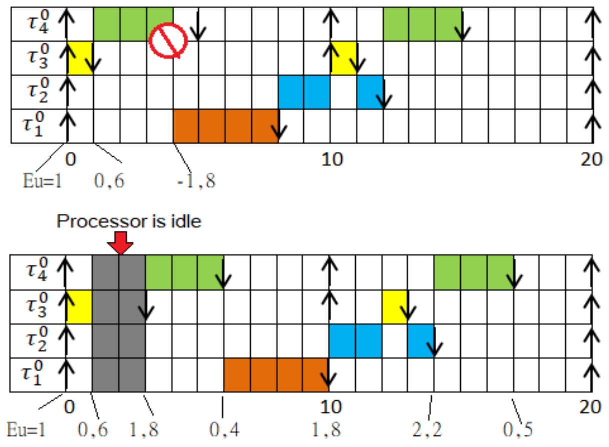


Figure 4.19: Scheduling of implementation I_2 after the execution of the second solution.

Figure 4.20 describes two schedules: (i)The first is done with the deadlines delivered by EN_S (i.e., before the execution of BT_S. We note that job $\tau_3 22^0$ exceeds its deadline, as it has been blocked in instant $t = 12$ waiting for the resource to be released. (ii) The second is done with the deadlines delivered by BT_S. We note that job $\tau_3 22^0$ meets its deadline even though it waited for the resource to be released. Thus, the proposed approach ensures resouce sharing constrants in addtion to real-time and energy requirements.

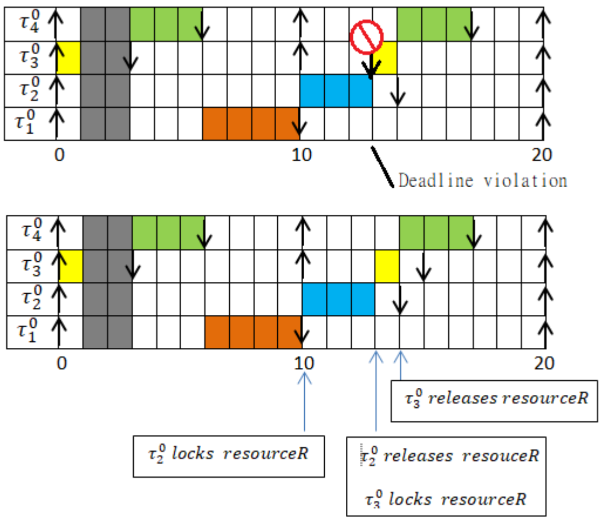

τ_4^0
τ_3^0
τ_2^0
τ_1^0
0
10
20
Deadline violation
τ_3^0 releases resourceR
τ_2^0 locks resourceR
τ_2^0 releases resouceR
τ_3^0 locks resourceR

Figure 4.20: Scheduling of implementation I_2 after the execution of the third solution.

We conclude that the proposed methodology ensure the system feasibility under real-time, energy and resource sharing constraints.

4.6 Discussion

This contribution proposes a new approach to configure feasible reconfigurable real-time system. Compared to the existing researches, that are presented in detail in Chapter 2, the originality of the current work is to not only calculate system deadlines that will certainly be respected online without any additional feasibility analysis of the device, but also to overcome additional challenges such as energy requirements and shared resources constraints whatever system implementation. Moreover, it has introduced software tool implementing the methodology concepts.

DEAD-CALC reduces the development time by computing the deadlines to be certainly respected without any feasibility analysis of the device. If we consider 300 tasks, then identifying effective deadlines will take certainly dozens of hours to be done. We used RANDOM-TASK to generate a set of random tasks, and then we used DEAD-CALC to compute the effective deadlines. Figure 4.21 shows the deadlines calculation time in branches 1, 2, 3 and 4. For example, we present demonstrative videos of generating 5O tasks and applying the four branches of the proposed approach in our website https://projects-lisi-lab.wixsite.com/deadcalc/services.

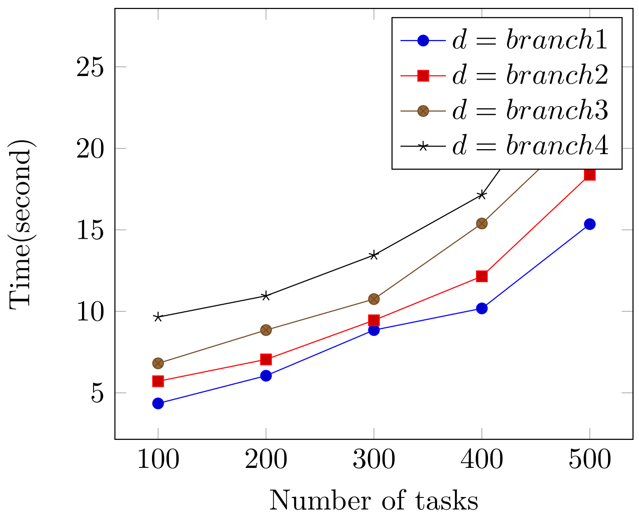


Figure 4.21: Deadlines calculation time in the four branches.

Moreover, DEAD-CALC reduces the development time even if there are several tasks and implementations to work on. Figure 4.22 presents the variation of the deadlines calculation time of the first solution depending on both the number of tasks and implementations.

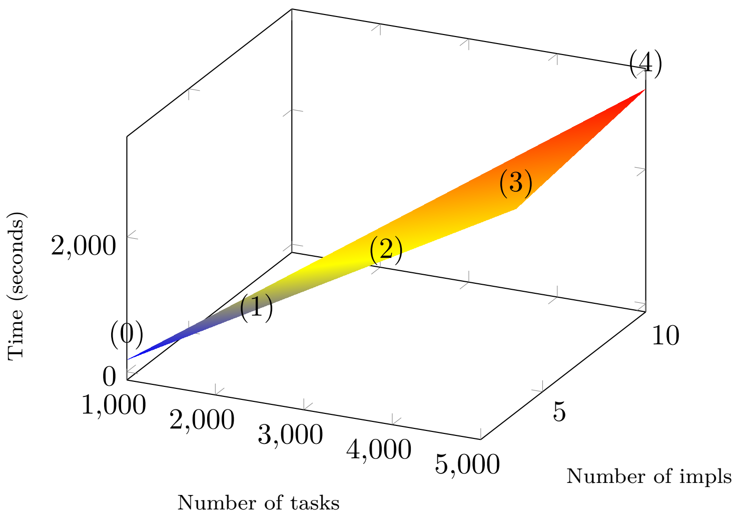


Figure 4.22: Calculation time of the first solution under a huge number of tasks and implementations.

DEAD-CALC reduces the development time by computing the deadlines to be certainly respected without any additional feasibility analysis of the device. The calculation can only consider the real-time aspects, or both the renewable energy and resource sharing constraints. To the best of our knowledge, no one in all related works considers the calculation of deadlines for feasible reconfigurable real-time systems under energy and resource sharing constraints.

4.7 Conclusion

This chapter focuses on a reconfigurable real-time system powered by a renewable energy source and encoded by a set of periodic tasks which may share resources under mono-core architecture. The main aim of the chapter is to calculate the deadlines to be certainly respected under the given constraints, without wasting time performing schedulability tests. This contribution is illustrated through three solutions that are dedicated respectively to overcoming the real-time, energy, and resource sharing challenges. A new visual simulator called DEAD-CALC is developed to be used by designers to compute and display deadlines, with few clicks, in arranged tables, and in a short time. This project can be a future reference for industrial partners who will be focusing on various real-time applications design.
In the next chapter, the proposed approach is focused on multi-core reconfigurable real-time systems.

Chapter 5

Configuring Feasible Reconfigurable Multi-core Real-time System with Energy Harvesting and Precedence Constraint

5.1 Introduction

The previous chapters dedicated to parametrize feasible real-time systems under given constraints in mono-core architecture. This chapter tackles the real-time scheduling problem on multi-core architecture. Moreover, renewable energy challenge and tasks precedence constraints are studied. The tasks are assumed to be mapped to cores statically and not allowed to migrate, and they can be periodic or aperiodic which are invoked to cope with external interruptions. A novel scheduling strategy is proposed to effectively compute deadlines, allowing for tasks and messages to meet related constraints. To cope with the energy availability issue, this method defines different operating modes, each of which is characterized by energy and frequency parameters. Then, this contribution determines deadlines ensuring real-time system feasibility by considering the invocation of aperiodic task execution and task precedence constraints on multi-core architecture.
The outline of the chapter is organized as follows: First, we present the motivation section. Next, we present the computational model and the assumptions based on the model proposed in the first chapter. Then, we explain in details the proposed approach. Last, we illustrates the approach on a case study while evaluating its efficiency. Finally, we present the discussion section to analyze and interpret this chapter's findings.

Note that this chapter has been published in the international journal Information Sciences[41].

5.2 Motivation

Multi-core real-time systems are becoming prevalent across all industries as they can increase performance by running multiple applications concurrently. In addition, the allocation of software to multiple cores also supports failure tolerance by supporting failover from one core to another. On another hand, these system introduce additional challenges such as finding efficient solutions to the task scheduling problem especially when taking into consideration various type of tasks. In fcat, such system is generally specified by a large number of tasks of various type and that may be dependent and need to transfer messages to communicate together when they are portioned in different cores.

On another hand, real-time systems operated by batteries are growing in different application domains. To prolong their lifetime, there is a need to reduce energy consumption. Hence, a new generation of processors [60, 59, 58] is designed to allow a dynamic variation of the voltage and operating frequency to balance computational frequency and energy consumption. Moreover, as presented in figure, renewable energy sources generate most of their energy at certain times of the day depending on environmental conditions, but it does not match the peak demand hours. The intermittency of renewable energy is unpredictable. Hence, it cannot provide an on-demand power source 24 hours a week. Therefore, the problem of allocating appropriate frequencies to the cores of the real-time system must be considered to cope with the uncertain energy availability.

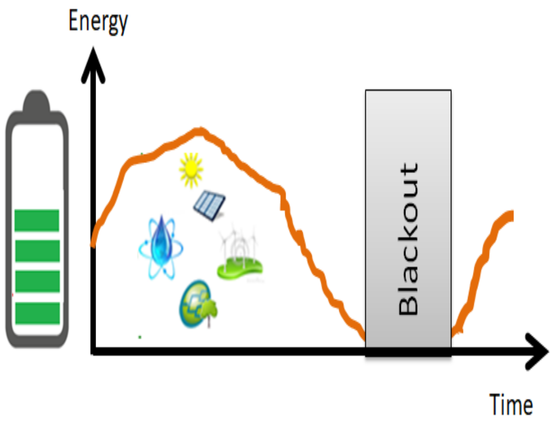


Figure 5.1: Renewabele energy availability.

The problem to be overcome in this chapter can be summarized as parametrizing feasible reconfigurable multi-core real-time systems having both periodic and aperiodic real-time tasks, powered by renewable energy and may have precedence constraints. Even if research results of each topic are separately rich, to our best knowledge there is no work that treats all the mentioned challenges together. From this viewpoint, this contribution proposes a new methodology to configure feasible scheduling of real-time tasks of various type on multi-core architecture while considering tasks dependency and energy harvesting challenges. This appraoch aims to:

- Parameterize operating modes by identifying energy and frequency parameters for each one, this Generates a new implementation of the system having tasks with updated parameters,
- Identify tasks and messages' deadlines which will be certainly respected online,
- Ensure precedence constraints by using calculated deadlines for tasks based on deadlines of their predecessors, i.e., a task must have longer deadline than those of its predecessors if they exist.

5.3 Formalization

In this section, we introduce the mathematical expressions that represents the characteristics of multi-core reconfigurable real-time system. Moreover, we introduce a formal description of the energy model.

5.3.1 System Model

This contribution focuses on multi-core architecture. Hence, a real-time system Π, has an z-core architecture, is denoted as a set of cores $\Pi = \{X_1, X_2, ..., X_z\}$. Moreover, Π consists of periodic and aperiodic tasks which are assigned to x cores . Same as previous contributions, $\mathcal{P}$ is the set of periodic tasks, , $\mathcal{P} = \{\tau_1^0, ..., \tau_n^0\}$ and $\mathcal{A}$ is the set of aperiodic tasks, $\mathcal{A} = \{\tau_1^2, ..., \tau_o^2\}$.

As in this contribution we consider dependent tasks, we must model the set of messages to ensure the communication between the dependent tasks. Thus, we denote by $\mathcal{M} = \{m_1, m_2, ..., m_b\}$ the set of b messages.

5.3.2 Energy Model

Certainly, the amount of harvested energy from a renewable source is not considered to be stable over time due to the variation of environmental conditions, such as the angle of sunlight incidence and cloud density. Thus, in this contribution we choose to work variation of instantaneous charging rate. Let $E = [E^u_{min}, E^u_{max}]$ be the instantaneous charging rate variation E^u_{min} is the minimum instantaneous charging rate produced by the energy source and E^u_{max} is the maximum one. The energy available in an ideal energy battery at $t = 0$ is denoted as E_a. The produced energy in the interval time $[0, t_1]$ is denoted as $E^p(0, t_1)$, where its minimum value is $E^u_{min} \times t_1$, and the maximum one is $E^u_{max} \times t_1$. Moreover, the energy consumed $E^c(0, t_1)$ is the cumulative amount of energy of tasks executed by x cores in $[0, t_1]$. Moreover, each core X_p requires energy, denoted E_p, to execute its tasks on time interval $[0, t_1]$, where $p \in \{1, ..., z\}$.

As mentioned above, the amount of harvested energy from a renewable source is out of control. To overcome this issue, we rely on designating different operating modes, each of which is defined as a time slot characterized by its own energy and frequency parameters. We denote by $\Theta = \{\theta_1, \theta_2, ..., \theta_x\}$ the set of operating modes, each of which is described by two intervals:

- $F^k = [f^k_m, f^k_M]$ is the processor frequency variation in the operating mode θ_k, where $k \in \{1, ..., x\}$.
- $E^k = [E^k_m, E^k_M]$ is the instantaneous charging rate variation in θ_k, in such a way that the difference between minimum instantaneous charging rate and maximum one is almost constant at each operating.

As aforementioned, we focus in this chapter on reconfiguration scenarios that consist on updating tasks parameters. As the execution time of a task is equal to its number of execution cycles divided by the operating frequency [75], then the modification of the frequency from one operating mode to another leads to the modification of the tasks' execution times. Hence, as represented in figure 5.2, each operating mode θ_k leads to new system implementation I_k while tasks execution times are updated. Thus, $\mathcal{I} = \{I_1, ..., I_x\}$ is the set of implementations, where each element I_k contains all tasks in $\mathcal{P}$.

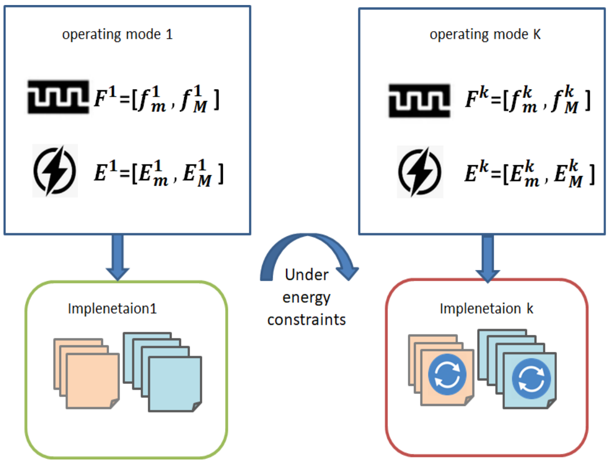


Figure 5.2: Operating mode and system implementations.

5.3.3 Processor Model

We assume in this contribution that processors allow varying the supply voltage and thus the operating frequency. In other words, they support Dynamic voltage and frequency scaling (DVFS), a technique that aims at reducing the dynamic power consumption by dynamically adjusting voltage and frequency of processor. Thus, a processor can dynamically adapt its working frequency in a continuous range $F = [f_m, f_M]$. Thus, in each operating mode θ_k, each core X_p may run at different frequency f^p, where $f^p \in F$.

On another hand, we assume that the interconnection among cores is based on a mesh topology where each core has a communication bus to each of the other cores.

5.3.4 Periodic Task Model

Each periodic task τ_i^0, $i \in \{1, ..., n\}$, in $\mathcal{P}$ is characterized by: i) release time R_i^0, ii) worst-case execution cycle (WCEC) C_i^0, iii) period P_i^0, iv) relative deadline D_i^0 to be calculated for system feasibility, i.e., $D_i^0 \leq P_i^0$, v) required energy E_i^0, and vi) execution time T_i^0 which depends on the frequency of core X_p on which τ_i^0 is running and WCEC. Hence, to compute T_i^0 [75], it is enough to satisfy (1).

$$T_i^0 = \left\lceil \frac{C_i^0}{f^p} \right\rceil \tag{5.1}$$

Also, we prove that the required energy E_i^0 depends on f^p when τ_i^0 is running on X_p. The details of calculating E_i^0 are presented in Proof 1 below.

$$E_i^0 = Cst_3 \times (f^p)^2 \times C_i^0 \tag{5.2}$$

where, Cst_3 is a constant.

Proof 1: Calculation of the required energy by a task
In the liturature [4, 85, 15, 27], the power consumption P is

$$\boxed{P = Cst_1 \times V^2 \times F} \text{ (a)}$$

where Cst_1 is a constant switched capacitance, V is the supply voltage and F is the frequency. Moreover, the frequency F can be expressed in terms of supply voltage V,and threshold voltage Vt as follows [4]:

$$\boxed{F = \frac{Cst_2 \times (V - Vt)^2}{V} \text{ (c)}}$$

We neglect Vt compared to V, i.e., $V - Vt \approx 0$, then formula (c) becomes

$$\boxed{F = \frac{Cst_2 \times V^2}{V} = Cst_2 \times V}$$

According to the previous formula, V can be expressed in terms of frequency F as follows:

$$\boxed{V = \frac{F}{Cst_2}} \text{ (d)}$$

where Cst_2 is a constant related to the circuit type of the processor.
If the system is running over ω time, the energy consumption is

$$\boxed{E = P \times \omega} \text{ (b)}$$

By (a), (b) and (d), we have

$$\boxed{E = Cst_1 \times \frac{F^2}{Cst_2^2} \times F \times \omega = Cst_3 \times F^3 \times \omega} \text{ (e)}$$

where $Cst_3 = \frac{Cst_1}{Cst_2^2}$.
If we apply formula (e) to the formalization proposed in this work, we can calculate the energy consumption E_i^0 for each task τ_i^0, i.e.,

$$\boxed{E_i^0 = Cst_3 \times (f^p)^3 \times T_i^0} \text{ (f)}$$

where f^p is the frequency of the related core X_p in which τ_i^0 is running, and T_i^0 is the execution time of τ_i^0. As we have in (1), $T_i^0 = \left\lceil \frac{C_i^0}{f^p} \right\rceil$, the formula (f) can be written as:

$$\boxed{E_i^0 = Cst_3 \times (f^p)^3 \times \left\lceil \frac{C_i^0}{f^p} \right\rceil}$$

$$\boxed{\Leftrightarrow E_i^0 = Cst_3 \times (f^p)^2 \times C_i^0}$$

Likewise for an aperiodic task τ_l^2, its required energy can be written as:

$$\boxed{E_l^2 = Cst_3 \times (f^p)^2 \times C_l^2}$$

Each periodic task τ_i^0 produces an infinite sequence of jobs τ_{ij}^0, where j is a positive integer. Each job τ_{ij}^0 is described by: i) release time r_{ij}^0, ii) relative deadline d_{ij}^0 to be calculated, and iii) end execution time f_{ij}^0.

5.3.5 Aperiodic Task Model

Each aperiodic task τ_l^2, $l \in \{1, ..., o\}$, is defined by: i) WCEC C_l^2, ii) execution time T_l^2 which depends on the frequency of core X_p on which τ_i^0 is running and the WCEC, and iii) required energy E_l^2, where

$$T_l^2 = \left\lceil \frac{C_l^2}{f^p} \right\rceil \tag{5.3}$$

and,

$$E_l^2 = Cst_3 \times (f^p)^2 \times C_l^2 \tag{5.4}$$

An aperiodic task can arrive in a completely random way. Thus, for each aperiodic task τ_l^2 we note by N_l the maximum number of occurrences estimated on time interval [1, t_1].

After defining periodic and aperiodic tasks, we identify the formula to compute

- Consumed energy $E_p(0, t_1)$ by each core X_p on time interval $[0, t_1]$.

  Let us denote by Ψ_p^0 the periodic tasks' required energy which have to be executed before instant t_1 on X_p. We calculate for each periodic task the maximum number of its jobs that can be executed on time interval $[0, t_1]$, then we multiply it by task's required energy in each invocation,

$$\Psi_p^0 = \sum_{\tau_i^0 \in X_p} E_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil \tag{5.5}$$

  and by Ψ_p^1 the aperiodic tasks' required energy which have to be executed before instant t_1 on X_p. We calculate for each aperiodic task the maximum number of occurences in time interval $[0, t_1]$, then we multiply it by the task's required energy,

$$\Psi_p^1 = \sum_{\tau_l^2 \in X_p} E_l^2 \times N_l \tag{5.6}$$

$$E_p(0,t_1) = \Psi_p^0 + \Psi_p^1 \tag{5.7}$$

- Consumed energy $E^c(0,t_1)$ by the entire system as the sum of cores' consumed energy in time interval $[0,t_1]$, i.e., all periodic and aperiodic tasks' required energy which have to be executed before instant t_1.

$$E^c(0,t_1) = \sum_{p=1}^{z} E_p(0,t_1) = \sum_{i=1}^{n} E_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} E_l^2 \times N_l \tag{5.8}$$

- Minimum frequency of the whole system as detailes in Proof 2 below,

$$f_m = max\{f_m^{X_p}\} \tag{5.9}$$

where,

$$f_m^{X_p} = \frac{1}{t_1}\Big(\sum_{\tau_i^0 \in X_p} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{\tau_l^2 \in X_p} N_l \times C_l^2\Big)$$

- Maximum frequency of the whole system as detailes in Proof 3 below,

$$f_M = \sqrt{\frac{E_{max}^u \times t_1}{Cst_3(\sum_{i=1}^{n} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{k} C_l^2 \times N_l)}} \tag{5.10}$$

where $x \in \{1, ..., m\}$.

Proof 2: Calculation of f_m In each core X_p, the cumulative execution time requested by jobs' tasks running on given time intarval $[0,\ t_1]$ must be at most equal to its duration $t_1 - 0 = t_1$, i.e., there is no any an overload on this interval,

$$\boxed{\sum_{\tau_i^0 \in X_p} T_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{\tau_l^2 \in X_p} N_l \times T_l^2 \leq t_1} \text{ (a)}$$

Since task execution time depends on the frequency of the core on which it is executed as mentioned in (1), then we must consider the case where this time is the greatest, i.e., when the core frequeny is the smallest.
Let us denote by $f_m^{X_p}$ the minimum frequency in core X_p, so the formula (a) can be written as:

$$\boxed{\sum_{\tau_i^0 \in X_p} \frac{C_i^0}{f_m^{X_p}} \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{\tau_l^2 \in X_p} N_l \times \frac{C_l^2}{f_m^{X_p}} \leq t_1}$$

$$\Leftrightarrow \boxed{\frac{1}{f_m^{X_p}} \left(\sum_{\tau_i^0 \in X_p} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{\tau_l^2 \in X_p} N_l \times C_l^2 \right) \leq t_1}$$

$$\Leftrightarrow \boxed{\frac{1}{t_1} \left(\sum_{\tau_i^0 \in X_p} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{\tau_l^2 \in X_p} N_l \times C_l^2 \right) \leq f_m^{X_p}}$$

Since $f_m^{X_p}$ is the minimum frequency in X_p, then

$$\boxed{f_m^{X_p} = \frac{1}{t_1} \left(\sum_{\tau_i^0 \in X_p} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{\tau_l^2 \in X_p} N_l \times C_l^2 \right)}$$

Finally, to find the minimum frequency f_m of the whole system Π it is enough to take the maximum among the minimum frequencies of its cores:

$$\boxed{f_m = max\{f_m^{X_p}\},\ \text{where } p \in \{1, ..., z\}}$$

Proof 3: Calculation of f_M
The processor's energy consumption must be at most equal to the maximum energy harvested in the storage associated to the processor. For that, we check that the harvested energy is less than or equal to the energy consumption by the system's tasks running on given time interval $[0, t_1]$.

$$\boxed{E^c(0,t_1) \leq E^p(0,t_1)}$$

$$\Leftrightarrow \boxed{\sum_{i=1}^{n} E_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} E_l^2 \times N_l \leq E_{max}^u \times t_1} \text{(a)}$$

Based on (2) and (4) and using the maximum frequency f_M by all cores, the formula (a) becomes as follows:

$$\boxed{\begin{array}{l}\sum_{i=1}^{n} Cst_3 \times (f_M)^2 \times C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} Cst_3 \times (f_M)^2 \times C_l^2 \\ \times N_l \leq E_{max}^u \times t_1\end{array}}$$

$$\boxed{\begin{array}{l}\Leftrightarrow (f_M)^2 (\sum_{i=1}^{n} Cst_3 \times C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} Cst_3 \times C_l^2 \times N_l) \\ \leq E_{max}^u \times t_1\end{array}}$$

$$\boxed{\Leftrightarrow (f_M)^2 \leq \frac{E_{max}^u \times t_1}{\sum_{i=1}^{n} Cst_3 \times C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} Cst_3 \times C_l^2 \times N_l}}$$

$$\boxed{\Leftrightarrow f_M \leq \sqrt{\frac{E_{max}^u \times t_1}{Cst_3 (\sum_{i=1}^{n} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} C_l^2 \times N_l)}}}$$

Since f_M is the maximum frequency in Π, then

$$\boxed{f_M = \sqrt{\frac{E_{max}^u \times t_1}{Cst_3 (\sum_{i=1}^{n} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} C_l^2 \times N_l)}}}$$

5.3.6 Message Model

Message m_{si} ensures the communication between two periodic dependent tasks τ_i^0 and τ_s^0 where τ_i^0 depends on τ_s^0, $i \in \{1, ..., n\}$ and $s \in \{1, ..., n\}$. It is defined by: i) release time R_{si}^3, while m_{si} can be activated only when the predecessor task τ_l^0 ends, then we consider the case when the activation is done as soon as possible, i.e., $R_{si}^3 =$

C_l^0, ii) worst-case transmission cycle (WCTC) C_{si}^3, iii) transfer time T_{si}^3 which is the time required to pass through the communication bus, iv) delay time B_{si}, v) period P_{si}^3, where $P_{si}^3 = P_i^0 = P_l^0$, and vi) relative deadline D_{si}^3. It is assumed that the energy consumed to make the transfer of messages is neglected. The transfer time T_{si}^3 depends on the frequency of core X_p on which τ_l^0 is running and the WCTC:

$$T_{si}^3 = \left\lceil \frac{C_{si}^3}{f^p} \right\rceil \tag{5.11}$$

Each message m_{si} produces an infinite sequence of instances m_{sif}, where f is a positive integer. Each instance m_{sif} is described by: i) release time r_{sif}^3, and ii) delay time b_{sif}.

5.3.7 Problems' Mathematical Formalization

The problem to be treated in this chapter is how to parameterize feasible scheduling of real-time tasks with precedence constraints upon a multi-core processor powered by renewable energy harvested from related environment. Our scheduling problem contains three subproblems,

- Harvested energy availability: During each operating mode, it is necessary to identify the appropriate frequency for ensuring the tasks' energy requirements without missing a given deadline. To cope with this issue, we define p operating modes each of which is described by the energy and frequency variation intervals. After that in each operating mode, we must ensure that energy consumed to execute tasks on time interval $[0, t_1]$ is always less than the energy harvested plus the initial energy stored in the ideal energy battery. This constraint is given by
$$E^c(0, t_1) \leq E_a + E_m^k \times t_1 \tag{5.12}$$
- Real-time tasks' scheduling: During each operating mode, each job has to be completed before the absolute deadline using the EDF scheduling algorithm. This constraint is given by
$$\forall i \in \{1, ..., n\}, \text{ and } j \in \{1, ..., \left\lceil \frac{t_1}{P_i^0} \right\rceil], f_{ij}^0 \leq r_{ij}^0 + D_i^0 \tag{5.13}$$
- Precedence Constraints: Given a partial order $\prec$ on tasks, the idea behind consistency with a given order is to enforce precedence constraints by using earlier deadlines. Thus, if $\tau_l^0 \prec \tau_i^0$ then this constraint is given by
$$R_l^0 = R_i^0, \text{ and } D_l^0 < D_i^0 \tag{5.14}$$
when both τ_l^0 and τ_i^0 are on the same core, otherwise by
$$R_s^0 = R_i^0, \text{ and } D_s^0 < D_{si}^3 < D_i^0 \tag{5.15}$$

5.4 Efficient Scheduling of Periodic and Aperiodic Tasks Under Real-time, Energy and Precedence Constraints on Multi-core Reconfigurable Systems

This section presents and details the proposed methodology to solve the challenges posed.

5.4.1 Overview of the proposed methodology

Compared to the previous contributions, the latter aims to deal with additional constraints, the most important of which is the multicore architecture. In fact, this contribution focuses on real-time scheduling with precedence constraints on multi-core systems powered by renewable energy harvested from the environment and having mixed task sets. The uncertainty of energy availability in energy harvesting systems makes the problem of dependent tasks scheduling on multi-core architecture more challenging. To overcome this issue, we rely, in this contribution, on varying the processor frequency according to the predefined environmental conditions.

As presented in figure 5.3, the proposed approach is decomposed into two phases as fellow:

- The first defines different operating modes, each of which is characterized by energy and frequency parameters to cope with the energy availability issue. The variation of frequency leads to the modification of tasks parameters. Hence, each operating mode corresponds to well-defined system implementation. In fact, each implementation contains all system tasks with updated parameters.

- The second computes the deadlines, ensuring the real-time system feasibility by considering the invocation of aperiodic tasks execution. Moreover, the tasks may run under precedence constraints. Based on tasks dependency, the computing of tasks deadlines has two alternatives:

  - If a task has not any predecessor, then the identification of deadlines is based on the maximum value among its jobs' deadlines.

  - If a task has one or more predecessors, then the identification of deadlines is based on the maximum value among its jobs deadlines and on the time needed to ensure message transfer.

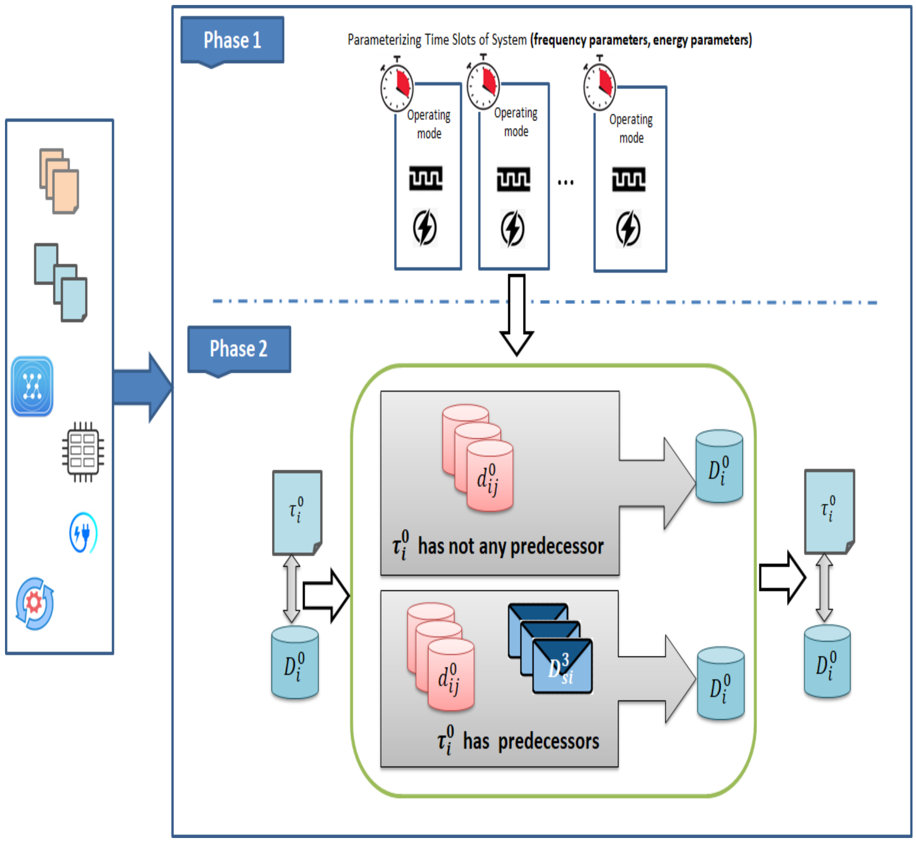

Phase 1
Parameterizing Time Slots of System (frequency parameters, energy parameters)
Operating mode
Operating mode
...
Operating mode
Phase 2
τ_i^0 has not any predecessor
d_{ij}^0
D_i^0
τ_i^0 has predecessors
D_{si}^3
τ_i^0

Figure 5.3: Overview of the proposed methodology.

5.4.2 Parameterizing Operating Modes of System

As mentioned previously, each operating mode θ_k is described by energy and frequency variation intervals.

Based on number p of operating modes, E^u_{min} and E^u_{max}, we calculate energy variation interval $E^k = [E^k_m, E^k_M]$ for each operating mode θ_k in such a way the difference between E^k_m and E^k_M is constant.

$$E^k_m = E^u_{min} + (y-1) \times (\frac{E^u_{max} - E^u_{min}}{p}) \tag{5.16}$$

$$E^k_M = E^u_{min} + y \times (\frac{E^u_{max} - E^u_{min}}{p}) \tag{5.17}$$

The calculation of frequency variation interval $F^k = [f^k_m, f^k_M]$ for each operating mode θ_k is done as detailed in Appendix D.

$$f^k_m = max\{\sqrt{\frac{E^k_M \times t_1}{Cst_3 \times (\sum_{i=1}^{n} C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^{o} C_l^2 \times N_l)}}\} \tag{5.18}$$

$$f_M^k = min\{\sqrt{\frac{E_M^k \times t_1}{Cst_3 \times (\sum_{i=1}^n C_i^0 \times \left\lceil \frac{t_1}{P_i^0} \right\rceil + \sum_{l=1}^o C_l^2 \times N_l)}}\} \tag{5.19}$$

where $x \in \{1, ..., m\}$.

5.4.3 Real-time and Energy Feasibility with Precedence Constraints on Multi-core Processor

After having parameterized operating modes, we present the deadline computing method which is executed in each one. Indeed, task execution time depends on the frequency of the core on which it is executed as mentioned in (1), i.e., task parameters change at each operating mode.

Remark 1. Throughout calculating deadlines in the proposed approach, we impose that $f^p = f_m^k$ for the fact that we consider in each operating mode its minimum instantaneous charging rate of energy $E_m^k \Rightarrow$ its minimum frequency f_m^k $\Rightarrow$ the maximum execution time for each task.

Let Δ_{ij} be the maximum cumulative execution time requested by tasks that have to be executed before each job τ_{ij} in X_p. Δ_{ij} includes three factors:

- $\phi_1 = (j-1) \times T_i^0$ represents the cumulative execution time requested by the previous instances of τ_i^0, i.e., if we are working on the jth instance, then we are sure that there are $(j-1)$ instances that have already been executed.
- $\phi_2 = \sum_{\tau_l^0 \in X_p and l \neq i} (\lfloor \frac{j \times P_i^0}{P_l^0} \rfloor - \alpha_l^i) \times T_l^0$ represents the cumulative execution time requested by the other tasks' jobs that are in the same core as τ_i^0, where α_l^{ij} is an integer, and

$$\alpha_l^{ij} = \begin{cases} 0 \text{ if } \lfloor \frac{j \times P_i^0}{P_l^0} \rfloor \times P_l^0 < (j-1) \times P_i^0 \\ 1 \text{ else.} \end{cases} \tag{5.20}$$

- $\phi_3 = \sum_{\tau_l^2 \in X_p} T_l^2 \times N_l$ represents the cumulative execution time requested by aperiodic tasks instances that are in the same core as τ_i^0.

  where, $l \neq i$.

$$\Delta_{ij} = \phi_1 + \phi_2 + \phi_3 \tag{5.21}$$

After calculating the maximum cumulative execution time Δ_{ij} estimated to be executed before τ_{ij}, we proceed to calculate its deadline d_{ij}^0. It is enough to take the difference between Δ_{ij} and the release time r_{ij}^0 of τ_{ij}^0, added to its execution time, where $j \in \{1, ... \left\lceil \frac{t_1}{P_i^0} \right\rceil\}$, i.e., the jobs of τ_i^0 that are executed on time interval $[1,\, t_1]$.

$$d_{ij}^0 = T_i^0 + (\Delta_{ij} - r_{ij}^0) \times \lfloor \frac{\Delta_{ij}}{r_{ij}^0} \rfloor \tag{5.22}$$

The proposed approach has the following alternatives:

- If task τ_i^0 has not any predecessor, i.e., τ_i^0 does not depend on any task, then it is enough to assign to each D_i^0 the maximum value among d_{ij}^0.

$$D_i^0 = max\{d_{ij}^0\} + T_i^0 \tag{5.23}$$

- If task τ_i^0 has one or more predecessors τ_l^0, i.e., $\Omega_i \neq \emptyset$ and the execution of τ_i^0 needs the response of its predecessors. Thus, for each task $\tau_s^0 \in \Omega_i$, we calculate the deadline D_{si}^3 of each message m_{si}. For that, we must consider the time needed to ensure message transfer, the finishing time of τ_s^0, i.e., R_{si}^3 and the delay time B_{si}.

The delay time of message m_{si} before being transmitted is due to the waiting of the end of the messages' transmission that are activated before m_{si}, i.e., first come, first served. The considered messages are only those using same communication bus as m_{si} that we denote by $\overline{\mathcal{M}_{si}}$.

Let us denote by L_{sif} the last idle instant before starting the fth instance of m_{si}, i.e., before r_{sif}^3. L_{sif} correspond to the minimum messages release time whose in $\overline{\mathcal{M}_{si}}$ and their executions remain until starting the fth instance of m_{si} and it is given by

$$L_{sif} = \min_{m_{vs} \in \overline{m_{si}}} \{\beta_{vs}\} \tag{5.24}$$

where,

$$\beta_{vs} = \begin{cases} \left\lceil \dfrac{r_{sif}^3 - r_{vs}^3}{P_{vs}^3} \right\rceil_0 \times P_{vs}^3 + r_{vs}^3 \text{ if } \left\lceil \dfrac{r_{sif}^3 - r_{vs}^3}{P_{vs}^3} \right\rceil_0 \times P_{vs}^3 + r_{vs}^3 \leq r_{sif}^3 \\ \left\lfloor \dfrac{r_{lsif}^3 - r_{vs}^3}{P_{vs}^3} \right\rfloor_0 \times P_{vs}^3 + r_{vs}^3 \text{ else.} \end{cases} \tag{5.25}$$

where $\lceil q \rceil_0 = max\{0, q\}$ and $\lfloor q \rfloor_0 = max\{0, q\}$.

Let us denote by $w(t)$ the workload requested by the messages $m_{vs} \in \overline{\mathcal{M}_{si}}$ before the instant t,

$$w(t) = \sum_{m_{vs} \in \overline{\mathcal{M}_{si}}} \left\lceil \frac{t - R_{vs}^3}{T_{vs}^2} \right\rceil_0 \times T_{vs}^3 \tag{5.26}$$

As L_{ljf} is the last idle instant before starting the fth instance of m_{si}, the difference between L_{sjf} and $w(L_{sjf})$ gives the total idle period before r_{sjf}^3 that we denote by I_{ljf},

$$I_{sjf} = L_{sjf} - w(L_{sjf}) \tag{5.27}$$

To calculate the delay time b_{sjf} of m_{sif} it is enough to take the difference between $w(L_{ljf})$ the workload requested by the messages $m_{vs} \in \overline{\mathcal{M}_{si}}$ added to the total idle period before r_{sjf}^3 and r_{sjf}^3,

$$b_{sjf} = (w(r_{sjf}^3) + I_{sjf} - r_{sjf}^3)_0 \tag{5.28}$$

The delay time B_{si} is defined as the maximum among b_{sif},

$$B_{si} = max\{b_{sif}\} \tag{5.29}$$

where $(q)_0 = max\{0, q\}$

The deadline of message m_{si} is given by

$$D^3_{si} = D^0_s + T^3_{si} + B_{si} \tag{5.30}$$

Remark 2. If a task τ^0_i depends on another task τ^0_s running on the same core, then there is no any message transfer, i.e., $T^3_{si} = 0$ and no any delay time, i.e., $B_{si} = 0$.

Let D^{Pred}_i be the maximum duration so that τ^0_i can receive all the messages coming from its predecessors after its activation.

$$D^{Pred}_i = max\{D^3_{si}\}, \forall\ s\ \in \Omega_i \tag{5.31}$$

A task must wait for the arrival of all messages coming from its predecessors and also must wait for the end of execution of the tasks having a priority higher than its own. Thus, its deadline must be the maximum value among these two waiting time.

$$D^0_i = max\{D^{Pred}_i, max\{d^0_{ij}\}\} + T^0_i \tag{5.32}$$

5.4.4 New Solution for Deadline Calculation of Mixed Real-time tasks on Multi-core Architecture under Energy and Precedence Constraints

The recursive algorithm below implements the proposed approach. It is based on the function $D\_Computing(\Omega_i)$ that applies the proposed approach to calculate a deadline for a given task τ^0_i. If the considered task has one or more predecessors, i.e., it depends on other tasks, then this function calls itself to compute the deadlines of these tasks. Indeed, the considered task τ^0_i must have a deadline longer than those of its predecessors to ensure the precedence constraints.

Algorithm 2 New method for deadline calculation

```
Require: $\Omega_i$, $\mathcal{P}$
Ensure: $D_i^0$
 1: function $D\_Computing(\Omega_i)$
 2:     for all $\tau_{ij}^0$ do
 3:         $d_{ij}^0 = T_i^0 + (\Delta_{ij} - r_{ij}^0) \times \lfloor \frac{\Delta_{ij}}{r_{ij}^0} \rfloor$
 4:     end for
 5:     if $\Omega_i = \emptyset$ then
 6:         $D_i^0 = max\{d_{ij}^0\} + T_i^0$
 7:         return $D_i^0$
 8:     else
 9:         while $\Omega_i \neq \emptyset$ do and $\tau_l^0 \in \Omega_i$
10:             if $\mathcal{P} \ni \tau_l^0$ then
11:                 $D_l^0 = D\_Computing(\Omega_l)$
12:             end if
13:             $\Omega_i = \Omega_i - \tau_l^0$
14:             $D_{si}^3 = D_l^0 + T_{si}^3 + B_{si}$
15:         end while
16:         $D_i^{Pred} = max\{D_{si}^3\}$
17:         $D_i^0 = max\{D_i^{Pred}, max\{d_{ij}^0\}\} + T_i^0$
18:         return $D_i^0$
19:     end if
20:     $\mathcal{P} = \mathcal{P} - \tau_i^0$
21: end function
```

5.5 Conducted Experimentation

In this section, we illustrate the proposed approach through a case study which consists on temperature controller system powered by a renewable energy source.

5.5.1 Presentation of Temperature controller System

The temperature controller system, presented in figure 5.4, is used in in the healthcare industry to increase the accuracy of temperature control. It does this by first measuring the temperature, it then compares it to the desired value. The difference between these values is known as the error. Temperature controllers use this error to decide how much heating or cooling is required to bring the process temperature back to the desired value.

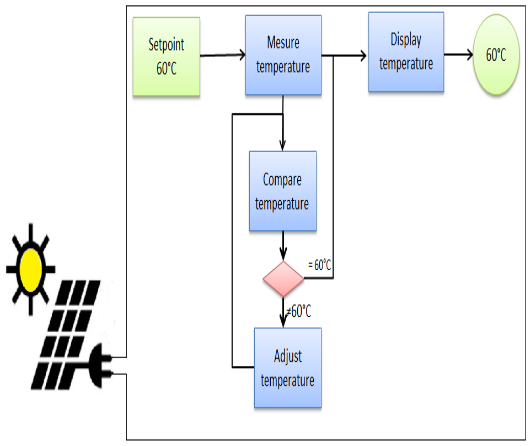


Figure 5.4: Case study modelisation.

We assume that the system has on two cores. The first core has three periodic tasks τ_1^0, τ_2^0 and τ_5^0. The second one has two periodic tasks τ_4^0 andτ_3^0 and an aperiodic task τ_1^2 which is activated at most once on time interval $[0, 60]$ according to user requirements. We present in Table 5.1 the tasks set.

Table 5.1: System tasks.

Task	Function	WCEC	Period	core
τ_1^0	mesure the temperature	2	10	X_1
τ_2^0	send the mesured value for displaying	5	20	X_1
τ_3^0	adjust the temperature	1	15	X_2
τ_4^0	compare the temperature to the desired value	2	10	X_2
τ_5^0	display the temperature value	1	20	X_1
τ_1^2	refresh display	2		X_2

We assume that τ_2^0 depends τ_5^0. As they are in the same core X_1, thus the communication between them is established without the intervention of any message. Moreover, τ_4^0 depends on τ_1^0. In fact, according to , it compares the measured value by τ_1^0 with the desired value. As τ_1^0 and $\tau_4^0$are not in the same core, thus the communication between them is established with the message m_{14}, where $T_{14}^3 = 2$,

5.5.2 Application of the Proposed Approach

According to user requirements, the cold room has three operating modes, i.e., $p = 3$, and it is powered by a renewable energy source, where $E_{min}^u = 0, 6$, and $E_{max}^u = 6, 6$. Furthermore, the processor can dynamically adapt its working frequency in some continuous range $F = [f_m, f_M]$ which is determined as bellow, (we suppose that $Cst_3 = 1$)

Let $f_m^{X_1}$ be the minimum frequency in core X_1, according to (9):

$$f_m^{X_1} = \frac{2}{10} + \frac{5}{20} + \frac{1}{20} = \frac{10}{20} = 0,5$$

Let $f_m^{X_2}$ be the minimum frequency in core X_2, according (9):

$$f_m^{X_2} = \frac{1}{15} + \frac{2}{10} + \frac{2}{30} = \frac{2}{30} + \frac{6}{30} + \frac{2}{30} = \frac{10}{30} = 0,3$$

Thus, $f_m = max\{0,5, 0,3\} = 0,5$

According to (10) :

$$f_M = \sqrt{\frac{6,6 \times 60}{Cst_3 \times (2 \times 6 + 5 \times 3 + 1 \times 4 + 2 \times 6 + 1 \times 3 + 2 \times 1)}} = 2,9$$

Then, we parameterize the three operating modes θ_1, θ_2, and θ_3. For that, we define energy parameters as follows:
According to (16),

$$E_m^1 = E_{min}^u + (1-1) \times (\frac{E_{max}^u - E_{min}^u}{3}) = 0,6 + (1-1) \times (\frac{6,6 - 0,6}{3}) = 0,6 \quad (5.33)$$

According to (17),

$$E_M^1 = E_{min}^u + 1 \times (\frac{E_{max}^u - E_{min}^u}{3}) = 0,6 + 1 \times (\frac{6,6 - 0,6}{3}) = 2,6 \quad (5.34)$$

Similarly, we have $E_m^2 = 2,6$, $E_M^2 = 4,6$, $E_m^3 = 4,6$, and $E_M^3 = 6,6$.

We present in 5.5 the operating modes' energy parameters.

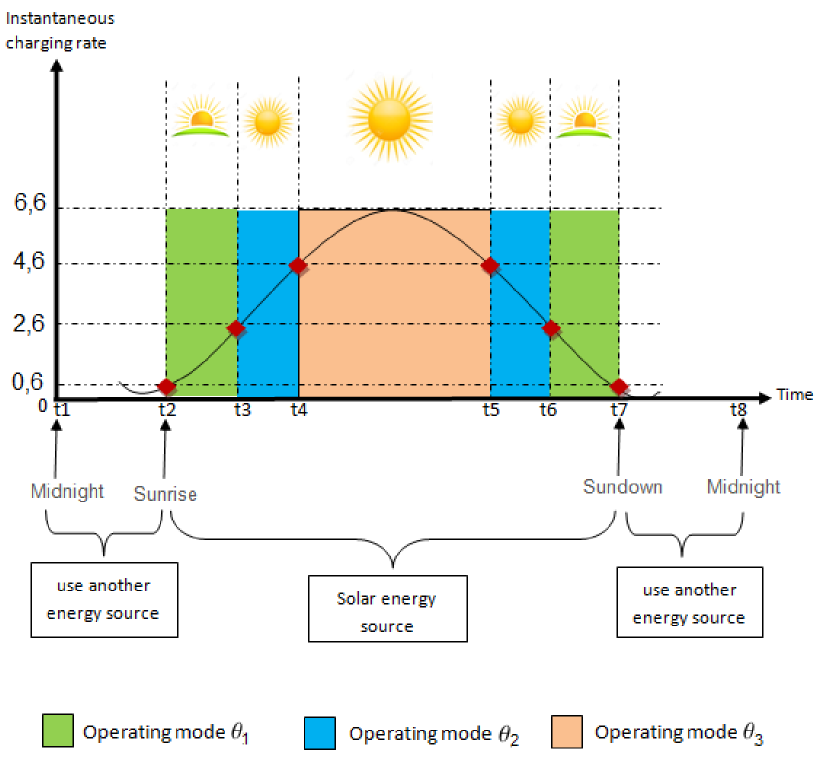


Figure 5.5: Presentation of operating modes.

After that, we define the frequency parameters as follows: According to (18),

$$f_m^1 = max\{\sqrt{\frac{0,6 \times 20}{16}}, \sqrt{\frac{0,6 \times 30}{28}}\} = max\{0,85,0,80\} = 0,85 \tag{5.35}$$

According to (19),

$$f_M^1 = min\{\sqrt{\frac{2,6 \times 20}{16}}, \sqrt{\frac{2,6 \times 30}{28}}\} = max\{1,8,1,6\} = 1,8 \tag{5.36}$$

Similarly, we have $f_m^2 = 1,8$, $f_M^2 = 2,4$, $f_m^3 = 2,4$, and $f_M^3 = 2,9$.

5.5.3 Case Study Results and Evaluation

The calculated results are displayed in Table 5.2.

Table 5.2: Calculation results.

	τ_1^0	τ_2^0	τ_3^0	τ_4^0	τ_5^0	m_{14}	m_{25}
Deadline in θ_1	4	9	2	10	11	7	9
Deadline in θ_2	2	5	1	6	8	4	5
Deadline in θ_3	2	5	1	4	7	3	5

Figure 5.6 shows the evolution of consumed and produced energy in θ_1 while calculating the consumed energy by using the minimum instantaneous charging in, i.e., $0,6j$. In fact, as presented in fig.5.5, the instantaneous charging in θ_1 varies between $0,6j$ and $2,6j$. We note that the curve of consumed energy is under the one of produced energy. Thus, we conclude that ,even if we consider low energy production, the energy produced is greater than the energy consumed over time, i.e., the energy constraints are respected.

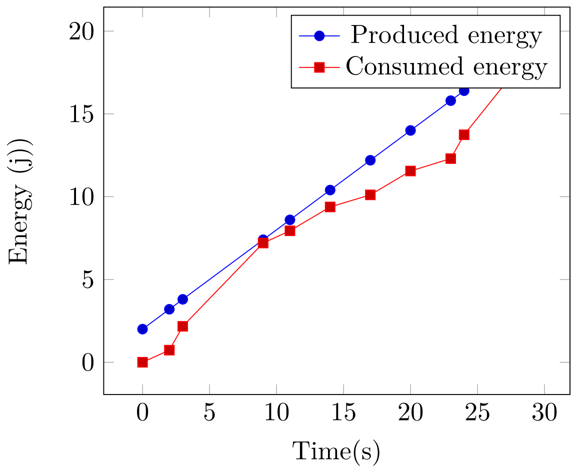


Figure 5.6: Evolution of the consumed and produced energy inθ_1.

Figure 5.7 shows the scheduling of tasks by using the calculated deadlines in θ_1. We note that real-time constraints are respected even if there is a wait time due to messages' transmission between dependent tasks. Therefore, the proposed approach guarantees the respect of both real-time and energy constraints while considering task dependency.

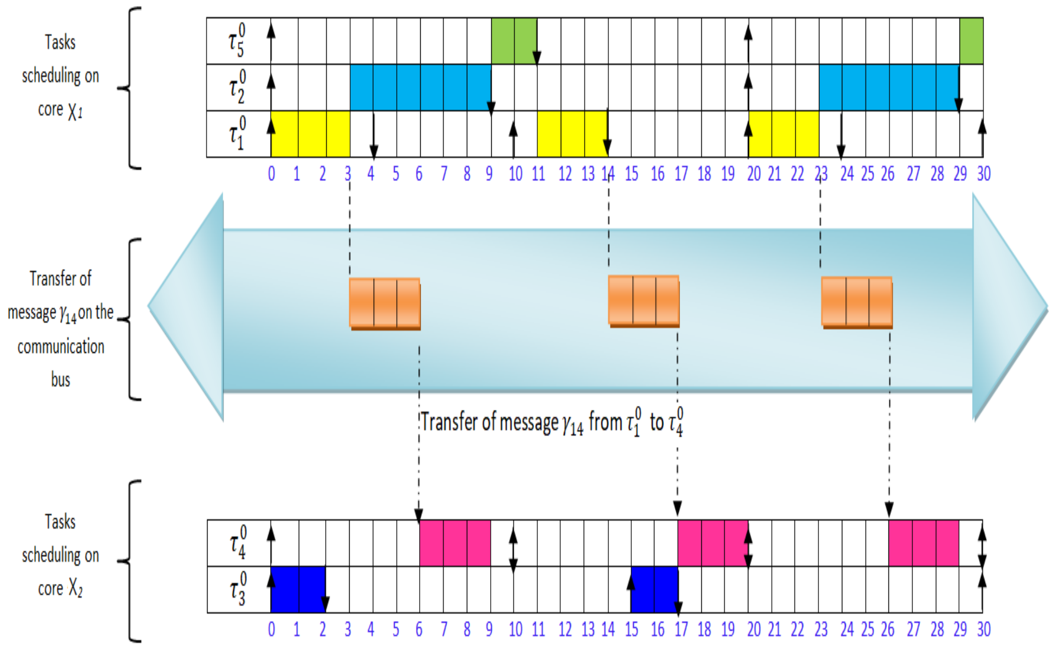

Tasks scheduling on core χ_1
τ_5^0
τ_2^0
τ_1^0
Transfer of message γ_{14} on the communication bus
Transfer of message γ_{14} from τ_1^0 to τ_4^0
Tasks scheduling on core χ_2
τ_4^0
τ_3^0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Figure 5.7: Scheduling of tasks by using deadlines in θ_1.

5.6 Discussion

The main contribution in this chapter is parametrizing feasible reconfigurable multi-core real-time systems while considering et set of challenges together. The main steps followed are the parameterization of operating modes to cope with energy requirements, and the computing of tasks and messages' deadlines to ensure real-time and precedence constraints by using calculated deadlines for tasks based on deadlines of their predecessors.

The originality of the current work compared with related researches, detailed in Chapter 2, is that it configures feasible scheduling of multi-core reconfigurable real-time system while considering mixed task sets, dependency, energy harvesting, and real-time constraints.

- Multi-core architecture: tasks are partitioned into different cores.
- Reconfiguration: tasks parameters are updated from one implementation to another.
- Real-time correctness: all periodic tasks meet their deadline.
- Energy requirements: each task has the required energy for its execution.
- Precedence constraints: each task have a longer deadline than those of its predecessors, if they exist.

- Mixed tasks set: the consideration of aperiodic tasks invocation.

5.7 Conclusion

In this chapter, we presented a new method for configuring feasible reconfigurable multi-core real-time systems having both periodic and aperiodic real-time tasks, powered by renewable energy and may have precedence constraints. Firstly, to cope with the energy harvesting availability issue, the proposed approach defines different operating modes and assigns to each one the appropriate interval of energy and frequency variation. Due to this variation, tasks parameters are updated from one operating mode to another, resulting new implementation of the system. Secondly, in each operating mode as-well as in each implementation, it calculates the tasks' deadlines which will be certainly respected online without any additional feasibility analysis. The originality of the contribution is that treats different and independent challenges together, i.e., multi-core, task dependency, energy constraints, and periodic and aperiodic real-time tasks. To the best of our knowledge, no one in all related studies treats these issues together.

Chapter 6

Conclusion

6.1 Thesis Context and Problems

The design of reconfigurable real-time systems is developing more and more with the increasing integration of critical functionalities in industrial processes. These systems face many challenges, the main one is the respect of real-time feasibility, where each task must meet its deadline. Furthermore, a real-time system often needs to execute both periodic and aperiodic tasks to achieve its functionalities, which complicates the scheduling problem. In fact, scheduling algorithms for mixed tasks set must be able to improve response times for soft aperiodic tasks without jeopardizing the schedulability of hard periodic tasks. Moreover, as they are reconfigurable, these systems can adapt their behavior at any time according to the evolution of the environment and the user requirements. This adaptation is manifested by adding or removing tasks, so that at a given time and under well-defined conditions, the system may only need to execute a subset of its tasks to achieve the required functionality. However, the real-time constraints may be violated, and the system may not be feasible. In addition to this, a real-time system may cope with several other constraints:

- Energy constraint: a real-time system have to operate continuously without missing the available energy. In fact, a task may not be able to complete its execution and then violates its deadline because of a lack of energy.
- Task dependency constraint: the real-time software tasks may be blocked because of :(i) resource sharing (a task may violate its deadline when it waits for a resource locked by another task) or (ii) precedence dependency (a task may violate its deadline when it waits for the result provided by another one, before its execution)

Thus, the main goal of this thesis is an efficient characterization of tasks' deadlines to ensure feasible real-time scheduling while coping with the exposed challenges above. However, the characterization of a deadline is by itself a relatively unexplored problem in the real-time community. To the best of our knowledge, most of the literature seems to consider that deadlines are somehow provided by others, possibly by control system engineers and no one in all related works considers the calculation of deadlines under the exposed challenges.

6.2 Thesis Contributions and Originalities

As mentioned previously, the main objective of the thesis is therefore to propose a new strategy for computing efficient tasks deadlines ensuring real-time feasibility on mono or multi-architecture with mixed tasks set while considering the energy availability and tasks dependency. In this thesis, we have chosen to address this issue incrementally to progressively deal with related problems.

First, we are particularly interested in the scheduling of tasks for mono-core architecture. We propose a new strategy based on parametrizing feasible scheduling of software tasks of various types (periodic, sporadic, and aperiodic) and constraints (hard and soft). We propose a new simulator called GIGTHIS-TOOl that applies the proposed approach.

Then we have extended it to address the scheduling tasks under energy and resource sharing constraints and reconfiguration property. We propose a new visual simulator named DEAD-CALC that encodes the proposed approach and integrates another tool RANDOM-TASK that is a random tasks generator.

Finally, an extension of the latter is proposed in order to configure feasible multi-core real-time systems having both periodic and aperiodic real-time tasks, powered by renewable energy, and may have precedence constraints. To cope with the energy harvesting availability issue, the proposed approach defines different time slots and assigns to each one the appropriate interval of energy and frequency variation.

6.3 Thesis Perspectives

The research project aims to resolve the identified problems. Nevertheless, the developed work is open to many perspectives:

- The combination of our proposed approaches is a necessity to make a more complete and reliable solution that integrates all the exposed challenges at the same time. This makes the proposed solutions more complete and reliable.

- After accomplishing the first prospect (the combination of the contributions), we will integrate artificial intelligence methodologies in making decisions to execute the adequate treatments that can respond to real-time system needs. In other words, we will develop intelligent agents, each of which integrates well-defined treatment to cope with well-defined constraints. Then, after predicting all possible factors that may affect system feasibility, another intelligent agent will alert the adequate agent to be executed to determine the effective temporal properties.

- Consideration of memory constraints: Real-time applications are often characterized by highly fluctuating memory requirements. However, the use of memory management in real-time systems has not been considered as an important issue, and has not received much consideration. In fact, each task should have enough access to the memory for its execution. Otherwise, it is probable that one or more tasks couldn't make progress because of a lack of memory, and then it may violate its deadline. Thus, we will consider memory constraints when identifying the temporal properties of tasks.

- The concepts of the proposed frameworks are implemented using WLangage and sql language. The developed tools in chapter 2 and chapter 3 need some improvements, such as more testing by applying more case studies. Moreover, we aim to improve the quality of the code to make the tool more powerful.

Furthermore, we aim to develop another tool that applies the contribution proposed in chapter 4.

- Implementing and applying the contribution of our thesis to a real-time operating system that will be evaluated by considering real case studies.

Bibliography

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. *Acm Sigmod Record*, 17(1):71–81, 1988.

[2] M. Abdel-Basset, R. Mohamed, M. Abouhawwash, R. K. Chakrabortty, and M. J. Ryan. Ea-msca: An effective energy-aware multi-objective modified sine-cosine algorithm for real-time task scheduling in multiprocessor systems: Methods and analysis. *Expert systems with applications*, 173:114699, 2021.

[3] M. S. Ajmal, Z. Iqbal, F. Z. Khan, M. Bilal, and R. M. Mehmood. Cost-based energy efficient scheduling technique for dynamic voltage and frequency scaling system in cloud computing. *Sustainable Energy Technologies and Assessments*, 45:101210, 2021.

[4] M. H. Awadalla. Processor speed control for power reduction of real-time systems. *Intr. J. of Electrical and Computer Engineering*, 5:701–713, 2015.

[5] Ö. Babaoğlu, K. Marzullo, and F. B. Schneider. A formalization of priority inversion. *Real-Time Systems*, 5(4):285–303, 1993.

[6] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3:67–99, 1991.

[7] P. Balbastre, I. Rippol, and A. Crespo. Minimum deadline calculation for periodic real-time tasks in dynamic priority systems. *IEEE Trans. Comput.*, 57:96–109, 2008.

[8] S. Banerjee, J. Chatterjee, and S. Tripathy. Application of magnetic energy storage unit as load-frequency stabilizer. *IEEE Transactions on Energy Conversion*, 5(1):46–51, 1990.

[9] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[10] M. Bell, F. Berkel, and S. Liu. Real-time distributed control of low-voltage grids with dynamic optimal power dispatch of renewable energy sources. *IEEE Transactions on Sustainable Energy*, 10(1):417–425, 2018.

[11] M. O. Ben Salem, M. Khalgui, A. Koubaa, E. Guerfala, Z. Li, and E. Tovar. Dual mode for vehicular platoon safety: Simulation and formal verification. *J. Information Sciences*, 402:216–232, 2017.

[12] M. O. Ben Salem, O. Mosbahi, M. Khalgui, Z. Jlalia, G. Frey, and M. Smida. Brometh: Methodology to design safe reconfigurable medical robotic systems. *Eur. Phys. J. B.*, DOI: 10.1002/rcs.1786:353–362, 2016.

[13] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[14] L. Borin, G. Lima, M. Castro, and P. D. Plentz. Dynamic power management under the run scheduling algorithm: a slack filling approach. *Real-Time Systems*, 57(4):443–484, 2021.

[15] T. Burd and R. Brodersen. Energy efficient cmos microprocessor design. In *Proc. 28th Hawaii Inter. Conf. System Sciences*, USA, 1995. Wailea, HI.

[16] G. C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[17] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proceedings Real-Time Systems Symposium*, pages 330–339. IEEE, 1997.

[18] S. Cai and V. K. Lau. Mimo precoding for networked control systems with energy harvesting sensors. *IEEE Transactions on Signal Processing*, 64(17):4469–4478, 2016.

[19] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. H. Anderson, and S. K. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms., 2004.

[20] S. Carrara and G. Marangoni. Including system integration of variable renewable energies in a constant elasticity of substitution framework: the case of the witch model. *Energy Economics*, 64:612–626, 2017.

[21] E. Carrascosa, J. Coronel, M. Masmano, P. Balbastre, and A. Crespo. Xtratum hypervisor redesign for leon4 multicore processor. *ACM SIGBED Review*, 11(2):27–31, 2014.

[22] J. Chen, P. Han, Y. Liu, and X. Du. Scheduling independent tasks in cloud environment based on modified differential evolution. *Concurrency and Computation: Practice and Experience*, page e6256, 2021.

[23] S. Chen, Z. Li, B. Yang, and G. Rudolph. Quantum-inspired hyper-heuristics for energy-aware scheduling on heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1796–1810, 2015.

[24] M. Chetto. Optimal scheduling for real-time jobs in energy harvesting computing systems. *IEEE Transactions on Emerging Topics in Computing*, 2(2):122–133, 2014.

[25] M. Chetto. Optimal scheduling for real-time jobs in energy harvesting computing systems. *IEEE Trans. Emerg. Topics Comput.*, 2:122–133, 2014.

[26] M. Chetto and H. El Ghor. Scheduling and power management in energy harvesting computing systems with real-time constraints. *Journal of Systems Architecture*, 98:243–248, 2019.

[27] H. Chniter, Y. Li, M. Khalgui, A. Koubaa, Z. Li, and F. Jarray. Multi-agent adaptive architecture for flexible distributed real-time systems. *IEEE Access*, 6:23152–23171, 2018.

[28] H. Chniter, O. Mosbahi, M. Khalgui, M. Zhou, and Z. Li. Improved multi-core real-time task scheduling of reconfigurable systems with energy constraints. *IEEE Access*, 8:95698–95713, 2020.

[29] Z. Deng, Z. Yan, H. Huang, and H. Shen. Energy-aware task scheduling on heterogeneous computing systems with time constraint. *IEEE Access*, 8:23936–23950, 2020.

[30] M. Digalwar, P. Gahukar, B. K. Raveendran, and S. Mohan. Energy efficient real-time scheduling algorithm for mixed task set on multi-core processors. *International Journal of Embedded Systems*, 9(6):523–534, 2017.

[31] P. Dziurzanski and A. K. Singh. Feedback-based admission control for firm real-time task allocation with dynamic voltage and frequency scaling. *Computers*, 7(2):26, 2018.

[32] R. El Osta, M. Chetto, and H. El Ghor. An efficient aperiodic task server for energy harvesting embedded systems. In *2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, pages 148–153. IEEE, 2019.

[33] N. Elvin and A. Erturk. *Advances in energy harvesting methods.* Springer Science & Business Media, 2013.

[34] X. Fu, B. Tang, F. Guo, and L. Kang. Priority and dependency-based dag tasks offloading in fog/edge collaborative environment. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 440–445. IEEE, 2021.

[35] A. Gammoudi, A. Benzina, M. Khalgui, and D. Chillet. Energy-efficient scheduling of real-time tasks in reconfigurable homogeneous multicore platforms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(12):5092–5105, 2018.

[36] M. Gasmi, O. Mosbahi, M. Khalgui, L. Gomes, and Z. Li. Performance optimization of reconfigurable real-time wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(7):2623–2637, 2018.

[37] O. Gasmi, M.and Mosbahi, M. Khalgui, L. Gomes, and Z. Li. R-node: New pipelined approach for an effective reconfigurable wireless sensor node. *IEEE Trans. Syst., Man, Cybern., Syst.*, 48:892–905, 2018.

[38] A. Goubaa, M. Kahlgui, F. Georg, and Z. Li. Efficient scheduling of periodic, aperiodic, and sporadic real-time tasks with deadline constraints. In *International Conference on Software Technologies*, pages 25–43. Springer, 2020.

[39] A. Goubaa, M. Khalgui, G. Frey, and Z. Li. New approach for deadline calculation of periodic, sporadic and aperiodic real-time software tasks. In *ICSOFT*, pages 452–460, 2020.

[40] A. Goubaa, M. Khalgui, Z. Li, G. Frey, and A. Al-Ahmari. On parametrizing feasible reconfigurable systems under real-time, energy, and resource sharing constraints. *IEEE Transactions on Automation Science and Engineering*, 18(3):1492–1504, 2020.

[41] A. Goubaa, M. Khalgui, Z. Li, G. Frey, and M. Zhou. Scheduling periodic and aperiodic tasks with time, energy harvesting and precedence constraints on multi-core systems. *Information Sciences*, 520:86–104, 2020.

[42] H. Grichi, O. Mosbahi, M. Khalgui, and Z. W. Li. New power-oriented methodology for dynamic resizing and mobility of reconfigurable wireless sensor networks. *IEEE Trans. Syst. Man Cybern.: Systems*, 48:1120–1130, 2018.

[43] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Resource sharing protocols for real-time task graph systems. In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 272–281. IEEE, 2011.

[44] J. P. Holdren, G. Morris, and I. Mintzer. Environmental aspects of renewable energy sources. *Annual Review of Energy*, 5(1):241–291, 1980.

[45] W. Housseyni, O. Mosbahi, and M. Khalgui. Adaptive task mapping and scheduling for reconfigurable distributed embedded energy harvesting systems. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 630–636. IEEE, 2017.

[46] W. Housseyni, O. Mosbahi, M. Khalgui, Z. Li, L. Yin, and M. Chetto. Multiagent architecture for distributed adaptive scheduling of reconfigurable real-time tasks with energy harvesting constraints. *IEEE Access*, 6:2068–2084, 2017.

[47] J. Huang, R. Li, X. Jiao, Y. Jiang, and W. Chang. Dynamic dag scheduling on multiprocessor systems: reliability, energy, and makespan. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3336–3347, 2020.

[48] C. Imes, D. H. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86. IEEE, 2015.

[49] F. Jarray, H. Chniter, and M. Khalgui. New adaptive middleware for real-time embedded operating systems. In *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, pages 610–618. IEEE, 2015.

[50] B.-S. Kim, H. Park, K. H. Kim, D. Godfrey, and K.-I. Kim. A survey on real-time communications in wireless sensor networks. *Wireless communications and mobile computing*, 2017, 2017.

[51] S. I. Kim and J.-K. Kim. A method to construct task scheduling algorithms for heterogeneous multi-core systems. *IEEE Access*, 7:142640–142651, 2019.

[52] W. Lakhdhar, R. Mzid, M. Khalgui, Z. Li, G. Frey, and A. Al-Ahmari. Multi-objective optimization approach for a portable development of reconfigurable real-time systems: From specification to implementation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(3):623–637, 2018.

[53] S. Lee, H. Kim, and J. Lee. A soft aperiodic task scheduling algorithm in dynamic-priority systems. In *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*, pages 68–72. IEEE, 1995.

[54] J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *RTSS*, volume 92, pages 110–123. Citeseer, 1992.

[55] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Unknown Host Publication Title*, pages 261–270. IEEE, 1987.

[56] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[57] J. Li, G. Zheng, H. Zhang, and G. Shi. Task scheduling algorithm for heterogeneous real-time systems based on deadline constraints. In *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 113–116. IEEE, 2019.

[58] K. Li. Energy and time constrained task scheduling on multiprocessor computers with discrete speed levels. *J. Parallel Distrib. Comput.*, 95:15–28, 2016.

[59] K. Li. Power and performance management for parallel computations in clouds and data centers. *J. Comput. Syst. Sci.*, 82:174–190, 2016.

[60] K. Li. Energy-efficient task scheduling on multiple heterogeneous computers: Algorithms, analysis, and performance evaluation. *IEEE Trans. Sustain. Comput.*, 1:7–19, 2017.

[61] Q. Li and C. Yao. *Real-time concepts for embedded systems*. CRC press, 2003.

[62] C. Lin, N. Xiong, J. H. Park, and T.-h. Kim. Dynamic power management in new architecture of wireless sensor networks. *International Journal of communication systems*, 22(6):671–693, 2009.

[63] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[64] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[65] Y. Liu, G. Xie, Y. Tang, and R. Li. Improving real-time performance under reliability requirement assurance in automotive electronic systems. *IEEE Access*, 7:140875–140888, 2019.

[66] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE*, 104(5):1013–1024, 2015.

[67] J. C. Lyke, C. G. Christodoulou, G. A. Vera, and A. H. Edwards. An introduction to reconfigurable systems. *Proceedings of the IEEE*, 103(3):291–317, 2015.

[68] S. Malik, S. Ahmad, I. Ullah, D. H. Park, and D. Kim. An adaptive emergency first intelligent scheduling algorithm for efficient task management and scheduling in hybrid of hard real-time and soft real-time embedded iot systems. *Sustainability*, 11(8):2192, 2019.

[69] P. Marwedel. *Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things*. Springer Nature, 2021.

[70] A. Mascitti, T. Cucinotta, M. Marinoni, and L. Abeni. Dynamic partitioned scheduling of real-time tasks on arm big. little architectures. *Journal of Systems and Software*, 173:110886, 2021.

[71] I. S. Moreno and J. Xu. Customer-aware resource overallocation to improve energy efficiency in realtime cloud computing data centers. In *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8. IEEE, 2011.

[72] P. K. Muhuri, R. Nath, and A. K. Shukla. Energy efficient task scheduling for real-time embedded systems in a fuzzy uncertain environment. *IEEE Transactions on Fuzzy Systems*, 29(5):1037–1051, 2020.

[73] P. K. Muhuri, A. Rauniyar, and R. Nath. On arrival scheduling of real-time precedence constrained tasks on multi-processor systems using genetic algorithm. *Future Generation Computer Systems*, 93:702–726, 2019.

[74] C. Obermaier, R. Riebl, A. H. Al-Bayatti, S. Khan, and C. Facchi. Measuring the realtime capability of parallel-discrete-event-simulations. *Electronics*, 10(6):636, 2021.

[75] T. Okuma, T.and Ishihara and Y. H. Real-time task scheduling for a variable voltage processor. In *in Proc. IEEE Int. Symposium on System Synthesis*, 1999.

[76] S. Pandey and U. Shanker. Causes, effects, and consequences of priority inversion in transaction processing. In *Handling Priority Inversion in Time-Constrained Distributed Databases*, pages 1–13. IGI Global, 2020.

[77] S.-J. Park and J.-M. Yang. Supervisory control for real-time scheduling of periodic and sporadic tasks with resource constraints. *Automatica*, 45(11):2597–2604, 2009.

[78] S. Priya and D. J. Inman. *Energy harvesting technologies*, volume 21. Springer, 2009.

[79] Y. Qin, G. Zeng, R. Kurachi, Y. Matsubara, and H. Takada. Energy-aware task allocation for heterogeneous multiprocessor systems by using integer linear programming. *Journal of Information Processing*, 27:136–148, 2019.

[80] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings., 10th International Conference on Distributed Computing Systems*, pages 116–117. IEEE Computer Society, 1990.

[81] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *RTSS*, volume 88, pages 259–269, 1988.

[82] G. Rehaiem, H. Gharsellaoui, and S. B. Ahmed. Real-time scheduling approach of reconfigurable embedded systems based on neural networks with minimization of power consumption. *IFAC-PapersOnLine*, 49(12):1827–1831, 2016.

[83] I. Ripoll and R. Ballester-Ripoll. Period selection for minimal hyperperiod in periodic task systems. *IEEE Transactions on Computers*, 62(9):1813–1822, 2012.

[84] D. Sangorrin, S. Honda, and H. Takada. Dual operating system architecture for real-time embedded systems. *Jul*, 6:1–24, 2010.

[85] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parallel Distrib. Syst.*, 19:215–236, 2008.

[86] J. P. Sha, L.and Lehoczky and R. R. Task scheduling in distributed real-time systems. *IECON'87*, 857:909–917, 1987.

[87] L. Sha, T. Abdelzaher, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. K. Mok, et al. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2):101–155, 2004.

[88] P. Simon, Y. Gogotsi, and B. Dunn. Where do batteries end and supercapacitors begin? *Science*, 343(6176):1210–1211, 2014.

[89] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *Proceedings of the 2004 annual ACM SIGAda international conference on Ada: The engineering of correct and reliable software for real-time & distributed systems using Ada and related technologies*, pages 1–8, 2004.

[90] A. Slimani, P. Ribot, E. Chanthery, and N. Rachedi. Fusion of model-based and data-based fault diagnosis approaches. *IFAC-PapersOnLine*, 51(24):1205–1211, 2018.

[91] A. Soroudi, A. Rabiee, and A. Keane. Stochastic real-time scheduling of wind-thermal generation units in an electric utility. *IEEE Syst. J.*, 11:1622–1631, 2017.

[92] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[93] J. A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.

[94] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao. The case for feedback control real-time scheduling. In *Proceedings of 11th Euromicro Conference on Real-Time Systems. Euromicro RTS'99*, pages 11–20. IEEE, 1999.

[95] J. A. Stankovic and K. Ramamritham. *Hard real-time systems.* IEEE Computer Society Press, 1988.

[96] G. L. Stavrinides and H. D. Karatza. Scheduling real-time parallel applications in saas clouds in the presence of transient software failures. In *2016 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 1–8. IEEE, 2016.

[97] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[98] D. Suleiman, M. Ibrahim, and I. Hamarash. Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction. In *4th International Conference on Electrical and Electronics Engineering*, volume 12, 2005.

[99] K. Tanaka. Real-time scheduling for reducing jitters of periodic tasks. *Journal of Information Processing*, 23(5):542–552, 2015.

[100] V. Vargas, P. Ramos, J.-F. Méhaut, and R. Velazco. Nmr-mpar: A fault-tolerance approach for multi-core and many-core processors. *Applied Sciences*, 8(3):465, 2018.

[101] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. *Leibniz International Proceedings in Informatics, LIPIcs 106 (2018)*, 106:24, 2018.

[102] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. *Leibniz International Proceedings in Informatics, LIPIcs 106 (2018)*, 106:24, 2018.

[103] P. Wägemann, T. Distler, H. Janker, P. Raffeck, V. Sieh, and W. Schröder-Preikschat. Operating energy-neutral real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(1):1–25, 2017.

[104] X. Wang, I. Khemaissia, M. Khalgui, Z. Li, O. Mosbahi, and M. Zhou. Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks. *IEEE Trans. Autom. Sci. Eng.*, 12:258–271, 2015.

[105] X. Wang, Z. Li, and W. M. Wonham. Optimal priority-free conditionally-preemptive real-time scheduling of periodic tasks based on des supervisory control. *IEEE Trans. Syst., Man, Cybern., Syst.*, 47:1082–1098, 2017.

[106] Y. Xiang and S. Pasricha. Run-time management for multicore embedded systems with energy harvesting. *IEEE Transactions on very large scale integration (VLSI) Systems*, 23(12):2876–2889, 2015.

[107] Y. Xiang and S. Pasricha. Mixed-criticality scheduling on heterogeneous multicore systems powered by energy harvesting. *Integration*, 61:114–124, 2018.

[108] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, and K. Li. Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems. *IEEE Trans. Emerg. Topics Comput.*, 3:167–181, 2018.

[109] J. Xu and D. L. Parnas. On satisfying timing constraints in hard-real-time systems. *IEEE transactions on software engineering*, 19(1):70–84, 1993.

[110] R. Xu, D. Mossé, and R. Melhem. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Transactions on Computer Systems (TOCS)*, 25(4):9–es, 2007.

[111] J. Zhang, H. Li, G. Frey, and Z. Li. Reconfiguration control of dynamic reconfigurable discrete event systems based on ncess. *IEEE Transactions on Control Systems Technology*, 28(3):857–868, 2019.

[112] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li. Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster. *J. Information Sciences*, 319:113–131, 2015.

[113] T. Zhang, S. Liu, W. Qiu, Z. Lin, L. Zhu, D. Zhao, M. Qian, and L. Yang. Kpi-based real-time situational awareness for power systems with high proportion of renewable energy sources. *CSEE Journal of Power and Energy Systems*, 2020.

[114] J. Zhou, M. Zhang, J. Sun, T. Wang, X. Zhou, and S. Hu. Drheft: Deadline-constrained reliability-aware heft algorithm for real-time heterogeneous mpsoc systems. *IEEE Transactions on Reliability*, 2020.