Scenario-Based Meta-Scheduling for Energy-Efficient, Robust and Adaptive Time-Triggered Multi-Core Architectures

DISSERTATION

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

vorgelegt von

M.Sc. Babak Sorkhpour

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät

der Universität Siegen

Siegen – July 2019

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Betreuer und erster Gutachter

Prof. Dr. Roman Obermaisser, Universität Siegen

Zweiter Gutachter

Prof. Dr. Raimund Kirner, University of Hertfordshire

Prüfungskommission:

Prof. Dr. Roman Obermaisser

Prof. Dr. Raimund Kirner

Prof. Dr. Kristof Van Laerhoven

Prof. Dr. Madjid Fathi (Vorsitz der Prüfungskommission)

Tag der mündlichen Prüfung: 09. July 2019

Scenario-Based Meta-Scheduling for Energy-Efficient, Robust and Adaptive Time-Triggered Multi-Core Architectures

DISSERTATION

to obtain the degree of Doctor

of Science Engineering

Submitted by M.Sc. Babak Sorkhpour

Submitted to the Faculty of Natural Sciences and Technology

the University of Siegen

Siegen July 09, 2019

I dedicate this dissertation to my parents, my son, and my family.

Acknowledgements

First, I would like to express my special appreciation, regards, gratitude, and thanks to my advisor Professor Dr. Roman Obermaisser for his assistance and guidance. You have been a tremendous mentor and life coach for me, showing admirable patience.

I would also like to thank Professor Dr. Madjid Fathi and Professor Dr. Raimund Kirner for their encouragement and advice, allowing me to overcome challenges and grow as a research scientist.

I offer special thanks to my family, including my parents, my sister, and my brother, for their endless and immeasurable love and support, especially throughout this part of my life.

I give special thanks to my wonderful son for his unconditional love, support, and understanding. This heroic man is my constant motivation and source of hope.

Finally, and above all, I am profoundly and forever indebted to the creator of love, my Lord, for his unconditional love, support, and encouragement throughout my life.

It is my pleasure and honor to express my gratitude to all the people who contributed, in whatever manner, to the success of this work.

Babak Sorkhpour

Siegen

July 10, 2019

Declaration of authorship

I hereby certify that this dissertation and its context has been written by me and is based on my own research work, unless mentioned otherwise. No other person's work, research, or publication has been used in this thesis without due acknowledgement. All references and verbatim extracts have been quoted, and all sources of data and information, including graphs, figures, tables, and data sets, have been specifically acknowledged.

ABSTRACT

Complex electronic systems are used in many safety-critical applications (e.g., aerospace, automotive, nuclear stations), for which certification standards demand the use of assured design methods and tools. Scenario-based meta-scheduling (SBMeS) is a way of managing the complexity of adaptive systems via predictable behavioral patterns established by static scheduling algorithms. SBMeS is highly relevant to the internet of things (IoT) and real-time systems. Real-time systems are often based on time-triggered operating systems and networks and can benefit from SBMeS for improved energy-efficiency, flexibility and dependability.

This thesis introduces an SBMeS algorithm that computes an individual schedule for each relevant combination of events such as dynamic slack occurrences. Dynamic frequency scaling of cores and routers is used to improve energy efficiency while preserving the temporal correctness of time-triggered computation and communication activities (e.g., collision avoidance, timeliness). Models of applications, platforms and context are used by scheduling tools to prepare reactions to events and to generate meta-schedules.

In this work, techniques and tools are developed to schedule a set of tasks and messages on Network-on-chip (NoC) architectures to minimize total energy consumption, considering time constraints and adjustable frequencies. This algorithm is intended for mixed-criticality and safety-critical adaptive time-triggered systems and can cover fault-tolerance requirements. It can also help to react to fault events by recovering the system. We also introduce a meta-scheduling visualization tool (MeSViz) for visualizing schedules. We also introduce a meta-scheduling visualization tool (MeSViz) for visualizing schedules.

We experimentally and analytically evaluate the schedules' energy-efficiency for cores and routers. In addition, the timing is analytically evaluated, based on static slack and dynamic slack events. Simulation results show that our dynamic slack algorithm produces, on average, an energy reduction of 64.4% in a single schedule and 41.61% energy reduction for NoCs. By compressing the schedule graphs the memory consumption can be reduced by more than 61%.

KURZBESCHREIBUNG

In vielen sicherheitskritischen Anwendungen (z.B. Luft- und Raumfahrt, Automotive, Kernkraftwerke) kommen komplexe elektronische Systeme zum Einsatz, für die Zertifizierungsnormen den Einsatz von sicheren Konstruktionsmethoden und -tools vorschreiben. Die Szenario-basierte Meta-Planung (SBMeS) ist eine Möglichkeit, die Komplexität von adaptiven Systemen über vorhersehbare Verhaltensmuster zu steuern, die durch statische Planungsalgorithmen festgelegt werden. SBMeS ist sehr bedeutsam für das Internet der Dinge (IoT) und Echtzeitsysteme. Echtzeitsysteme basieren oft auf zeitgesteuerten Betriebssystemen und Netzwerken und können von SBMeS für mehr Energieeffizienz, Flexibilität und Zuverlässigkeit profitieren.

Diese Abhandlung stellt einen SBMeS-Algorithmus vor, der einen individuellen Zeitplan für jede relevante Kombination von Ereignissen wie dynamische Schlupfereignisse berechnet. Die dynamische Frequenzskalierung von Prozessorkernen und Routern dient der Verbesserung der Energieeffizienz unter Beibehaltung der zeitlichen Korrektheit von zeitgesteuerten Berechnungs- und Kommunikationsaktivitäten (z.B. Kollisionsvermeidung, Echtzeitfähigkeit). Mit Hilfe von Modellen von Anwendungen, Plattformen und Kontexten werden Planungswerkzeuge eingesetzt, um Reaktionen auf Ereignisse vorzubereiten und Meta-Planungen zu generieren.

Im Rahmen dieser Arbeit werden Techniken und Werkzeuge entwickelt, um eine Reihe von Berechnungen und Nachrichten auf Network-On-Chip (NoC)-Architekturen zu planen, mit dem Ziel, den Gesamtenergieverbrauch unter Berücksichtigung von Zeitvorgaben und einstellbaren Frequenzen zu minimieren. Der Algorithmus unterstützt sicherheitskritische adaptive zeitgesteuerte Systeme und kann die Anforderungen hinsichtlich der Fehlertoleranz abdecken. Er kann auch dazu beitragen auf Störungsfälle zu reagieren, indem er das System wiederherstellt. Außerdem stellen wir ein Meta-Planungstool (MeSViz) zur Visualisierung von zeitgesteuerten Plänen vor.

Wir analysieren und bewerten die Energieeffizienz der Pläne experimentell für Prozessorkerne und Router. Darüber hinaus wird das Zeitverhalten anhand von statischen und dynamischen Schlupfereignissen analytisch bewertet. Simulationsergebnisse zeigen, dass unser dynamischer Schlupfalgorithmus im Durchschnitt eine Energieeinsparung von 64,4% in einem einzigen Zeitplan und 41,61% Energieeinsparung für NoCs erbringt. Durch die Komprimierung der Zeitpläne kann der Speicherverbrauch um mehr als 61% reduziert werden.

List of Abbreviations

WCET	Worst-case execution time
TTEthernet	Time-triggered ethernet
NoC	Network-on-a-chip
CAN	Controller area network
MPSoC	Multi-processor-system-on-a-chip
MILP	Mixed integer linear programming
MPSoC	Multi-processor system-on-a-chip
MeS	Meta-scheduler
SBMeS	Scenario-based meta-scheduling
MeSViz	Meta-schedule visualizer
TT	Time-triggered
TTS	Time-triggered systems
TTN	Time-triggered network
TTC	Time-triggered communication
TTA	Time-triggered architecture
LIN	Local interconnect network
DVFS	Dynamic voltage and frequency scaling
SDF	Slow-down factor
TSDF	Task slow-down factor
MSDF	Message slow-down factor

Contents

Chapter 1: Introduction

Many algorithms, methods, and techniques are proposed for scheduling distributed embedded real-time systems. Schedulability analysis is a primary component of real-time systems scheduling. In particular, the real-time system depends on static schedules that define the use of computational and communication resources based on a global time base. In [1], Kopetz explains how the correctness of a real-time system also depends on the timing of the computational results.

A group of tasks and messages is said to be schedulable with a certain scheduling method if enough resources (e.g., cores, routers) are available to execute all these tasks and transmit all messages before their deadlines. Each real-time task and message is assigned a deadline, which is defined in an ***application model (AM)***. In the time-triggered paradigm [1] of real-time scheduling, processes are controlled and scheduled by the progression of time only, and a schedule is designed for the total duration of a system's execution. One of the typical techniques used for ***time-triggered systems (TTS)*** is the schedule table. These are easy to verify and thus favorable in safety-critical systems that must be certified [2].

Scenario-based scheduling can support adaptive ***TTS*** by decreasing dependence on expensive and complex hardware, dynamic computational costs, and equipment vendor solutions and by replacing or reducing componential hardware functions with scheduling implementations on low-cost multi-purpose devices.

Energy-efficiency, energy-management, energy-saving, and energy reduction methods and algorithms are used in many applications (e.g., mobile phones, smart TVs), while their applicability in safety-critical systems is restricted.

Network-on-chip (NoC) technology contributes significantly to the overall energy consumption of an ***MPSoC***, and we introduce a meta-scheduler (***MeS***) for ***SBMeS*** that supports ***dynamic voltage and frequency scaling*** **(*DVFS)*** in time-triggered ***NoCs*** and ***MPSoCs***.

Some of the results and methods (e.g., meta-scheduling) described in this thesis are also used in the ***SAFEPOWER*** project platform and documents [3].

1.1. Motivation

Embedded systems are pervasive in modern safety-relevant systems. They range from automotive electronics to aerospace flight control and multi-purpose complex aerospace vehicle systems; and many premium carmakers plan to invest heavily in e-cars, which significantly depend on embedded systems [4].

However, in the era of the IoT, the minimizing of power consumption is a primary concern for system designers. Scheduling optimization helps engineers and system designers to increase energy-efficiency and improve the behavior of a system [5].

Many embedded systems are based on ***time-triggered networks (TTN)*** and used in safety-critical applications (e.g., healthcare, e-cars, space, military, nuclear stations, and aircraft). Efficient scheduling algorithms and methods are required for such systems (e.g., mathematical programming, artificial intelligence, scheduling heuristics, neighborhood search [6]), where failure has severe consequences [7]. "*Scheduling limited resources among requesting entities is one of the most challenging problems in computer science* [8]".

In ***SBMeS*** systems, the MeS generates specific schedules for each situation triggered by relevant events (e.g., fault and slack). To evaluate schedules, system designers must design, model, compare, understand, debug, and simulate the schedules. These are the important challenges for ***SBMeS***. Adapting to significant events within the computer system or in the environment is another challenge in ***TTS***.

NoCs have emerged in recent years to improve performance and solve the challenges of existing interconnect solutions for many cores. ***NoCs*** provide a scalable and high-performance communication architecture for complex integrated systems.

Moreover, this solution tackles the challenge of power consumption, which is one of the essential concerns of complex embedded systems. Research findings indicate that the communication interconnect can consume up to 36% of the power required in an ***MPSoC*** [9]. This significant power consumption calls for low-power techniques for ***NoCs***.

The output of most schedulers is in text format, making it difficult to identify problems, especially when a large number of schedules are generated and must be debugged or compared [10].

This challenge is intangible when using text logs with large amounts of data or abstract graphics that absorb the engineer's mental resources. The majority of the schedule visualizers are designed to illustrate a single schedule and cannot cover multiple schedules in one scope. However, the generation of schedules via scenario-based scheduling solutions and algorithms for real-time multi-processor systems is gaining importance [10]. The ***MeS*** approach is to add dynamic actions by computing several valid schedules, dynamically chosen based on the system status [11].

SBMeS is a scheduling technique [12, 13], which predicts, controls, and models the circumstance events in safety-critical systems. ***MPSoC*** systems typically represent one of the most power consuming components of embedded systems, and most research focuses on reducing power and energy consumption of computational cores. ***MPSoCs*** typically support the scaling of frequency and voltage (e.g., ***DVFS***), as well as multiple sleep states for cores. However, frequency tuning for both cores and routers in multi-core architectures (e.g., ***NoC***) has so far been an open research challenge.

In addition, energy efficiency and energy management are becoming important issues in real-time systems design [14]. One ***NoC*** design goal has emerged in the embedded and real-time systems market as a means of reducing and managing power consumption [15].

To deploy ***TT NoCs***, static scheduling of application workloads is a prerequisite. In addition to ensuring application requirements, such as precedence constraints and deadlines, energy consumption is influenced significantly by task allocations and communication/execution plans [14].

Finding the optimum schedules for maximum energy reduction is the goal of the scheduling techniques presented in this work. This will result in better task allocations and communication plans. The scheduling algorithm extends previous work (e.g., [12], [14], [16]), introducing an energy reduction scheduling method for ***TTS*** [17]. Energy reduction is achieved via ***DVFS*** by ***SBMeS***, a widely used technique that provides the support to reduce energy consumption of embedded systems and multi-core architectures.

Energy-efficiency in ***DVFS*** is achieved by dynamically adjusting the voltage, frequency, and performance settings of the application. To achieve the full advantage of the slack that results from variations in task execution time, it is important to recalculate the frequency and voltage settings during these periods (i.e., online).

Energy-efficiency optimization uses dynamic frequency scaling, in which we can individually scale the frequency of each core and router. This algorithm is suitable for mixed-criticality [18] and safety-criticality [19] by supporting fault-tolerant [20] applications and adaptive systems. Compared to static slack (SS) scheduling techniques, our approach provides more energy efficiency and enhanced flexibility.

The frequency is tuned for each component (core or router) and is optimized depending on the task or message, regarding events and the system global time.

However, optimal voltage and frequency scaling algorithms are computationally expensive and complex, if used at runtime. Therefore, to overcome and reduce online complexity, we propose quasi-static scheduling [21] for frequency scaling, supporting ***TT*** multi-core architectures. This method allows exploitation of dynamic slack (DS) and avoids energy dissipation due to online adaptation of the frequency settings.

Our method can support fault-tolerance and energy-efficiency, which are significant objectives in many safety-critical systems. Our algorithm considers the task execution times, message injection and transmission times, and possibilities of frequency changes

and can schedule and map the tasks and messages to be executed before the corresponding deadlines [22].

Our ***SBMeS*** model optimizes the trade-off between reliability, fault-tolerance, and energy-efficiency to handle multiple tasks and message sets regarding events (i.e., based on frequency tuning and scaling inside cores and routers). We consider the impact of each event on other events (e.g., increasing or decreasing frequency) for better resource and energy management in the ***TTS***.

We present scheduling techniques for attaining the optimal schedules for different events with maximum energy reduction, while meeting other functional and nonfunctional constraints. A corresponding optimization problem is formulated in ***IBM Ilog CPLEX***, and the results indicate significant improvements in energy efficiency.

In the e-car area, safety is one of the most critical parameters; hence, the fault-tolerant scheduling method used in ***SBMeS*** (cf. Section 4.6.1) can be applied in automotive systems to achieve greater safety.

In this work, slack occurrence is defined as an event. The required algorithm, methods, and scenarios are designed to support this event in improving energy-efficiency.

In many works, schedule results are explained by presenting the abstract text of the essential information (e.g., tasks, messages, makespan, deadlines, times).

A visualization of a schedule enables the engineers and system designers to easily perform sanity checks – checking and tracing the tasks, messages, makespan, nodes, and in ***SBMeS***, the behavior and reactions for each scenario after an event. Although scheduling is an essential issue in ***TTS***, embedded systems, and computer science, few visualization tools [8] are available to help engineers and scientists extend and develop more reliable scheduling algorithms, methods, and models [23]. Some visualization tools show only abstract schedules as graphical output and contain neither complete information nor detailed explanations of the events and the schedules (e.g., differences or changes) [7].

Comparing, understanding, and debugging thousands of schedules generated by ***SBMeS*** poses severe challenges for system designers, as discussed later in this work.

This work introduces a scenario-based tool, ***MeSViz*** – designed to support developers and engineers in evaluating scheduling algorithms, models, and methods for adaptive ***TTS***. This tool can show event details and schedule changes and dependencies due to events. It visualizes schedules on four different layers: the first presenting individual schedules for each scenario, the second displaying multiple schedules for multi-scenario events, the third generating graphs, and the fourth showing energy and timing.

1.2. Research scope

This thesis presents ***SBMeS*** models and optimal algorithms and techniques, which can support adaptive ***TTS*** and reliability requirements for ***MPSoCs*** and ***NoCs***. This thesis addresses ***TTS*** because the temporal interference between the cores and routers of ***MPSoCs*** and ***NoCs*** significantly complicates the analysis of ***worst-case execution times (WCET)*** [24].

The novelty contributions of this thesis can be summarized as follows:

1. ***Scheduling of TT communication and computational activities***: The scheduling algorithm considers the task-execution times, message-transmission times and the possibilities of frequency changes. Our scheduler supports the mapping and scheduling of both tasks and messages to multi-core architectures, for minimizing the total energy consumption regarding the timing and frequencies of cores, routers and slack distribution.

2. ***DVFS in time-triggered architecture***: This work enables the use of *DVFS* for both communication and computation resources for *MPSoCs*, in order to attain energy efficiency of *TT* multi-core architectures. Hence, it can extend energy efficiency optimisation of *SBMeS* for ***MPSoCs*** with ***time-triggered communication (TTC)*** not only providing frequency scaling of not only cores but also the ***NoC's*** routers.

3. ***The scheduling method for improved reliability and fault-tolerance***: ***SBMeS*** can be used for automotive and safety-critical systems to achieve greater safety. In the e-car area, safety is one of the most critical parameters.

4. ***Trade-off between fault-tolerance, reliability, and energy-efficiency:*** Our proposed model optimizes the trade-off between fault-tolerance, reliability, and energy-efficiency in ***SBMeS*** to handle multiple tasks and message sets regarding reliability [20] and energy-efficiency for adaptive ***TTS*** (i.e., base frequency tuning and scaling inside cores and routers).

5. ***An optimal scheduling algorithm for energy efficiency***: Our proposed optimization method establishes the minimum energy consumption for each slack event by ***mixed-integer quadratic programming (MIQP)*** equations. The ***MIQP*** model considers different parameters (e.g., cores, routers) and decision variables (e.g., slow-down factors of cores and routers) in the constraints and the objective function.

6. ***Visualization of meta-schedules***: Our ***MeSViz*** is proposed for specific visualization and evaluation of single and multi-schedules on ***MPSoCs***.

7. ***Memory optimization regarding schedules size:*** Delta scheduling is used to reduce and optimize memory usage of schedules and can store a significant number of schedules based on delta graph generator models [25], [16].

In contrast, our approach is an off-line static-scheduling algorithm for determining the optimal meta-schedules and dynamic frequency scaling in ***TT MPSoCs*** and ***NoCs*** for energy-efficiency in safety-critical embedded systems. This work thus develops a DS - reclamation technique to reduce static and dynamic energy consumption.

1.3. Structure of the thesis

The remainder of the dissertation is structured as follows:

Chapter 2 analyses the literature, related and previous works, basic concepts, and the current state of the art. In section 2.8, we compare this work with related works in two tables to clarify the research gap.

Chapter 3 introduces the scheduling model for the adaptive ***TT*** and safe and mixed-criticality systems. Our model supports the static scheduling for ***TTS*** and creates specific data models for physical (hardware), application (logical), and context (scenarios and events) layers. The general definitions, energy model, and objective function are then distinguished (e.g., constraints and variables).

Chapter 4 provides a detailed description of our specific ***SBMeS*** technique, algorithm, and developed tool (***MeS***). The system model and algorithms, architecture, and fault assumptions are then distinguished.

Chapter 5 presents the techniques for visualization, evaluation, and convergence of schedules. It presents our specific meta-scheduling visualizing technique and the developed tool (***MeSViz***).

Chapter 6 concerns the implementation. In this chapter, we provide information about the schema model, based on standard ***XML*** as an input. The implementation of ***MeS*** and ***MeSViz*** and outputs are explained.

Chapter 7 presents the evaluation of the scenarios and models. In this chapter, we provide different scenarios and use cases for energy-efficiency (cores and communication network), energy-efficiency with fault-tolerance, and saving memory. In addition, the results are evaluated and visualized using ***MeSViz***.

Chapter 8 concludes the dissertation and suggests an outlook for future work on ***SBMeS*** on adaptive ***TT*** and mixed-criticality systems.

Chapter 2: Related work and basic concepts

In this section, we begin with basic concepts and review the ideas of related works in the area of real-time systems, ***TTS***, energy management, scheduling, and visualization; then briefly introduce our previous works [10, 12, 13]. In heterogeneous embedded systems (e.g., ***MPSoC***), many elements are used (e.g., CPU, GPU, and ***NoC***) and they must be designed, developed, and operated to satisfy the reliability and performance requirements of safety-critical embedded-systems applications. ***MPSoCs*** are increasingly popular, which they are using high computational power and low power consumption embedded systems.

2.1. Real-time systems

A real-time system must execute concurrent tasks and messages in such a way that all tasks and messages meet their specified deadlines. Communication and task deadlines are significant constraints that a scheduling algorithm for ***RTES*** must satisfy in addition to optimizing energy-savings [22].

2.1.1. Embedded real-time systems

In general, opinion about the real-time system is often considered to be implemented through an embedded system platform [23].

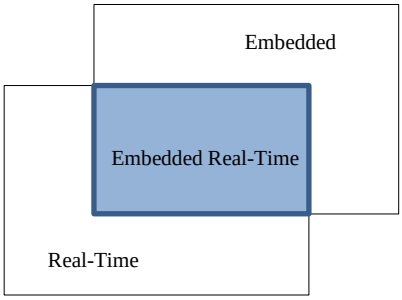


Figure 1. Conceptual difference [23]

In Figure 1, the embedded system and real-time system are strongly correlated, but there are applications and areas where this correlation is not observed [23]. However, this does not concern a difference between two different systems, but rather a new vision of the application of two systems in a single scope. Figure 2 represents the embedded system situation and the new scope of its additivity.

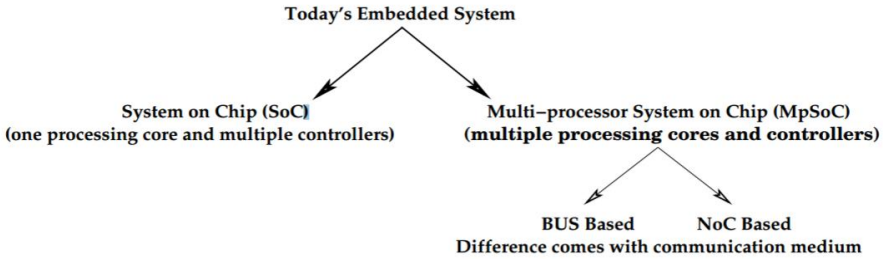


Figure 2. Current and future trend in an embedded system [23]

2.1.1.1. Makespan

In [26], Eitschberger et al. present a scheduling model to balance faults and energy to maximize the performance in static schedules. However, it is vital to minimize the time length of a schedule (the so-called *makespan* [27]): in effect, the duration until all tasks have finished processing, while integrating fault tolerance techniques. They propose that one of the recently emerging topics is the problem of minimizing energy consumption by ***NoCs*** [26, 28].

For example, ignoring the possibility of slack recovery is regarded as wasting energy. Energy consumption is also affected by the frequency scaling of cores and routers. By increasing the execution time via decreasing the clock frequencies in cores, the length of the makespan also increases. Fault-tolerance techniques typically result in performance overhead, which leads to an increase of the makespan.

However, decreasing the clock frequency also decreases energy consumption. This mechanism leads to a two-variable trade-off decision between performance and energy consumption, on both cores and routers.

2.1.2. Dependability

One of the most critical non-functional parts of real-time systems is *dependability* [29], which is the ability of a real-time system to provide its agreed level of service to its application [30]. According to [29], the IEC/IEV[1] $191-02-03$ has a specific description: "Dependability is the collective term used to describe the availability performance and its influencing factors: reliability performance, maintainability performance and maintenance support performance." According to [31], the conceptual structure of dependability comprises three important components, which are faults, means, and attributes of dependability. In this work, we focus on faults.

2.1.2.1. Mixed-criticality systems

Safety and reliability in ***mixed-criticality systems (MCSs)*** (e.g., vehicles, airplanes, drones) and their standards (e.g., $ISO\ 26262, IEC\ 61508, \mathrm{DO}-178\mathrm{B}, \mathrm{DO}-254,$ and $\mathrm{ISO}\ 26262$) are the primary focus of many works [32].

However, the current tendency is to increase integration, build safer and more reliable complex or large embedded systems. These systems requirements commonly refer to ***MCSs*** [33]; that is, including two levels: safety critical (high criticality) and mission-critical (low criticality). It is crucial for tasks to meet the requirements for criticality levels [34]. There can be up to five levels of standards [32].

Isakovic et al. [35] explain that physical component designs and interfaces with a specific computer system should provide a clear design methodology approach, but systems become more complex primarily when working with heterogeneous systems and

[1] International Electrotechnical Commission

protocols. However, it is challenging to ensure that the functional properties meet the system specifications and regulatory guidelines. The authors also offer a mixed-criticality integration solution based on a ***TTA*** for a hybrid system-on-a-chip platform.

The ***SBMeS*** algorithm is designed for static scheduling because dynamic scheduling has more complexity and is typically not amenable to certification.

Our algorithms and tools, used in [12, 13, 36], are extended with support for frequency scaling in a ***TT*** multi-core architecture. This technique can also be used for ***TTS*** with mixed-criticality [19], while meeting requirements for adaptivity, energy efficiency, and fault-tolerance, as in [25]. For example, ***SBMeS*** enables reactions to fault events by pre-planning for each event and fault, with a recovery strategy via static scheduling. In many safety-critical real-time systems, fault-tolerance and energy-efficiency are essential objectives [33].

2.1.3. Adaptivity

The design and development of embedded systems is driven by the constraints of certification standards [19]. While these standards advocate static resource allocations [37], adaptivity is also desirable for fault recovery and energy-efficiency [38]

Safety-critical systems are vital and sensitive, and any failure can have a significant effect on people or cause damage to the environment.

Recent technology trends favor the adoption of multi-core architectures with ***NoC*** for safety-critical applications [39]

In addition, ***TTNoC*** [40] and AEthereal [13] effectively support the fault isolation and predictability requirements of safety-critical systems. Time-triggered control is a valuable solution for safety-critical systems to manage the complexity and provide analytical dependability and timing models.

This work addresses adaptivity in ***TTS*** by deploying multiple precomputed schedules and routering between such schedules at runtime.

2.2. Time-triggered systems (TTSs)

According to [17], a real-time system is defined as one in which the correctness of its behavior depends on computational results as well as the physical time in which these results are produced concerning the global time base. In ***time-triggered architectures (TTA)***, activities are triggered by the progression of global time.

By dedicating a priori defined bandwidth to ***TT*** messages, timely delivery of all messages is guaranteed.

2.3. Global time base

In ***TTS*** architectures, activities are triggered by the progression and control of global time [17].

2.3.1. Time-triggered networks (TTNs)

TTNs (e.g., TTEthernet and FlexRay [41]) are advantageous in safety-critical systems for managing the complexity of fault-tolerance and analytical dependability models (TTEthernet is currently under standardization by SAE as *AS*6802 [42]) [13]. In addition, ***TT*** architectures in ***NoCs***, such as ***TTNoC*** [40] and ***AEthereal*** [43], support the fault-isolation and safety-critical requirements.

In-vehicle network protocols (e.g., FlexRay, Local Interconnect Network (LIN) [44]) use the ***TT*** pattern, the time of which is divided into communication cycles.

2.3.1.1. Messages

Once a task finished, data or an information packet is sent to the next dependent task, which this packet called message. Each message is scheduled for routing, sending, and receiving with the timing determined according to the tasks and messages dependency and to avoid message collisions in path links and routers.

2.3.2. Time-triggered multi-processor system-on-a-chip (MPSoC) and network-on-a-chip (NoC)

MPSoCs in heterogeneous systems include many elements (e.g., CPU, GPU, Cache, SRAM, FPGA and ***NoC***) [45]. With latency-sensitive applications running on such ***MPSoC*** platforms, the cores and routers must be designed and operated to satisfy the performance requirements. ***DVFS***, adaptive routing, and frequency scaling in routers and links can potentially improve energy efficiency and performance of the ***NoC***.

An ***MPSoC*** can be viewed by its physical and logical viewpoints. Physically, the platform consists of cores interconnected by a ***NoC***.

The message-based services of the ***NoC*** enable the abstraction from the internal implementation details of the cores, which can range from state machines in hardware to full-scale processors with operating systems and application software.

An application on an ***MPSoC*** can be described by a ***directed acyclic graph (DAG)*** [46], where tasks are connected by edges to represent data dependencies. An edge between two tasks represents a service required by the destination task. For example, the computation of set values in a “controller task” depends on sensory data from an “I/O task” providing information about actual and desired values.

Spatial and temporal allocation of the elements from the logical view of the physical platform is required before execution begins. Tasks are mapped to suitable cores, while message-based services are mapped to paths and time intervals on the ***NoC***. Likewise,

router configurations must be coordinated in such a way that the routing decisions prevent collisions.

Another advantage of using ***NoCs*** is the effective support for multiple clock domains, which is particularly important for the proposed concept in this work. It is essential to support the clock domain [47] crossings [48] for the proposed DFS [49] technique to reduce the system's power consumption.

NoCs are composed of three main building blocks: links, ***network interfaces (NI)***, and routers. Links serve as a physical connection between cores, routers, and ***NIs***. The ***NI*** is an interface that makes a logical connection between the cores and the network. The routers send and receive the packets, which are delivered by the other routers or cores through the ***NI***. Consequently, routers are responsible for sending the packets to the right destination, using the destination address included in the packet.

NoCs are composed of three main building blocks: links, ***NI***, and routers. More cores typically means higher power consumption: including more cores thus presents a design hurdle of architectural complexity and higher energy consumption. For example, the ***Kalaray MPPA2-256*** [50] comprises 288 cores: 256 computing cores, 16 management cores, and four quad cores (see Figure 3). ***Kalray's MPPA*** [51] technology addresses these challenges by combining high-performance cores with low-power processors [52].

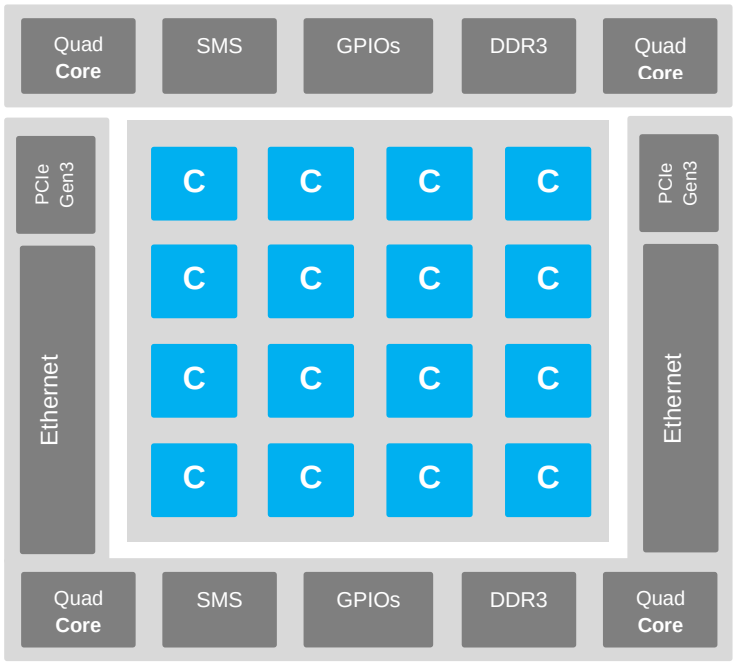


Figure 3. Kalray's MPPA network-on-chip (The MPPA2® – 256 Bostan2 processor [52])

Many ***MPSoCs*** are based on complex communication infrastructures similar to ***NoCs*** (e.g., Samsung Exynos, Multicore ***DSPs***, and ***many-core Kalray MPPA***). Using ***NoC*** for the implementation of the proposed concept is essential as ***NoCs*** make the interconnected elements independent of the clock frequency. More precisely, buffers between different components can be used as clock segregation boundaries if different frequencies are

demanded. Moreover, a routing algorithm is required to route the messages to the right destination.

In a real-time system [53], the correctness of its behavior depends on the values and the timing of computational results. Therefore, it is a beneficial solution in safety-critical systems to manage the complexity of analytical dependability and timing models.

2.3.3. Time-triggered (TT) execution environments

Real-time operating systems (RTOS) are used in embedded systems where time is a vital parameter and function; also RTOS needs to control and serve many resources requirement (e.g., core, memory). For example, ***XtratuM*** is a hypervisor based on ***RTLinux HAL*** to meet safety-critical real-time systems requirements. These ***RTOSs*** or hypervisors are supported by specific standards for specific use cases (e.g., ARINC 653 for avionics) [54].

Figure 4 shows how the Xtratum [55]/Xoncrete [56] design from the ***SAFEPOWER*** project can be applied, along with the meta-scheduling for the hard-core system [3].

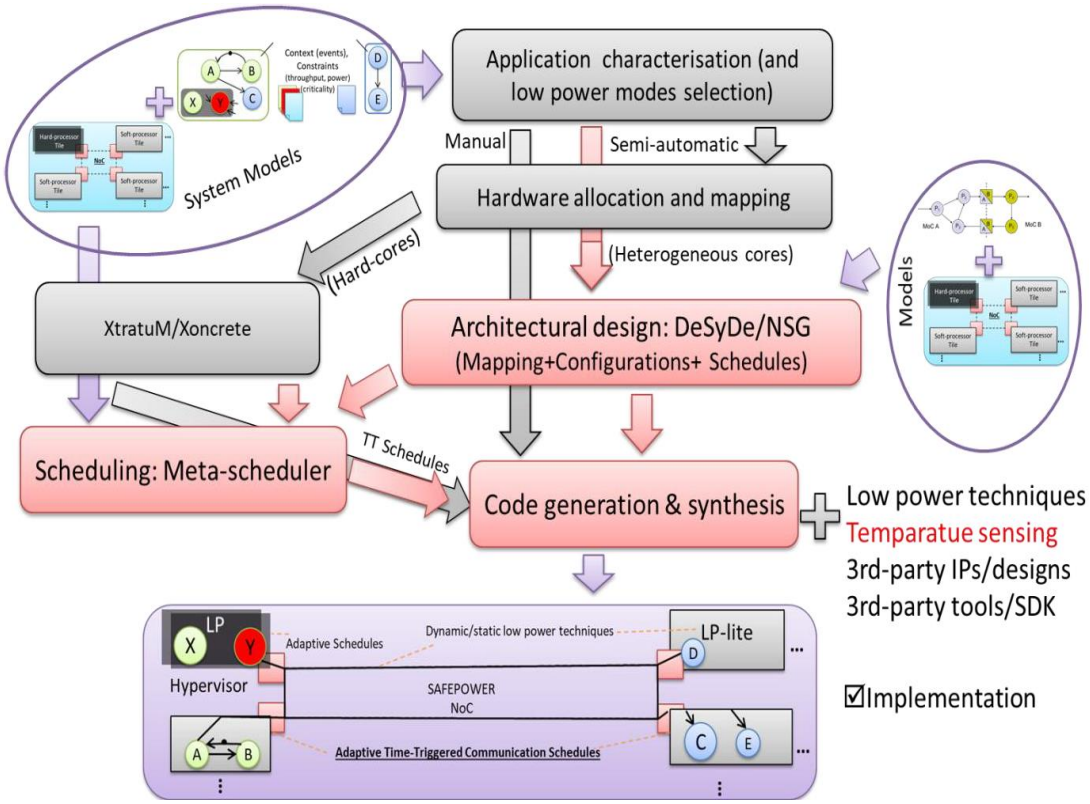


Figure 4. Simplified design flow and meta-scheduler (MeS) integration in SAFEPOWER [3]

2.3.3.1. Tasks

Applications or processes that is running on a core called to task. These tasks require a schedule for timing and allocation regarding priority, coherency, and/or dependency.

2.3.3.2. Worst-case execution time (WCET)

The ***worst-case execution time (WCET)*** of a task is pre-estimated or pre-computed as an input for scheduling. However, this determination is difficult to achieve.

2.3.3.3. Slack

The SSs happen when the system is under loaded, and DS occurs when it finishes faster or earlier than predicted ***WCETs***. Finally, DS can provide space for other tasks to be executed [34].

We further note that the execution time variations sometimes leads to DS time [57], which can be exploited by different levels of ***WCET*** and changed without causing a performance penalty. In this work, we use the slack time in cores to reduce energy consumption.

2.4. Energy management techniques

2.4.1. Clock gating and power gating

Clock gating and power gating are the well-known, general techniques for reducing power dissipation and energy management in hardware systems. Clock gating with logical techniques reduce clock power using a circuit to prune the clock tree [58]. Power gating is designed to shut off the circuit not in use and to increase time delay [59].

2.4.1.1. Dynamic voltage and frequency scaling (DVFS)

One of the most famous and prevalent run-time techniques for improving power efficiency is ***DVFS*** [60]. ***DVFS*** techniques are widely used for scaling and optimizing power [60], providing a solution in which the chip's voltage-frequency levels are varied at run-time [15].

In this work, ***DVFS*** is used to establish an optimized frequency for each core. The power efficiency of ***DVFS*** is primarily related to the performance of the slack estimation method [61].

Embedded systems use some well-known, energy-efficient techniques [62], including ***DVFS***, to improve energy-saving and the power-efficiency of chips at run-time [63].

Bianco et al. [64] analyzed a similar problem, concerning ***DVFS*** on ***NoC***, focusing on single voltage, single frequency, and links working at a single frequency. They considered only the ***DVFS*** related to the data transmission, due to the bit switching activity, and their simulation results did not consider or include the flit queueing effects and the contentions due to routing.

Caria et al. [65] show that a combination of fine-tune energy router hardware (akin to ***DVFS***) and ***traffic engineering (TE)*** in the network enables most energy savings. Their case study suggests that the energy savings of ***DVFS*** vary between 25% for highly utilized networks and 50% for lightly loaded networks.

2.4.1.1.1. Frequency tuning on network-on-chip (NoC)

In the literature [66], ***DVFS*** is one of the best known energy-efficiency techniques for improving power efficiency of chips at run-time [15]. We also use ***DVFS*** to establish optimized frequencies for cores and routers.

Chai et al. worked on a combination of execution time, ***DVFS***, slack time, and power consumption to find energy-efficient schedules with minimized processor frequencies [67].

An ***NoC*** provides the different network interconnection models between ***processing elements (PEs)*** through routers. It can permit hop-by-hop communications between ***PEs***. To serve and provide higher traffic demands in ***NoCs***, ***PEs*** and routers run at higher clock frequencies. Therefore, "power consumption grows rapidly and limits ***NoC*** scalability" [64].

The use of ***DVFS*** to increase power efficiency of the ***NoC*** can be implemented on different levels (i.e., at the router level or the NI level). However, in this work, the ***NI*** serves as a platform for handling DVFS by supporting different schedules provided by the MeS, making the routers simpler and with low overhead.

2.4.1.1.2. Distributed dynamic voltage and frequency scaling (DVFS) algorithm at the router and core level

Sharma et al. [68] propose the communication energy technique, similar to the optimal solution obtained by ***integer linear programming (ILP)***, as a low complexity heuristic mapping algorithm.

However, few platforms can support full stage ***DVFS*** at the router and interconnection level. For example, in $Xilinx\ ZYNQ-7000$, the input clock is $100\ MHz$ and the PL fabric clocks generated on the software side are equal to $85\ MHz$ and $170\ MHz$ [69].

Some simulations and models are proposed as a scalable power-gating technique for reducing energy consumption. For example, Farrokhbakht et al. [70] worked on a **scalable mapping and routing technique (*SMART*)**, which results in SPLASH-2 benchmarks that show it can decrease the static energy and average packet latency of the ***NoC*** network by 27.3% and 49.9%.

2.4.1.1.3. Worst case execution times (WCETs) and distributed dynamic voltage and frequency scaling (DVFS)

WCET is the maximum execution time of a task, which is pre-estimated or pre-computed and uses analytical or measurement-based techniques. ***WCET*** is an essential input for scheduling in ***TTA***.

Our approach is driven by certification requirements and is based on an off-line and static-scheduling algorithm to compute the optimal meta-schedules and dynamic ***SDFs*** for

more energy efficiency. It addresses the DS reclamation scheduling technique to minimize static and dynamic energy consumption.

In this work, we use slack distribution to achieve power efficiency with scenario-based schedules containing different $SDFs$ for ***DVFS*** in both cores and routers.

2.5. Energy management for different types of resources

2.5.1. Network-on-chip (NoC)

NoCs enable scalable communication for connecting several resources within a chip (e.g., routers). Energy consumption is primarily by computing (i.e., cores) and communication (e.g., routers) [71] and can contribute considerably to total chip power. For example, on-chip routers and links consume up to 18% in the Intel SCC and 20% in the Alpha 21364 processor and communication power takes 33% in RAW architectures [14].

Sheikh et al. investigated the combined optimization of ***performance, energy, and temperature (PET)*** [72] and propose an optimization framework. They assessed multiple cores for frequency switching.

Jingcao Hu et al. [73] used static schedules for communication and computation of tasks on heterogeneous ***NoC*** architectures for multimedia. Their algorithm achieved approximately 44% energy on average, compared to the standard earliest-deadline-first scheduler.

Various power saving techniques have emerged for ***NoCs*** at system-, architecture-, and circuit-level [74]. Experimental results indicate that slack algorithms can reduce energy consumption by 20∼50% [75] more than existing ***DVFS*** algorithms [61].

In [76], the experimental results indicate that a dynamic SDF, on average, has 10% energy gains over static models.

In [73], the authors provide energy-efficient scheduling with static schedules for both communication activities and computational tasks in heterogeneous ***NoC*** architectures, under real-time systems constraints and for a multimedia application.

The algorithm achieves a 44% energy reduction, on average, compared to the standard earliest-deadline-first scheduler. In addition, scenario-based scheduling methods are presented.

In [77], the authors identify a relationship between execution time, ***DVFS***, slack, and energy consumption, achieving energy-efficient scheduling with lower processor frequencies. They report that 28% ∼ 36% of the total chip power consumption depends on ***NoC*** energy consumption (e.g., on-chip routers and links).

In [78], it is observed that the low power technique reduces energy consumption in an average ***NoC*** by 49%, with negligible effects on network bandwidth and delays.

In [79], the authors used ***voltage/frequency scaling (VFS)*** and propose a power-efficient network calculus-based technique to minimize the power consumption of the ***NoC***, which can save up to 50% of the total power consumption.

Han et al. [80] present a low power methodology and routing algorithm for temperature to achieve an ultra-low-power ***NoCs***. The experimental results demonstrate an average power reduction of 36% over 21 applications.

Li et al. [81] propose a two-step solution for the problem of energy-efficient mapping and scheduling in ***NoCs***. They used quadratic binary programming to minimize the communication energy and a genetic algorithm to minimize the overall system energy consumption.

2.5.2. Processor

Lee et al. [61] [82] worked on a method of DVFS-enabled multi-cores to reduce energy consumption by executing the tasks in parallel on a sufficient number of cores and assigning the lowest possible frequencies. Their scheme saves up to 67% of the energy when executing the task on a single core. However, it does not support scenario-based scheduling and communication frequency scaling.

Related work indicates that slack-algorithms for DVFS [61] are valuable methods of reducing energy consumption by 20~50% [75]. In [76], the results indicate that a dynamic $slowdown$ for cores, on average, results in 10% energy gains over static models. In [13], the results show that a DS algorithm for cores, compared to the SS, produces a maximum of 64.4% energy savings in a single schedule and 41.61% energy savings on average.

Hangsheng et al. found that the communication interconnect can consume up to 36% of the power in an ***MPSoC*** [9].

Also, compared to the related works in case of energy, we use optimization techniques for adaptive routing and frequency scaling in cores, routers, and links which can potentially improve ***MPSoCs'*** energy-efficient performance. The potential of DVFS for the router is interesting and our ***SBMeS*** could extended for this.

2.5.3. Task procrastination and slack reclamation

Using reclaim the SS or DS in traditional real-time systems is not a new practice (e.g., Sprunt et al. [83], Strosnider et al. [84]).

Chatterjee et al. [85] take the slack time [76] of tasks into account to improve deadline satisfaction and reduce energy consumption. They propose an algorithm for fault-tolerant resource allocation in real-time dynamic scenarios. On average, they reduced energy consumption by 29.1% and 6.7%, compared to previous work.

In [76], static and dynamic task procrastination is introduced as a slack-reclamation technique for energy reduction over the static task SDF. Their simulation found that dynamic SDF results on an average of 10% energy gains over the static SDF.

Dynamic procrastination reduces idle energy consumption by 15%, while meeting all timing requirements. Mahapatra et al. [86] propose an energy-efficient slack distribution technique for multimode distributed real-time systems. Their work provides a network traffic monitoring technique and a slack distribution mechanism to achieve power efficiency. A co-simulation framework was used to evaluate their proposed technique.

The relationship between execution time and energy consumption for energy-efficient scheduling is presented in sections 3.9 and 3.12, and this shows how ***DVFS*** is used for slack times between tasks to save energy with lower processor frequencies.

2.6. Scheduling for time-triggered systems (TTS)

Message and task scheduling are problems for ***MPSoCs*** that are solved using TT scheduling. In task scheduling, the scheduler must decide when and where a task is to be executed, using each cove in every time slot.

The static TT schedule determines that TT messages through which link must be sent over the network at the predefined timing, deadlines and how the messages reach their destination before a global timeout.

In this work, message and task scheduling are related, and we focus on these aspects of the system because these problems and their associated constraints (energy-efficient, reliability, and fault) are an interesting component of previous associated works, which have produced valuable and positive results [85, 87].

2.6.1. Algorithms for static

Nowadays, many different techniques, algorithm, and methods are using for synthesis and generating the TT schedules (e.g., GA, constraint programming (CP) [88] ILP, MIQP, and satisfiability modulo theory [89]).

In static or pre-runtime scheduling, a feasible schedule for a set of tasks is calculated off-line and pre-compiled [17] which was generated before run-time. The scheduling technique has to guaranty all deadlines, considering the resource, precedence, synchronization requirements for all tasks and conditions [17]. In scheduling the precedence relations between the tasks and messages, executing in the different nodes and events can be depicted in the form of a precedence graph [17]. For scheduling a problem, "*a solution can be detected as a finding a path, a feasible schedule, in a search tree by applying a search strategy* [17]" Figure 5.

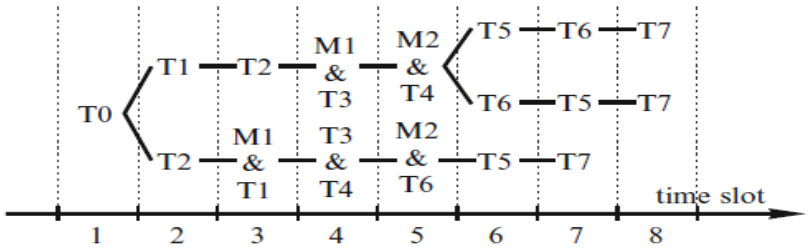


Figure 5. A search tree for time slot [17]

Static-scheduling is a reliable solution for robust and timely systems, especially in *TTS*. Their scheduling problem was established by formulating a *mixed-integer linear programming* (*MILP)* problem.

Murshed et al. [90] have provided a static message-based scheduling approach that guarantees the absence of collisions in message routing for a single task per core. Their solution of the MILP problem is the allocation of tasks to cores and scheduling the messages between these allocated tasks while minimizing the critical path delay.

One of the Boolean problem [87] in scheduling is a synthesis of ***TT*** schedules; namely, when a schedule calls the correct schedule, it must meet all its requirements, otherwise it is incorrect. However, the other problem relates to the quality definition, with finding, choosing, and rating high-quality schedules being an interesting problem for new research. However, the "quality" is a complex and varied concept that can change quickly, according to context.

Wang et al. [91] worked on heuristic static task scheduling for performance yield maximization in ***MPSoC***. This assigns tasks to cores, working from the root node to the end node of the task graph until all tasks are scheduled.

Mirzoyan et al. [92] worked on heuristic variation-aware task scheduling methods, using data flow graphs. This was limited to real-time streaming applications, with a ***synchronous data flow graph (SDFG)*** [93] model that does not support fault-tolerant and energy-efficient.

2.6.1.1. Static scheduling for energy efficiency

Energy efficiency, energy management, and energy reduction are essential issues in embedded systems. Many real-time systems must adapt to run-time critical events, such as faults inside or outside the system and environmental conditions.

The authors of [22] worked on an algorithm for task scheduling and time constraints where task deadlines were satisfied, saving more energy without degrading performance.

The authors of [73] developed energy-aware scheduling, search, and repair, with their model guaranteeing the prevention of missed task deadlines. This is also considered in this work.

2.6.1.2. Static scheduling for reliability

The use of the ***TT*** scheduling method for embedded systems is compelling because it provides predictability and greater safety and reliability. Some related mode switched scheduling techniques are proposed in the literature (e.g., Burns and Davis [33], Hu et al. [34], Baruah et al. [94], Burns and Baruah [95], and Ekberg and Yi [96]).

Guy Avni et al. [87] worked on ***TT*** schedules for switched networks with faulty links. They addressed the problem of the resistant schedule and propose how one can use the fixed error-recovery protocol to guarantee messages arrive at their destinations, regardless of which links failed. They focused on two ***TT*** scheduling limitations. The first is the computational constraints and variables complexity of synthesizing, named an ***NP***-complete problem [97].

2.6.2. Meta-scheduling (MeS) and mode changes

Most ***TT*** systems and ***TTN*** schedules are generated offline (static scheduling), but when the application or hardware model changes, most schedules cannot support events and their recent change. For example, since the systems using ***TTN*** or ***TTS*** are usually safety-critical, events and changes caused by the link, router, and core failures complicate scheduling tasks and messages [98].

Quasi-static scheduling supports adaptive behaviors to reduce system costs when a limited number of predefined cases in scenario-based systems occur. The quasi-static scheduling technique is restricted to dynamic reactions and activities, while having very limited scheduling overhead.

Most of the research on meta-scheduling is conducted for enterprise grids, clouds, and data centers; for example, GridWay,[2] community scheduler frameworks,[3] Moab cluster suite,[4] Maui cluster scheduler,[5] DIOGENES, and synfiniWay's meta-scheduler [11].

However, Jung et al. [99] worked on meta-scheduling for green computing to reduce energy consumption and thus reduce CO_2 emissions into the atmosphere. This approach is known as "GreenMACC." Following many rule changes in the EU concerning green and pure energies, and aimed to benefit the environment by reducing fossil fuels and controlling greenhouse gas emissions (e.g., CO_2 [100]). To this end, for example, *BMW*, *Bosch*, and *Continental* invested in battery cell [101] and most carmakers plan to invest heavily in research and development in the domain of e-cars, which depends significantly on embedded systems [4].

In [102], Fohler presents a method for supporting schedule changes based on operational modes, by switching and traversing static-schedules in a schedule status tree.

Jung et al. [103] worked on ***synchronous data flow***, which is commonly used in signal processing or streaming applications. Their model can be used for dynamic behavior

[2] http://www.gridway.org
[3] http://toolkit.globus.org/toolkit/docs/4.0/contributions/csf/
[4] http://www.adaptivecomputing.com/
[5] http://www.adaptivecomputing.com/products/maui/

changes and classified as a multi-mode dataflow model. The proposed technique minimizes the number of required processors for multiprocessor scheduling by considering task migration between modes. The focus is on minimization of required resources, but not the parameterization of resources, such as frequency scaling.

2.6.3. Optimization techniques for scheduling

The basic structure of the scheduling technique for ***TT*** communication in ***NoCs*** is built by formulating an ***MILP*** problem, as in [90], and extended to ***MIQP*** in this work. It consists of constants, decision variables, constraints, and an objective function, which are modeled by a set of ***MIQP***-compatible equations. The ***MILP*** and ***MIQP*** problems are solved using the ***IBM*** optimizer.

Tariq et al. [46] investigated the problem of scheduling and energy-aware mapping on a heterogeneous ***NoC*** base.

Anup Das et al. [104] propose an ***ILP*** to reduce communication energy, fault-tolerant migration overhead, and tasks remapping.

Ishak et al. [105] worked on ***nonlinear programming (NLP)*** for computing and identifying the optimal frequencies for heterogeneous *NoC*-based *MPSoCs* for all tasks and communication links for an ***ILP***- based algorithm. Their objective is to assign an optimal frequency for each task and each message which the total energy consumption of all the tasks and messages is minimized.

2.6.3.1. Quadratic Optimization

In [106], *quadratic optimization (QP)* is described as an essential mathematical area of ***NLP***. It is used in most problems, including those in planning, scheduling, economics, engineering, and routing.

2.6.3.2. Mixed integer quadratic programming (MIQP)

In [107], *MIQP* is used to solve scheduling problems. Here, the objective function is quadratic, concerning the integer and continuous variables, while the constraints are linear with respect to the variables of both types.

2.6.3.3. Genetic algorithm (GA) and optimization techniques

Li et al. [81] worked on a ***genetic algorithm (GA)*** to achieve near-optimal voltage and frequency assignments to address the problem of energy-efficient contention-aware application mapping and scheduling in ***NoCs***. Their reported results indicate that jointly utilizing dynamic voltage scaling for processors and frequency tuning ***NoC*** links provides excellent potential for overall energy reduction in ***MPSoCs*** and overall system energy consumption is significantly reduced.

Optimum solutions provide better solutions than feasible answers or other methods, such as ***GAs***. Majd et al. [108] used ***GAs*** as they believed these to better locate a near optimal

solution than a list schedule. They report that many existing approaches do not consider communication costs when applying ***GAs*** to ***MPSoC*** scheduling problems. In contrast, we do consider the communication costs. To handle the communication delays between processors, Majd et al. used a combination of ***GA*** and the imperialist competitive algorithm [109], while we use ***MIQP*** together with the above mentioned linearization method.

At the cost of higher runtime, the ***MIQP***, ***MILP***, and ***MIP*** methods find better solutions than heuristics such as ***GAs*** [110].

2.7. Scenario-based scheduling in event-triggered systems

Many published articles concern task scheduling with limited resources [21]. However, few works focus on scenario-based scheduling.

SBMeS is a chain solution for different scenarios of each event in the system (e.g., fault, slack), providing unique schedules and processing functions at the desired throughput. Scenario-based schedule chains are required to computationally process large volumes of constraints and variables, sometimes within very short periods, to facilitate real-time application needs.

Using ***SBMeS*** to provide the desired throughput of reliable schedules may lead to the violation or reduction of other services (e.g., frequency scaling). Hence, in an embedded system achieving deadline, safety and reliability of ordered schedules are of paramount importance. Therefore, fault-tolerance, reliability, and energy-efficiency are the three most desired features of ***MPSoCs***, allowing the scenario-based scheduled system to adapt to changes in the environment.

The ***SBMeS*** model is based on per-core and router ***DVFS*** with multiple frequency supply networks. We use ***DVFS*** on both cores and routers and propose the use of ***non-minimal path adaptive routing*** [111] to balance network traffic. Specifically, the ***MeS*** scheduler regularly tunes the operation frequency of the routers and the cores, based on the scheduling-scenario and ***NoC*** workload. In addition, the ***MeS*** takes into consideration the router and cores utilization, in conjunction with the tasks slack allowance.

SBMeS requires attention to avoid cascading fault-events collision and to make valuable trade-offs between model constraints (e.g., fault, energy) and event controller and system failure protection in schedules, especially for the real-time operating system and software applications across the ***TT*** embedded-system that cooperate with the critical systems (e.g., aerospace, medical, and automotive).

In [112], the authors introduce quasi-static schedules of data flow graphs in the presence of limited channel capacities. However, the results do not guarantee the preservation of deadlock freedom.

In [113], Wei et al. worked on fault-tolerance in hard real-time systems and showed how quasi-static task scheduling algorithms can support fault-tolerance and energy efficiency for hard real-time systems.

Most existing works address the issue of energy optimization, with separate consideration of the dynamic or SS, quasi-static scheduling for single cores in the context of DVS, DFS, and DVFS.

This work proposes the ***SBMeS*** method, which is more flexible and extendable than other quasi-static techniques for a wide range of applications (e.g., energy minimization of real-time multitasking systems). This technique can exploit and integrate other scheduling problems in the quasi-static application domain. The performance is superior to that obtained by the previously proposed dynamic and scenario-based approaches.

This work develops an energy-efficient scheduling algorithm for DS and ***slowdown factors*** in ***multi scenario-based systems*** for both cores, specifically using ***SBMeS*** for static scheduling on *adaptive* ***TT MPSoC*** and ***NoC*** systems.

2.7.1. Reliability and redundancy in scenario-based meta-scheduling (SBMeS)

In [29], reliability is defined as the conjunction of accurate service delivery, “up-time.” In other words, reliability in a real-time system is a timing operation and the interval of time depends on the system design.

In ***TT*** networks, the application layer supports redundancy failures [114]. ***SBMeS*** can support multi-link and multi-topology for redundancy and non-collision messaging schedules. Here, a task and message should be scheduled on different cores and paths, and the defined core and paths should be disabled and removed from operational tables to prevent system crash, abnormal behavior, and common-mode failures. These rules and solutions are simplistic and used in problem formulae.

Huang et al. [115] worked on scheduling for energy-efficient and mixed-criticality. They use DVFS to speed up and slow down tasks [28].

2.7.2. Scenario-based meta-scheduling (SBMeS) for reconfigurable systems

Reconfigurable computing and systems are considered a hi-tech, effective solution combining the flexibility of traditional processors and systems with the high processing efficiency of ASIC. A reconfigurable architecture can meet the requirements of different task processes through different system structures, using reconfigurable devices [116]. It is vital that, during hardware reconfiguration, other hardware tasks continue to work without interruption. On the other hand, since network topology can change intentionally (e.g., reconfiguration purposes) unintentionally failures (e.g., core or router) these changes are not non-maskable.

Our proposed ***SBMeS*** method can use reconfigurable processor architecture [117] when the platform model begins to change during the runtime.

Cai et al. [116] worked on a hybrid-scheduling algorithm and technique for reconfigurable processor architecture and grouping tasks, devices, and processors to general and reconfigurable elements. ***SBMeS*** compare to their work can use in all reconfigurable processor architecture without any dependency on system design. This means that hardware and software changes can be scheduled without harmful effects on system safety or reliability.

2.7.3. Scenario-based meta-scheduling (SBMeS) for robust systems

A robust system can work well even when unexpected (reasonable) platform changes occur. ***SBMeS*** can be used in robust-systems to predict environmental change in terms of event context and scenarios. It can also control small disturbances in the environment which can cause even small system errors. Hence, ***SBMeS*** can serve robustness and predictability needs as two "universal challenges" in the domain of embedded systems design.

Eitschberger [28] worked on "Energy-efficient and Fault-tolerant Scheduling for Many-cores and Grids." His primary objective was integrating fault-tolerance into schedules and minimizing energy consumption using ***DVFS*** based on tasks. We addressed reuse and extended some parts of his model (e.g., energy, fault) to task and communication scheduling. We also added ***DVFS*** to communication and integrated them into ***SBMeS***.

2.8. Overview of scheduling-related works

An overview of related work on scheduling is presented in Table *1*, summarizing the previous works and techniques covered in this work.

Table 1. An overview of related works compared to scenario-based meta-scheduling (SBMeS)

Related work	***Techniques***												***Requirements***				
				Energy			***Optimization***		***Memory management***		***Platform***						
	Meta, Super, hybrid scheduling	Visualization	Core Fault	DVFS	Cores / Tasks	Routers	MILP/ GA/ ILP	MIQP	Messages	Tasks	Time-Triggered system	NoC	MPSoC	Real-time system	Fault-tolerant	Energy-efficient	Mixed-criticality
Eitschberger [28]	√	-	√	√	√	-	√	-	-	-	-	-	√	-	√	√	-
Cai et al. [116]	√	-	√	-	-	-	√	-	-	-	-	-	√	-	-	-	-
Murshed [90]	-	-	-	-	-	-	√	-	-	-	√	√	√	√	-	-	-
Wang et al. [91]	√	-	-	-	–	-	√	-	-	-	-	-	√	√	-	-	-
J. Hu et al. [73]	-	-	-	-	√	√	√	-	-	-	-	√	√	√	-	√	-
Tariq et al. [46]	-	-	-	√	√	√	√	-	-	-	-	√	√	√	-	√	-
Guy Avni et al. [87]	-	-	-	-	-	-	√	-	-	-	√	-	-	√	√	-	-
Jung et al. [103]	√	-	-	-	-	-	√	-	-	-	-	√	√	√	-	√	-
Fohler [102]	√	-	-	-	-	-	√	-	-	-	√	-	-	√	√	-	-
Anup Das et al. [104]	√	-	√	√	√	√	√	-	-	-	-	-	√	√	√	√	-
Ishak et al. [105]	-	-	-	√	√	√	√	-	-	-		√	√	√	-	√	-
Li et al. [81]	√	-	-	√	√	√	√	-	-	-	-	√	√	√	-	√	-
Majd et al. [108]	-	-	-	-	√	√	√	-	-	-	-	√	√	-	-	-	-
Huang et al. [115]	-	-	-	√	√	-	-	-	-	-	-	-	-	√	–	√	√
J. Falk et al. [112]	√	-	-	-	-	-	√	-	-	-	-	-	√	-	-	-	-
Maleki et al. [25]	–	-	√	√	√	-	√	-	√	-	√	√	–	√	√	√	√
This Work	√	√	√	√	√	√	-	√	√	√	√	√	√	√	√	√	√

2.9. Summary

In summary, the existing work on ***SBMeS*** and low power and energy-efficient scheduling focuses on different models, methods, and architectures to the current study; for example, with or without ***DVFS*** or dynamic power management capabilities and attempting to dynamically or statically manipulate the task execution slacks to exploit them.

While our work is related to [73], we cover not only SS time but also support DS time and scenario-based scheduling. We use slack time of tasks to calculate the best ***slowdown factor*** for the communication and computation to maximize energy efficiency

2.10. Visualization of schedules

To better understand the complex schedule results, developers and system designers need abstract graphical visualizations of their schedules. Effective evaluation and validation methods and tools for tracing and viewing schedule outputs save time and reduce costs.

An overview of the existing tools and a short overview of related work is given in Table 2. Most focus on simulations to generate the visualizations [7]. Some tools, such as Ghost, FORTISSIMO, ARTIST, and RTSSim, support neither shared resources nor multiple cores, which are both necessary features in adaptive ***TT MPSoC*** and ***NoC*** [7].

Table 2. Overview of existing real-time visualizations tools [7]

Project	Simulation	Visualization	"live" simulation	Shared resources	Multiple Cores	Sporadic tasks	Open-source	Programming Languages
Alea2	_	√	√	_	√	√	√	java
ARTISST	√	√	_	_	_	√	√	C++
Cheddar	√	√	_	√	√	√	√	Ada
FORTISSIMO	√	_	_	_	√	√	√	C++
gltraceviz	_	√	_	√	_	√	√	C++
CHOST	√	√	_	_	_	√	_	C
Jedule	_		_	_	√	√	√	java
MAST	√	√	_	√	√	√	√	Ada
Paje	_	√	_	√	√	√	√	Objective C
Realtss	√	√	_	√	_	_	√	TCL
RTsim	√	√	_	√	√	_	_	*unknown*
RTSIM	√	√	_	√	√	√	√	C++
RTISSim	√	√	_	√	_	√	_	C
Schesim	√	√	_	√	√	√	√	Ruby
STORM	√	√	_	√	√	√	√	java
STRESS	√	√	_	√	√	√	_	C
ViTE	_	√	_	√	√	√	√	C++
VizzScheduler	√	√	√	_	√	_	_	java
MeSViz	-	√	-	√	√	√	√	C++

In this work, ***MeSViz*** is designed and implemented for visualization schedules according to adaptive ***TT***, ***MPSoC***, ***NoC***, and real-time requirements.

Chapter 3: System model

The problem of scheduling ***TT*** communication in ***NoCs*** is discussed in this section. The scheduling problem distinguishes two models; namely, a physical model and a logical model. The physical model comprises the on-chip resources, including the cores, the routers, and their connectivity, via the ***NoC***. The logical model defines tasks and their dependencies based on exchanged messages.

3.1. Overview of the models

In this work, four models are used to design a comfortable, fast, and accurate scheduling model.

The inputs models for ***SBMeS*** are the application model (AM), physical model (PM), schedule model (SM), and context model (CM). AM has constants of the application (e.g., number of tasks, WCET of each task, precedence constraints). PM has constants of the platform (e.g., number of nodes, links between nodes). SM has values for decision variables (e.g., allocation to a node, start time of execution); and CM has fault and events details (e.g., event type, event execution time).

In the following section, we briefly describe ***MeS***, which optimizes the placement of tasks and messages to optimize execution times and injection times. ***MeS*** is improved by adding the new optimization model described in section 6.3.

1. ***Dependability requirements*** covered by ***MeS*** for scenarios concern detecting, isolating, or mitigating errors and transient physical problems (e.g., core fault technique in the section 4.6) that occur in systems or schedules.
2. ***Timing requirements*** covered by ***MeS*** concern time constraints, execution time determinism, predictability, composability, and flexible worst-case response for slack time by static analysis.
3. ***Independence of network topology****:* the ***MeS*** scheduling and routing method is independent of the hardware platform and can cover the topology of different networks (e.g., mesh, direct network, indirect network, balanced tree).

3.2. Input models for a meta-scheduler (MeS)

3.2.1. The physical model (PM)

The PM comprises the physical dependencies at the hardware layer. Figure 6 shows a platform example and describes the nodes and links the priorities, dependencies, and hierarchy that the system designer uses to shape the PM section of the input file.

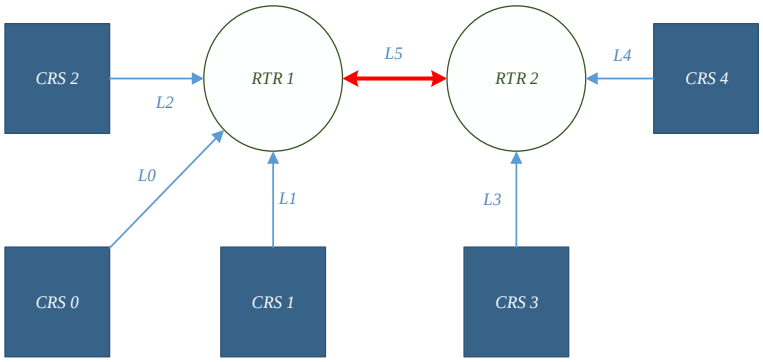


Figure 6. Conceptual physical model (PM)

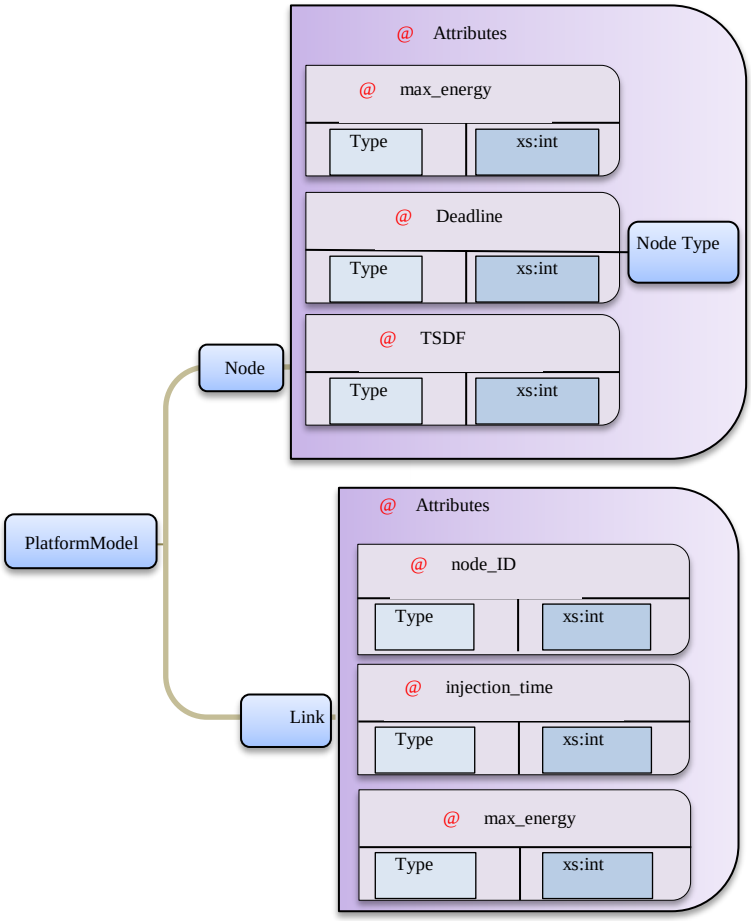


Figure 7. General physical model (PM) schema

The PM has two main elements: nodes and links. Figure 7 highlights that each element contains specific attributes with built-in derived type (e.g., the node contains an ID, type (router or core) and frequency and the link contains an ID from and to the node).

3.2.2. The application model (AM)

The AM concerns application dependencies at the software layer. It presents the task and message priorities, dependencies, and hierarchy that the system designer uses to shape the AM section.

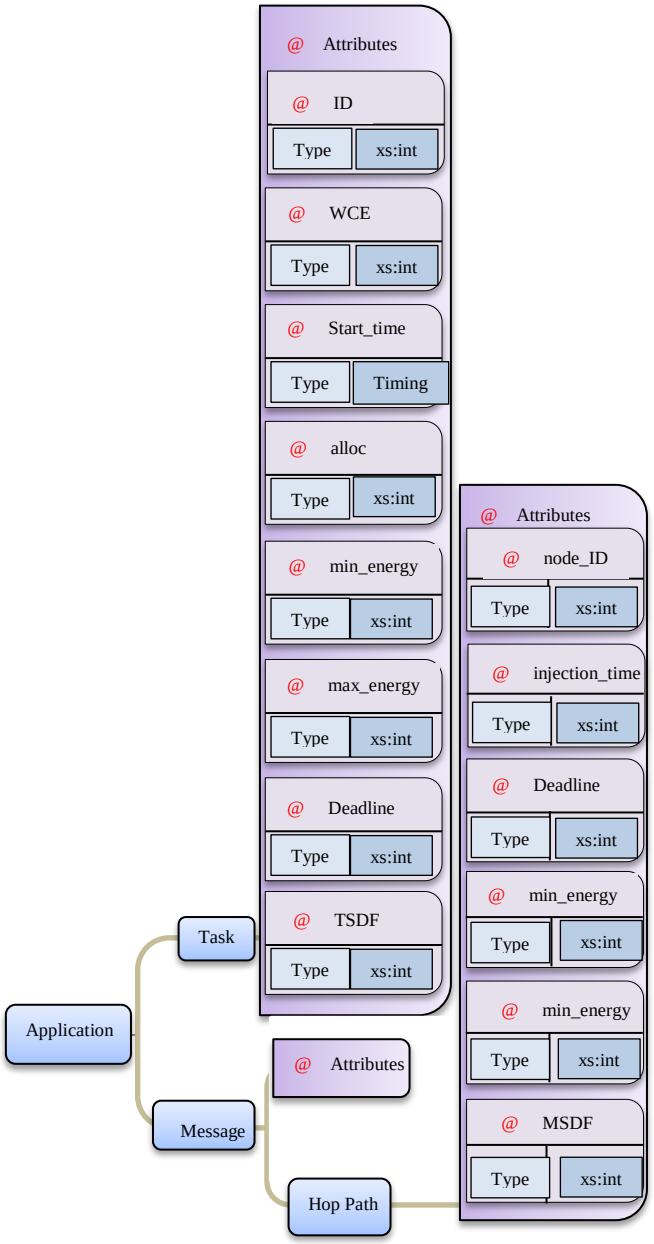


Figure 8. General application model (AM) schema

The AM has two main elements: tasks and messages. Figure 8 highlights that each element contains specific attributes with built-in derived types (e.g., task contains an ID, WCET, start time, allocation, min and max energy, deadline, and SDF, and message contains a message ID, sender and receiver ID, SDF).

3.2.3. The context model (CM)

CM concerns the possible scenarios in the system management layer and can identify when single or multi faults and events occurred. The CM describes the faults and events for each element in the PM and the AM, including priorities, dependencies, timing, and hierarchy. CM has three main elements: slack, energy, and fault. Figure 9 shows that each element contains specific attributes with built-in derived type (e.g., slack event contains the new execution time and related task ID, energy event contains energy level, and fault

including node, link, and core fault, which they are represented by a related object ID e.g., link ID).

In this work slack, fault and energy are used as input scenarios in the CM, which is covered by the ***MeS***.

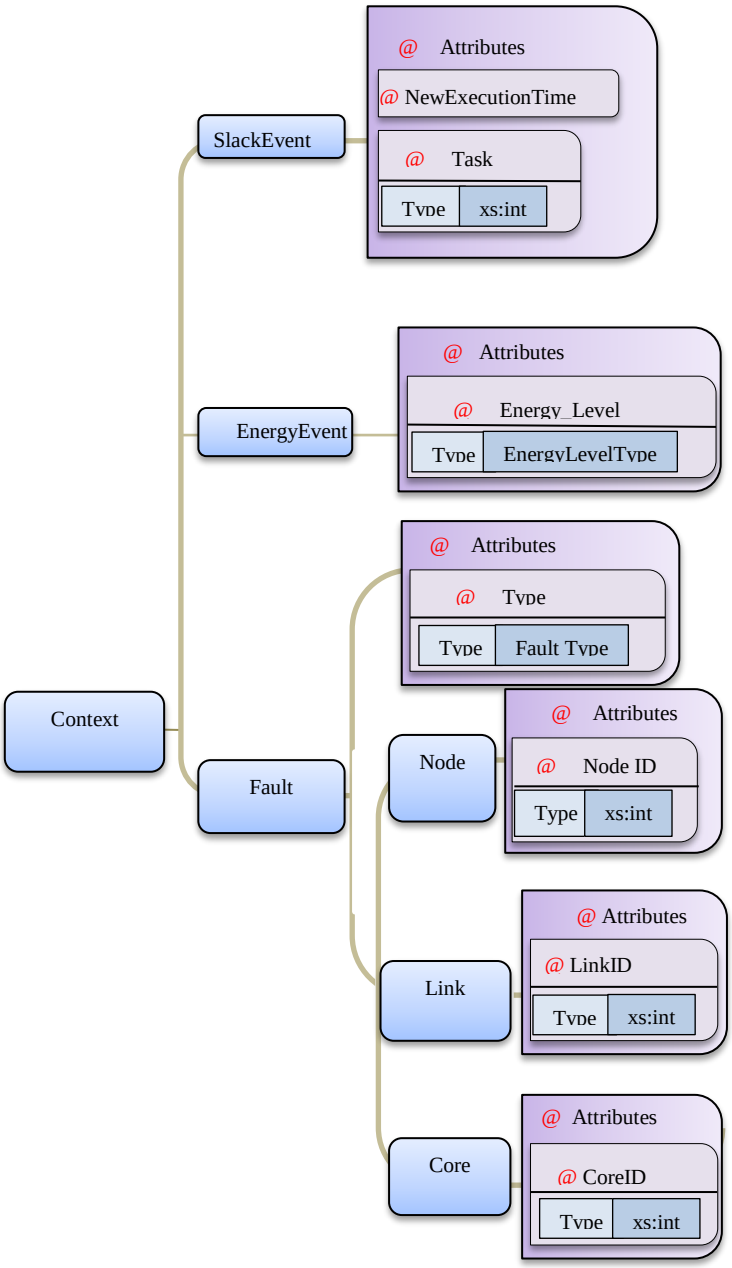


Figure 9. General context model (CM) schema

3.2.4. The schedule model (SM)

MeS stores all generated schedules information in a schedule tree. Each schedule is saved in one node. These elements are used as input data in the ***MeSViz*** visualization.

The *SM*, as shown in Figure 10, contains specific data structures, which are the PM, AM, and CM.

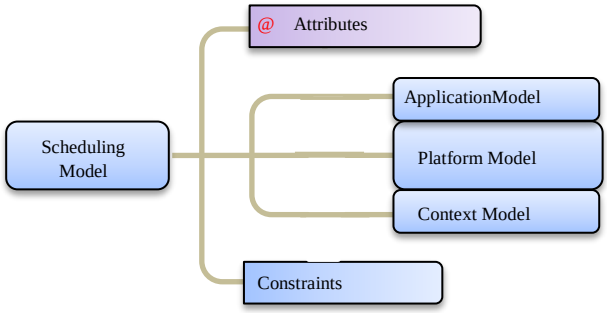


Figure 10. Meta-scheduler (MeS) data structure schema model

3.3. Modelling of the optimization problem

This work provides an extended model of that used in [12], [13], and [16], and in the following, we present the new respective parameters.

It is essential to model the relationships between physical, logical, and application constraints, without topology dependencies, along with the resulting impact on performance. System optimization is the process of utilizing these models to search for optimum or feasible solutions that best match the physical, logical, and application constraints of the implementation. For a given implementation method, the physical constraints characterize architectural aspects; but in our model, these features are free of network topology. e.g., one of the physical constraints facing the implementation of interconnection networks is the available links between two nodes.

3.4. Choice of optimization technique

Effects of each scheduling algorithms and techniques (e.g. Mathematical, *Artificial* Intelligence, *Scheduling* Heuristics, *Neighborhood search* [6]) on systems have to be analyzed, understood and defined where a failure would lead to severe consequences [7]. For instance, we can use the popular *MILP*, *MIQP*, or quadratically constrained quadratic program *(QCQP)*-based optimization [115] and *mixed-integer quadratically-constrained quadratic program (MIQCQP)* [118], as our objective function by default.

A quadratic objective function and equations with linear constraints in a *MIQP* problem are included. Also, some of the variables in the equations are constrained to other values, e.g., integer values [119]. *MIQP* is a well-known *NP*-hard problem [120]. Some optimizer tools, such as *CPLEX*, *MATLAB*, and *Gurobi* can help to speed up the computation process in *MIQP* problem-solving.

This work differs from existing and previous works in three important aspects. First, we address the problem of both task mapping and task scheduling. Second, we work and present a model, where *DVFS* for cores and frequency tuning for *NoC* routers can be combined together for achieving optimal system energy consumption; therefore, the communication and/or computation dynamic and static slacks of tasks and messages can be shared and distributed efficiently and optimally to achieve more energy saving via overall energy reduction. Third, using *SBMeS* technique to solve cores crash and

controlling their action on system topology, schedules, and energy-efficient is the particular goal and challenge of this work.

3.5. Static scenario-based meta-scheduling (SBMeS) and mapping policy

Our static-scheduling policies are designed for fault-control and to improve energy-efficiency on two parallel levels: the first level is the mapping of resources (e.g., cores to tasks, routers, and paths to messages) and the second uses $SDFs$ for frequency tuning in both cores and routers. In addition, using ***SBMeS*** and injecting to the static scheduling system dynamic slack for each task and faults for each core can create the full range of optimized schedules for energy consumption'. Figure 11 shows the basic structure of our ***SBMeS*** systems.

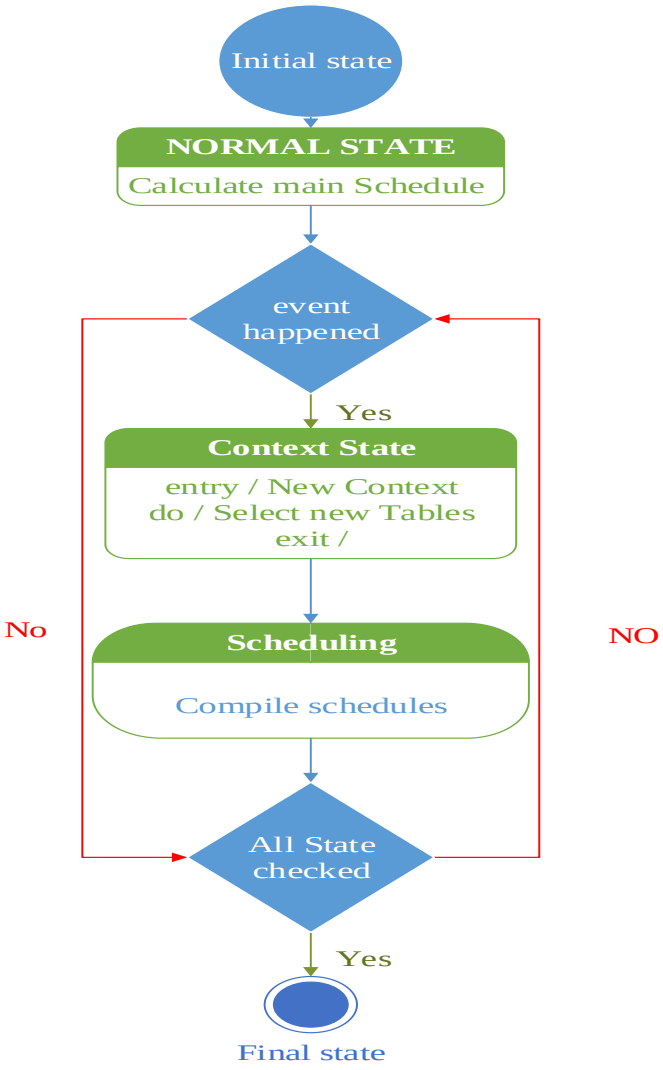


Figure 11. The basic model of meta-scheduling (MeS)

3.6. Decision variables, constants, and constraints

In this part, the constraints and variables used in CPLEX for ***SBMeS*** are introduced.

3.7. Definitions

For an ***MeS*** solution, one must design a scenario and define decision variables, constants, and algorithms. In this work, most of the decision variables are common between case studies, where they are used as input control parameters. ***MeS*** must allocate, build, and calculate connectivity and dependencies between cores, router, tasks, messages, hops, and events to generate a Gantt map and graph mapping (e.g., Table 3).

Table 3. Overview of input table [90]

Domain	Constant name	Type	Description
PM	N_{crs}	$\mathbb{N}$	Number of cores
	N_{rtr}	$\mathbb{N}$	Number of routers
	N_{node}	$N_{crs} + N_{rtr} \in \mathrm{N}$	Numbers of nodes
	$ALLOC$	$\begin{bmatrix} alloc_1 \\ \vdots \\ \vdots \\ alloc_k \end{bmatrix} \in \{1, \dots, \lvert CRS \rvert\}^k$ $k = \lvert TSK \rvert$	Allocation of tasks/tasks to cores (also routers, since one core connected to one router)
	$HOPS$	$\begin{bmatrix} hops_1 \\ \vdots \\ \vdots \\ hops_n \end{bmatrix} \in \{1, \dots, MaxHop\}^n$ $n = \lvert MSG \rvert$	Number of hops for each message
	$Path$	$\begin{bmatrix} p_{11} & \cdots & p_{1n_r} \\ \vdots & \ddots & \vdots \\ p_{n_m 1} & \cdots & p_{n_m n_r} \end{bmatrix} \in \{1,2, \dots, n_r\}$	Each row is the path of a message, the maximum number of hops (columns) given by the number of routers
	N_{tsk}	$\mathbb{N}$	Number of tasks
AM	N_{msg}	$\mathbb{N}$	Number of messages
	WECT/ET	$\begin{bmatrix} et_1 \\ \vdots \\ et_{n_j} \end{bmatrix} \in \mathbb{N}^{n_j}$	Task worst case execution time
	MD	$\begin{bmatrix} md_1 \\ \vdots \\ \vdots \\ md_n \end{bmatrix} \in \{1, \dots, Max\}^n \quad \lvert n = \lvert MSG \rvert$	Duration of message transmission (depending on message length)
	SRC	$\begin{bmatrix} s_1 \\ \vdots \\ s_{n_m} \end{bmatrix} \in \{1,2, \dots, n_j\}^{n_m}$	Source task of messages
	$DEST$	$\begin{bmatrix} d_{11} & \cdots & d_{1n_j} \\ \vdots & \ddots & \vdots \\ d_{n_m 1} & \cdots & d_{n_m n_j} \end{bmatrix}, d_{ij} \in \{0,1\}$	Destination task of messages
	CON	$\begin{bmatrix} c_{11} & \cdots & c_{1n_r} \\ \vdots & \ddots & \vdots \\ c_{n_r 1} & \cdots & c_{n_r n_r} \end{bmatrix}, c_{ij} \in \{0,1\}$	Connectivity of routers
	$TSDF$	$[tsdf$	SDF of all tasks

		$k = \|TSK\|, \quad tsdf_{min} \geq 1$	
	$MSDF$	$\begin{bmatrix} msdf_1 \\ \vdots \\ \vdots \\ msdf_n \end{bmatrix} \in \{msdf_{min}, \dots, msdf_{max}\}^n$ $n = \|MSG\|, \quad msdf_{min} \geq 1$	SDF of all messages
SM	SSM	$sm \in$ SSM	set of all scheduling models

The following definitions are used to describe our optimizations model:

CRS: the set of cores (i.e., execution nodes of the platform)

RTR; the set of routers (i.e., message forwarders of the platform)

TSK: the set of tasks to be executed

MSG: the set of messages to be sent, from all tasks

$MSG_{IN}(t)$: the set of input messages of a task t

$MSG_{OUT}(t)$: the set of output messages of a task t

$et(t)$: the execution time of a task t

FLT: the set of faults to be handled

$md(m)$: the message transmission duration of message m from node A to B. $md(m)$ is equivalent to the size of the message m. Assuming that there are *hops(m)* intermediate nodes between the source and destination of m, the total message duration becomes the following:

$$md_{total}(m) = md(m) \cdot (hops(m) + 1) \qquad (1)$$

$t_{inject}(m)$: the time at which the sending of a message *m* began

f: clock frequency, with f_{max} being the maximum clock frequency and f_{sel} being the selected clock frequency:

$$f_{sel} \leq f_{max} \quad (2)$$

3.7.1. Collision avoidance constraint

These constraints ensure that only one message is transmitted for any link between two cores at a given point in time. "*A message can visit the link between two cores, A and B, only if there is a direct connection between them. Each core or router, must receive only one message at a specific time from its directly connected core* [24]". If a certain link must transmit more than one message, these messages should be sent at disjointed time intervals.

This work introduces SDF for both tasks and messages, which has a direct effect on the message collision avoidance.

3.7.2. Connectivity constraints

These constraints use the connectivity constants to consider the network path topology. They are executed so that routers may reduce the number of them in the model (e.g., cores, routers, and links).

3.7.2.1. Links reliability

The ***SBMeS*** technique for routing is independent of the platform network topology, hence it can support most network topologies (e.g., mesh, direct network, indirect network, balanced tree). It can therefore be used for more reliable systems. To achieve this, multi-links between cores and routers are stored in matrix CON, which the $c \in CON$ denotes the connection of the link.

3.7.3. Task allocation and assignment constraints

Tasks are not scheduled on the ***NoC*** but rather on cores. For tasks to be assigned only to cores, and not to routers, the constraints check the allocability constant *ALLOC*, with cores of a value of 0 not allocated tasks. The allocated cores are stored in the allocation array $alloc$.

We use $ALLOC$ to model the allocation of each task $t_i \in TSK$ to one of the cores in CRS:

$$ALLOC = \begin{bmatrix} alloc_1 \\ \vdots \\ \vdots \\ alloc_k \end{bmatrix} \in \{1, \dots, |CRS|\}^k$$
$$k = |TSK| \qquad (3)$$

In sections 7.3.2 and 7.3.3, each core can have only one task assigned, while in other cases (c.f., section 7.3.4 and section 7.3.5), each core can have multiple tasks assigned.

3.7.4. Message duration

MD is a vector with all the message durations $md(m)$ from one hop to the next of all the messages $m \in MSG$:

$$MD = \begin{bmatrix} md_1 \\ \vdots \\ \vdots \\ \mathrm{md_n} \end{bmatrix} \in \{1, \dots, Max\}^n \quad | \, n = |MSG| \qquad (4)$$

In other words, MD is the duration of the message transmission (depending on message length) from one *hop* to the next.

3.7.5. Path and visited cores

Path P describes the cores that a message visits. A two-dimensional array is used where the rows define the $MSG\ m$ and the columns represent the $CRS\ c$.

$$Path = \begin{bmatrix} p_{1,1} & \cdots & p_{1,c} \\ \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,c} \end{bmatrix} \in \{1, \dots, n\}^{m \times c} \quad (5)$$

3.7.6. Task dependency constraints

A dependency between two tasks t_k and t_l is given when the input message of the task t_l depends on the output message of the task t_k. For example, in the AM of the task, T4 depends on T0, T1, T2, and T3.

The following relationship between the injection times of input messages $MSG_{IN}(t)$ and output messages $MSG_{OUT}(t)$ of any task $t \in TSK$ holds:

$$\forall t \in TSK.\ \forall m_i \in MSG_{IN}(t).\ \forall m_o \in MSG_{OUT}(t).$$
$$t_{inject}(m_i) + \overline{md}_{total}(m_i) + \bar{et}(t) \leq t_{inject}(m_o) \quad (6)$$

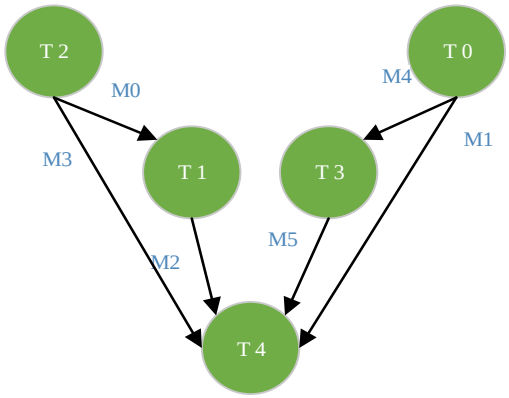


Figure 12. Conceptual AM

3.7.7. Message deadlines

To support real-time computing, we need real-time constraints for tasks and messages. For example, we denote with $D(m)$ the relative deadline of a message m. The injection of a message m must be sufficiently early that the communication can occur before its deadline $D(m)$. To ensure this, we must use the following time constraint for each message $m \in MSG$:

$$t_{inject}(m) + \overline{md}(m) \cdot (MaxHop + 1) \leq D(m) \quad (7)$$

We should ensure that the message m is only read at the destination after its deadline:

$$D(m) \leq t_{read\_dest}(m) \quad (8)$$

3.8. Decision variables

3.8.1. Slowdown factors (SDFs)

An essential decision variable in our ***DVFS*** optimizations model is the SDF of tasks and messages. $tsdf(t)$ denotes the SDF of a core for computing task t and $msdf(m)$ denotes the SDF for transmitting message m. Assuming that the contextual parameter C is zero,

then $tsdf(t)$ is proportional to the effective execution time $\overline{et}(t)$: $tsdf(t) \propto \overline{et}(t)$, and $msdf(m)$ is proportional to the effective message duration $\overline{md}(m)$: $msdf(t) \propto \overline{md}(m)$.

TSDF is the vector of the SDFs of all tasks in TSK:

$$TSDF = \begin{bmatrix} tsdf_1 \\ \vdots \\ \vdots \\ tsdf_k \end{bmatrix} \in \{tsdf_{min}, \dots, tsdf_{max}\}^k$$
$$k = |TSK|, \qquad tsdf_{min} \geq 1 \quad (9)$$

Analogously, $MSDF$ is the vector with the $SDFs$ of all messages in MSG:

$$MSDF = \begin{bmatrix} msdf_1 \\ \vdots \\ \vdots \\ msdf_n \end{bmatrix} \in \{msdf_{min}, \dots, msdf_{max}\}^n$$
$$n = |MSG|, \qquad msdf_{min} \geq 1 \qquad (10)$$

Based on these $SDFs$, the effective execution time $\overline{et}(t)$ of a task t and the effective message duration $\overline{md}(m)$ of a message m can be written as follows:

$$\overline{et}(t) = et(t) \cdot tsdf(t) + C \quad (11)$$
$$\overline{md}(m) = md(m) \cdot msdf(m) + C \quad (12)$$

The total effective message duration $\overline{md}_{total}(m)$ of a message m from sender to receiver becomes as follows:

$$\overline{md}_{total}(m) = \overline{md}(m) \cdot (hops(m) + 1) \qquad (13)$$

In addition, for the ***DVFS*** technique, energy consumption is related to frequency. When the frequency of cores and routers is reduced, the task execution times and message DM are increased accordingly. $MSDF$ and $TSDF$ are defined as the normalized frequency parameters in each schedule model (sm) [13]. Each SM has its $MSDF$ and $TSDF$ array for tasks and messages, which are assigned to the cores and routers. For example, $TSDF4 = 5$ means that the execution time of $T4$ is increased five times, while the $MSDF5 = 3$ means that the DM of $M5$ is increased by three times in the whole path from sender to receiver.

3.8.2. Hop count

$hops(m)$ denotes the number of intermediate visits of a message m along its transmission path from the sender to the receiver task:

$$HOPS = \begin{bmatrix} hops_1 \\ \vdots \\ \vdots \\ hops_n \end{bmatrix} \in \{1, \dots, MaxHop\}^n$$

$$n = |MSG| \quad (14)$$

$MaxHop$ is the maximum permissible number of intermediate hops. The source task, which is allocated to a core, initiates a message that will be transmitted through routers to the core of the destination task. $HOPS$ in [13] and [90] is used as a decision variable to specify the number of visited routers $hops_n$ of a message m to the destination task. In the absence of cyclic paths, the maximum $HOPS$ count is $MaxHop = RS + 1$.

3.9. Energy consumption

The power consumption P is equal dynamic power P_{dyn} plus static power P_{stc} [121]:

$$P = P_{dyn} + P_{stc} \quad (15)$$

There are analytical and empirical techniques for modeling P_{dyn} [122]. We also use an analytic method to model P_{dyn}. P_{dyn} tends to dominate contribution to power consumption, compared to P_{stc} [122]. Thus, in this work, we consider only dynamic power and dynamic energy consumption. P_{dyn} can be modeled as follows:

$$P_{dyn} = A \cdot S \cdot U^2 \cdot f \quad (16)$$

A is the activity factor that refers to fraction between 0 and 1 and can express the total capacity of circuits during each cycle charged or discharged, S is the switched capacitance that refers to aggregate load of system that depended on wire lengths on chip, U is the voltage, and f is the frequency [122].

Extracted from the above equation, the effective switch capacitance $S_{effective}$ is given as follows:

$$S_{effective} = A \times S \quad (17)$$

To keep our analytical model within the limits of the solver CPLEX, we consider $S_{effective}$ to be constant one. With this simplification the power consumption P_{dyn} becomes proportional to the clock frequency f and the square of voltage U:

$$P_{dyn} \propto f \cdot U^2 \quad (18)$$

Let $E_{T,dyn}(t)$ denote the dynamic energy consumption of a task t and $E_{M,dyn}(m)$ denote the dynamic energy consumption of transferring a message m. With P_{dyn} being the average power consumption, we can model the dynamic energy consumption as follows:

$$E_{T,dyn}(t) = P_{dyn} \cdot et(t) \quad (19)$$

$$E_{M,dyn}(m) = P_{dyn} \cdot md(m) \quad (20)$$

Where $et(t)$ and $md(m)$ are inversely proportional to the clock frequency:

$$et(t) \propto \frac{1}{f} \quad \wedge \quad md(m) \propto \frac{1}{f} \quad (21)$$

3.9.1. Effective Speed

$et(t)$ of a task t and $md(m)$ of a message m are given at the maximum clock frequency f_{max}. However, the effective execution time, denoted as $\overline{et}(t)$, and the effective message duration, denoted as $\overline{md}(m)$, depend on the selected clock frequency f_{sel} and a contextual parameter *C*:

$$\overline{et}(t) = et(t) \cdot f_{max}/f_{sel(t)} + C \qquad (22)$$

$$\overline{md}(m) = md(m) \cdot f_{max}/f_{sel(m)} + C \qquad (23)$$

This parameter C is a contextual setting delay, which is either zero in case that the clock frequency for the current task or message is the same as before, or otherwise a constant C_k, in case that the clock frequency has been changed at the beginning of the current task or message. The parameter C models the fact that changing the clock frequency in a processor takes some time. In our calculations, we assume that $C_k = 1ms$.

In DVFS the voltage U varies approximately proportionally with the clock frequency f, i.e., $U \propto f$, the power P_{dyn} is thus proportional to the cube of the clock frequency:

$$P_{dyn} \propto f^3 \quad (24)$$

Based on the above equation we can derive that the energy $E_{T,dyn}(t)$ of a task t and the energy $E_{m,dyn}(m)$ of a message m are proportional to the cube of the clock frequency multiplied by the execution/communication time:

$$E_{T,dyn}(t) \propto f^3 \cdot et(t) \ (25)$$

$$E_{m,dyn}(m) \propto f^3 \cdot md(m) \ (26)$$

3.9.2. Frequency Co-Efficient

Based on equations 24 and 25, the standard energy metrics cannot completely support our metric needs. In the following section, we denote the above proportionality factor of the energy consumption as the frequency co-efficient FE_t for a task and FE_m for a message respectively. However, the effective frequency co-efficient of tasks is denoted as $\overline{FE_T}$ and the effective message frequency co-efficient is denoted as $\overline{FE_m}$. These frequency co-efficients serve to quantify the energy reductions in the experimental evaluation and the abstraction from technology-specific parameters, such as the switched capacitance.

$$FE_T = et(t) \cdot f^3 \qquad (27)$$

$$\overline{FE_T} = et(t) \cdot (f_{max}/f_{sel(t)})^3 \ (28)$$

$$FE_m = md(m) \cdot f^3 (29)$$

$$\overline{FE_m} = (m) \cdot (f_{max}/f_{sel(m)})^3 \ (30)$$

3.10. Quadratization of the product

We use ***MIQP*** to solve the constraint of our optimization model. Hence, any term with three or more variables cannot be used in our optimizations model. However, in the constraints described so far, there are also non-quadratic expressions. For example, $msdf(m)^2 \cdot dm(m) \cdot hops(m)$ with $msdf(m)$ and $hops(m)$ are optimization variables.

To make our expressions quadratic [123], we use Fourer's strategy for "non-linear optimization" [124]. While this strategy is typically used for linearization, we use it here for "quadratization" [119]. The key principle is to split the optimization processes into two phases, wherein each phase a different variable is treated as constant. A ***real-time*** system must execute concurrent

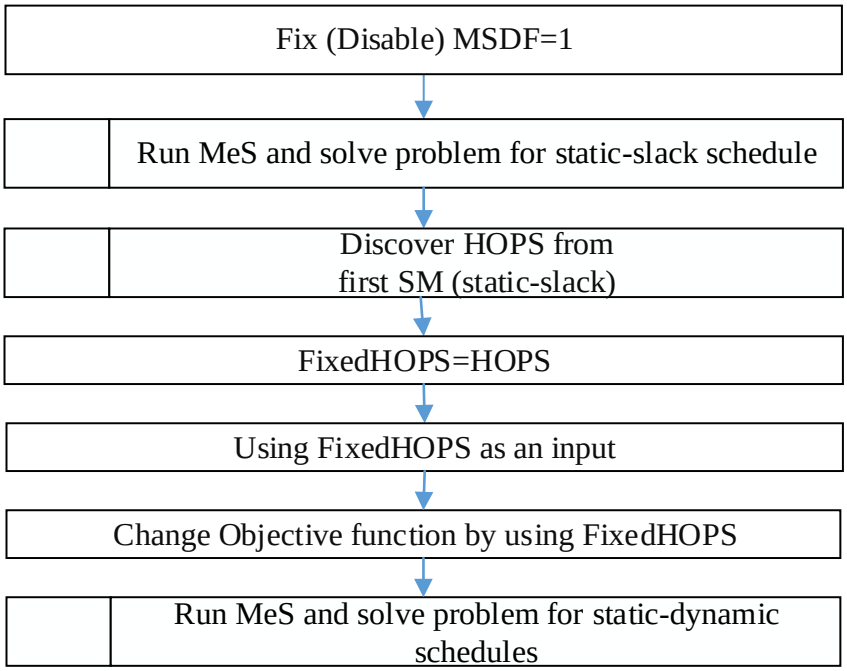


Figure 13. Quadratization technique via hops

Figure 13 illustrates how we use the quadratization of the constraint system of our model. In the first step, we optimize the model, with the $SDFs$ set to constant one (i.e., maximum frequency). From the obtained result, we extract the number of hops ($hops(m)$) and use these values as constants in a second optimization run, where we use the flow-down factors as a decision variable. In this way, we use CPLEX with an ***MIQP*** algorithm.

3.11. The objective function

The objective of our static-scheduling function is to maximize energy efficiency for both cores and routers by lowering clock frequency (c.f., [90]). The objective function is to minimize the makespan.

In this work, we use a predefined makespan as a constraint variable for controlling schedule timing. Power consumption is minimized by increasing timing (e.g., task execution times and message duration times). Timing is controlled via the slow-down factors of $tsdf$ and $msdf$. According to our energy model, $FE_T(t) = f^3 \cdot et(t)$ and

$FE_M(m) = f^3 \cdot md(m)$ and the SDF technique, which is $f_{sel} = \frac{f_{max}}{SDF}$ and $et(t) \propto \frac{1}{f}$ and $md(m) \propto \frac{1}{f}$, it follows that

$$P_{task} \propto tsdf(t)^3 \cdot et(t) \qquad (31)$$

$$P_{message} \propto msdf(m)^3 \cdot md(m) \qquad (32)$$

Figure 14 illustrates how, in one core, the $TSDF$ (which can be simulated for $MSDF$ for messages and routers) has an effect on multi-task timing and core frequency. For example, execution time $T0$, after reduction frequency from f_{max} to $f_{sel(\,t_0)}$, changes to $\bar{et}(\,t_0)$ and $et(\,t_0) < \bar{et}(\,t_0)$.

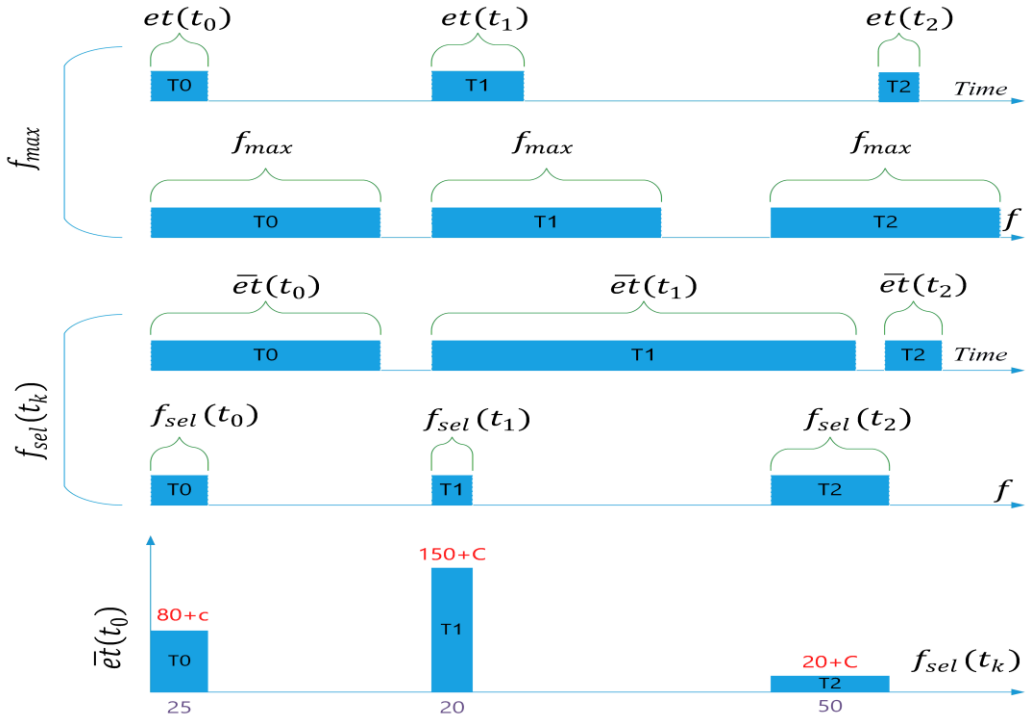


Figure 14.The effect of TSDF on tasks et(t) and core fault

The critical aspect of our SDF modeling is related to multi-frequency changes to tasks or messages. This means the frequency of each core for each task t is dependent upon and tuned by the $tsdf(t)$. Therefore, each task runs a specific and unique frequency $f_{sel(\,t)}$. In addition, in the same core, $f_{sel(\,t_k)}$ can be less than, equal to, or greater than $f_{sel(t)}$ of other tasks.

Table 4. Sample data calculated for Figure 14

$et(\,t_0)$	$et(\,t_1)$	$et(\,2)$
20s	30s	10s
f_{max}	f_{max}	f_{max}
100 Mhz	100 Mhz	100Mhz
$tsdf(t_0)$	$tsdf(t_1)$	$tsdf(2)$
4	5	2
$\bar{et}(\,t_0)$	$\bar{et}(\,t_1)$	$\bar{et}(\,t_2)$
$80s + C$	$150s + C$	$20s + C$

$f_{sel(t_0)}$	$f_{sel(t_1)}$	$f_{sel(t_2)}$
25 Mhz	20 Mhz	50 Mhz

Table 4 lists sample input data and calculations for SDF effects on task execution times and core frequency, as used in Figure 14.

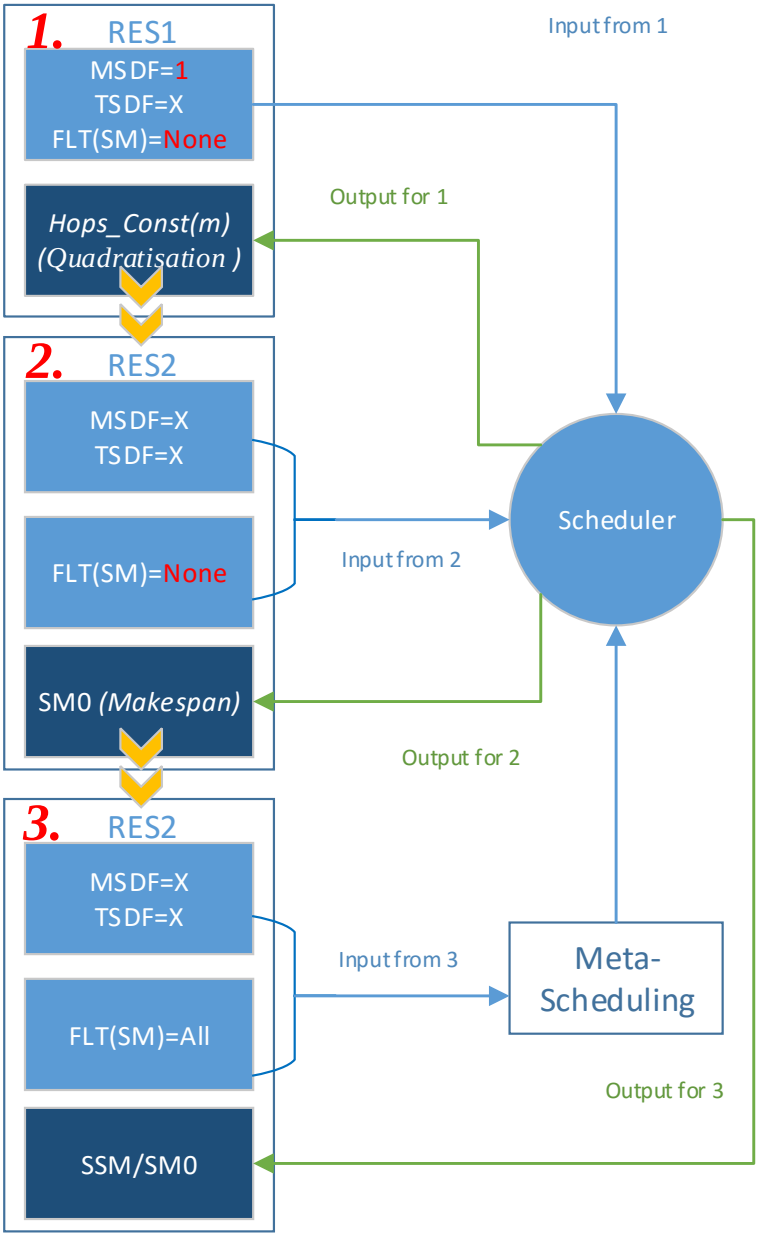


Figure 15. Three-step scheduling for finding optimum solutions

As described in section 2.6.3.2 on assuring quadratic expressions, the optimization process is partitioned into two phases.

For the first phase, we assume all $MSDFs$ are constant (Figure 13 and Figure 15):

$$\forall m \in MSG\ .\ msdf_{const}(m) = 1 \qquad (33)$$

Based on that, we get a quadratic constraint system with its goal maximized by CPLEX:

$$\forall t \in \text{TSK}.\ \forall m \in MSG\ .$$

$$CP_1(m,t) = et(t) \cdot tsdf(t)^2 + md(m) \cdot msdf_{const}(m)^2 \cdot (hops(m) + 1) \quad (34)$$

$$RES_1 = maximize\left(\sum_{i=1}^{|TSK|}\sum_{j=1}^{|MSG|}\left(CP_1(m_j,t_i)\right)\right) \qquad (35)$$

Goal: finding $hops(m)$ using the quadratization technique (section 3.10).

Phase 2: based on the optimization result, RES_1, the second optimisation phase is completed. First, the hop counts are extracted from the intermediate result RES_1:

$$\forall m \in MSG\ .\ hops_{const}(m)\ is\ extracted\ from\ RES_1 \qquad (36)$$

Next, the optimization result – including the $SDFs$ – is calculated using the constant values of the hop counts from the first solution:

$$\forall t \in \text{TSK}.\ \forall m \in MSG\ .$$

$$CP_2(m,t) = et(t) \cdot tsdf(t)^2 + md(m) \cdot msdf(m)^2 \cdot (hops_{const}(m) + 1) \quad (37)$$

$$RES_2 = maximize\ \left(\sum_{i=1}^{|TSK|}\sum_{j=1}^{|MSG|}\left(CP_2(m_j,t_i)\right)\right) \qquad (38)$$

***Goal*:** in this step, SM_0 with SS generate and maximum makespan is discovered.

Phase 3: finally, $makespan(SSM) = makespan(SM_0)$ are used as the timing constraint for $flt(sm)$ and the objective function is equal to RES_2.

Goal: scheduling all scenarios and creating SSM

3.12. Slack recovery technique

We use the slack time of tasks to increase energy-efficiency in ***NoCs*** by optimizing the system's timing. When tasks are running in a system, as shown in Figure 16, two types of slacks will occur: SS and DS.

SS is the time gap for dependent tasks in different cores, between the end of the first and the start of the second (e.g., SS SS0 between T1 and T2, SS1 between T3 and T0, and SS3 between T6 and T5). Other SS examples include two independent tasks on the same core (e.g., SS2 occurs between T6 and T4 on core 4.) DS occurs when a task finishes earlier than its ***WCET*** (e.g., DS0 in T1, DS1 in T3, DS2 in T6, and DS3 in T4).

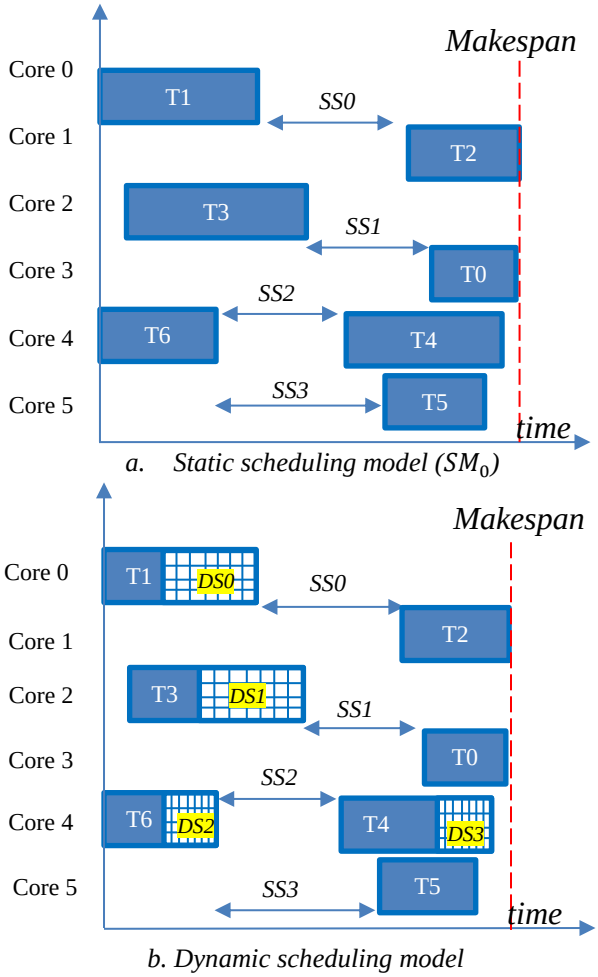


Figure 16. a. Static slack (SS) and b. Dynamic slack (DS) sample

3.12.1. Makespan and slack

The makespan $mks(sm)$ of a scheduling model sm is the time from the start of the first task in sm to the completion of the last task in sm. A useful scheduling optimisation is, for example, to reduce the makespan to fit it the time slot of a time-triggered system. ***MeS*** uses the occurrence of DS to reduce the makespan by shifting tasks.

With this method, the occurrence of DS shifts the system into a schedule with shorter makespan by going down along an edge of the SM tree. SM_0 sits at the top of the SM tree, thus all other schedules which have DS will have a lower makespan:

$$\forall sm \in \mathrm{SSM}/\{SM_0\}.\ mks(SM_0) > mks(sm) \qquad (39)$$

Given a scheduling model sm, by optimising the makespan, we obtain an optimized scheduling model sm_{opt}. The achieved saving of makespan time ($SavT$) is given as follows:

$$SavT = mks(sm) - mks(sm_{opt}) \quad (40)$$

Given the scheduling model shown in Figure 16.b, the scheduling model sm_{opt} resulting from the makespan optimization based on DS deployment is shown in Figure 17.

With this method, we aim to minimize energy consumption by speeding up computation, enabling the system to go sooner into idle mode.

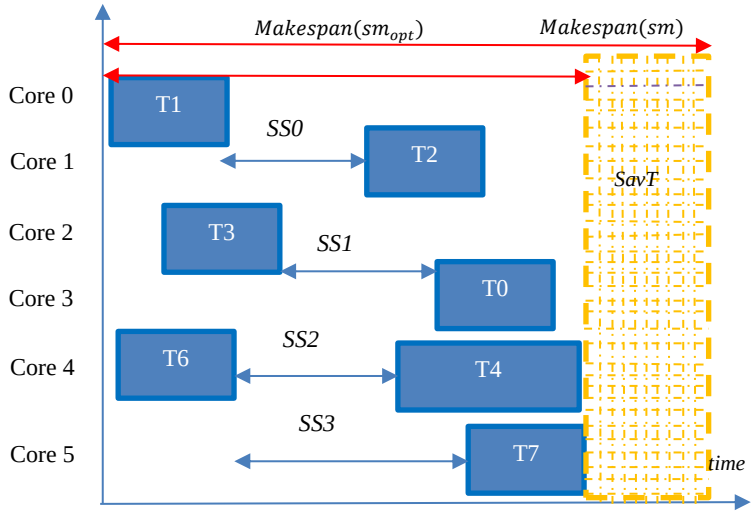


Figure 17. Usage of dynamic slack (DS) to reduce makespan

3.12.2. Power consumption and slack

With this method, we aim to minimize energy consumption to deploying DS to slow down the components, while preserving makespan. This means that we preserve the finish time of the schedule, but use the extra time for the remaining jobs with lower frequency, thus saving energy since a lower frequency means lower power consumption. In this method, when DS occurs, the next tasks do not shift, but run with an increased execution time due to lower frequency. Thereupon, energy-efficiency is increased.

Given the scheduling models shown in Figure 18, the scheduling model Figure 18.a resulting from the power consumption optimization based on DS of T4 deployment is shown in Figure 18.b. In this example, tasks T5 and T6 depend on T4. When DS0 occurs for T4, this free slot and SSs are used to achieve the maximum SDF by expanding and increasing the execution time of T5 and T6 via the frequency of the allocated core(s). The maximum time for T5 is equal to the following:

$$Max(\mathrm{et}_{t_5}) = \mathrm{et}(t_5) + DS_0 + SS_2 \quad (41)$$

Also, the optimal SDF for T5 is proportional to $Max(et_{t_5})$:

$$\mathrm{et}(t_5) \cdot TSDF(t_5) \leq Max(\mathrm{et}_{t_5}) \quad (42)$$

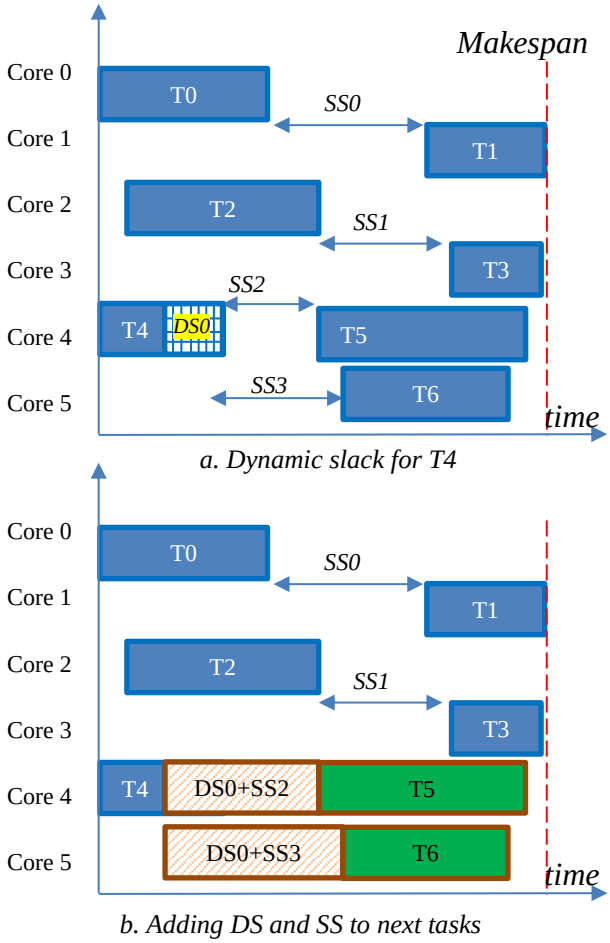


Figure 18. Usage of SDF regarding dynamic slack (DS)

Chapter 4: Scenario-based meta-scheduling (SBMeS)

The basis of the ***SBMeS*** technique expressed in this work is static scheduling [90]. This technique seeks to resolve scheduling problems in scenario-based scheduling and quasi-static task mapping [125], which is variously known as “super-scheduling” [11] and “meta-scheduling” [13].

Few articles focus on super-scheduling or ***SBMeS*** and adaptive ***TT*** (e.g., Persya et al.). [11] focuses on the dynamic super-scheduler, but we are using the static scheduler and pre-compiled schedules.

In this work, ***MeS*** is developed to solve ***SBMeS*** problems, with multi-task scheduling on a multi-core. We also propose an energy-efficient scheduling algorithm identify faults, DS, and $SDFs$ in multi-scenario-based systems for ***NoC***-based ***MPSoC*** (***EES–MPNoC*** [77]). Our scheduling algorithm achieves minimal frequency scheduling and can reduce the dynamic power consumption of ***NoCs***.

SBMeS and quasi-static scheduling strategies are used in platform-based designs (e.g., SAFEPOWER and ArtistDesign [126]). Where a group of schedules is generated off-line and at runtime, the scheduler chooses a specific schedule based on the designed scenario.

4.1. Meta-scheduler (MeS)

MeS [12, 13] designed to generate schedules based on events and scenario changes. When an event such as a fault (e.g., failed link or router) or slack occurs, the system reacts by routing to precomputed schedules [17]. In static scheduling, a feasible schedule is calculated offline [17]. Figure 19 shows how the meta-scheduler is used in a real system for ***MPSOC***.

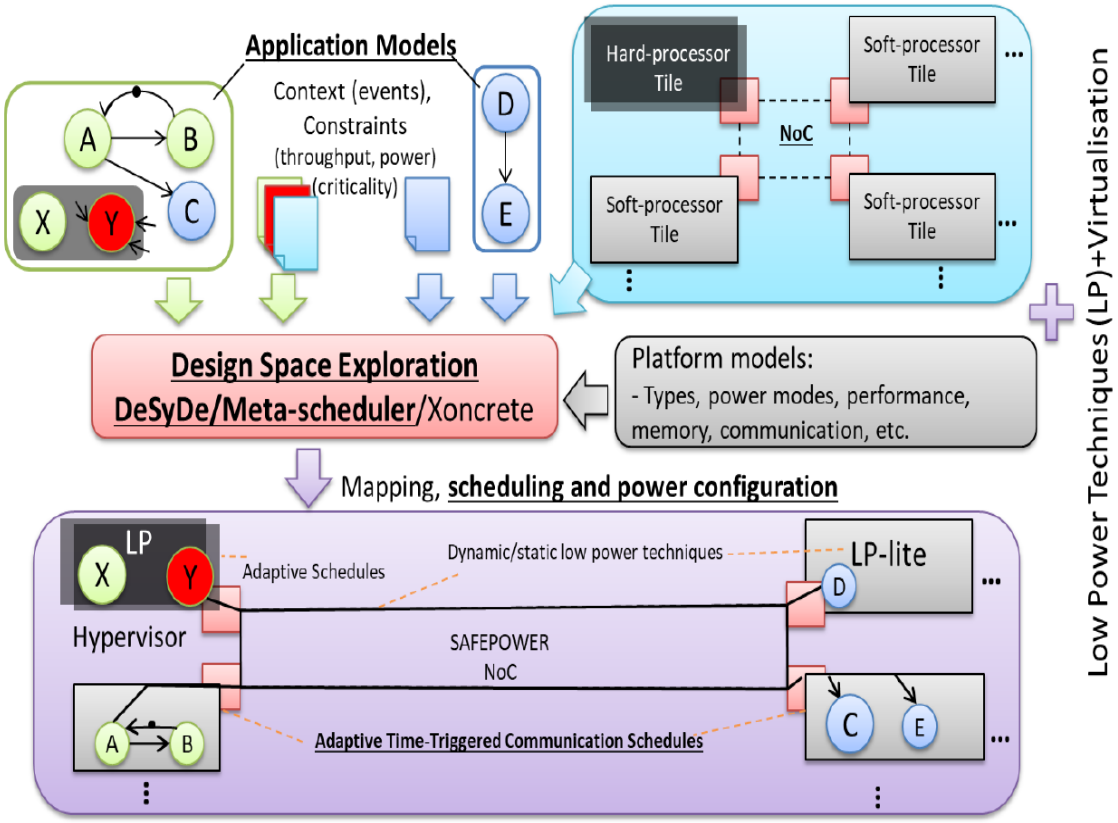


Figure 19. Usage of scenario-based meta-scheduling (SBMeS) on system-level design for SAFEPOWER multi-processor system-on-a-chip (MPSoC) [3]

4.2. System model and algorithm

The technique developed in ***MeS*** is ***depth-first algorithm establishing schedule backwards with tabu-set for re-convergence (FAESB-TSR)***, as presented in Figure 20. As part of the ***MeS*** solution, we developed a scenario-based graph traversal algorithm based on a heuristic tool and a ***backtracking algorithm*** based on this heuristic for identifying and managing events. The main goal of this method is to fetch, visit, execute, and traverse the events. The basic concepts of the ***FAESB-TSR*** are described in [6] (e.g., freezing and thawing, incrementing the algorithmic steps).

The bases all our scheduling model designs begin with AM, PM, and SM. As mentioned in section 3.1, ***MeS*** works with multiple scheduling models, organized in tree form to switch between schedules based on dynamic events. Thus, we introduce SSM as the set of all scheduling models. However, among the scheduling models $sm \in$ SSM, the one using SS is denoted as SM_0 All other SSM/$\{SM_0\}$ use DS.

In Figure 20, schedule model $SM = \{< vr, fz >\}$ denoting vr is variable and fz is a frozen element (e.g., task, core, message) with a value of 0 or 1. SM is equal to $AM \cup PM$ and all constants and constraints are used as input for the scheduling problem.

Context event CE denotes scenarios with relevant events on the timeline that lead to a change in SM and $CE = < ce, evt, @c, nv >$, with ce denoting the event name (e.g., slack, fault), evt denoting instant event time, $@c$ denoting a reference to a constant from

SM, and nv denoting a new value. Finally, the context model $CM = < ce, @c, vl >$ denotes all events.

Execution-Event $EE =< en, ei, @dv >$ corresponds to an application activity, such as the start of a task execution or communication activity. en denotes an event name, ei an event instant, $@dv$ a reference to a decision variable from SM and fz from *SM* is frozen.

Calendar $Cl = CE \cup EE$ contains a union of all events with the respective instants and creates an event and scenario guide map.

T denotes current scheduling time. When $T = ei$, the relevant instructions in Cl will be executed.

Schedule (AM, PM, SM): a new schedule is computed (e.g., using ***MIQP*** on CPLEX) when the values for non-frozen variables $fz(vr) = false$ are identified; hence, frozen variables have predefined contents ($fz\{vr_0 \ldots vr_n\}$).

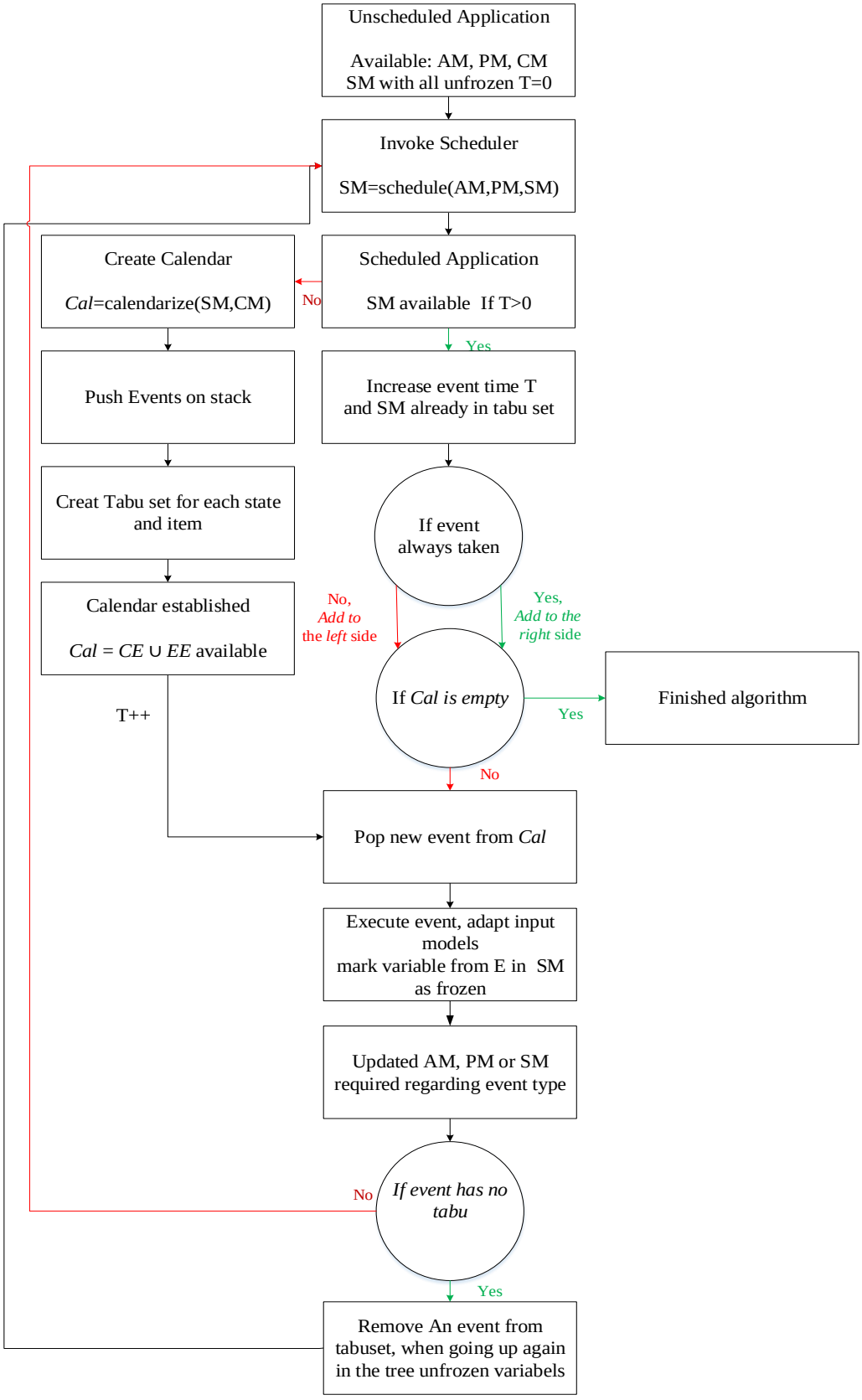


Figure 20. Depth-first algorithm establishing schedule backwards with tabu-set for re-convergence (FAESB-TSR)

4.3. Architecture of meta-scheduler (MeS)

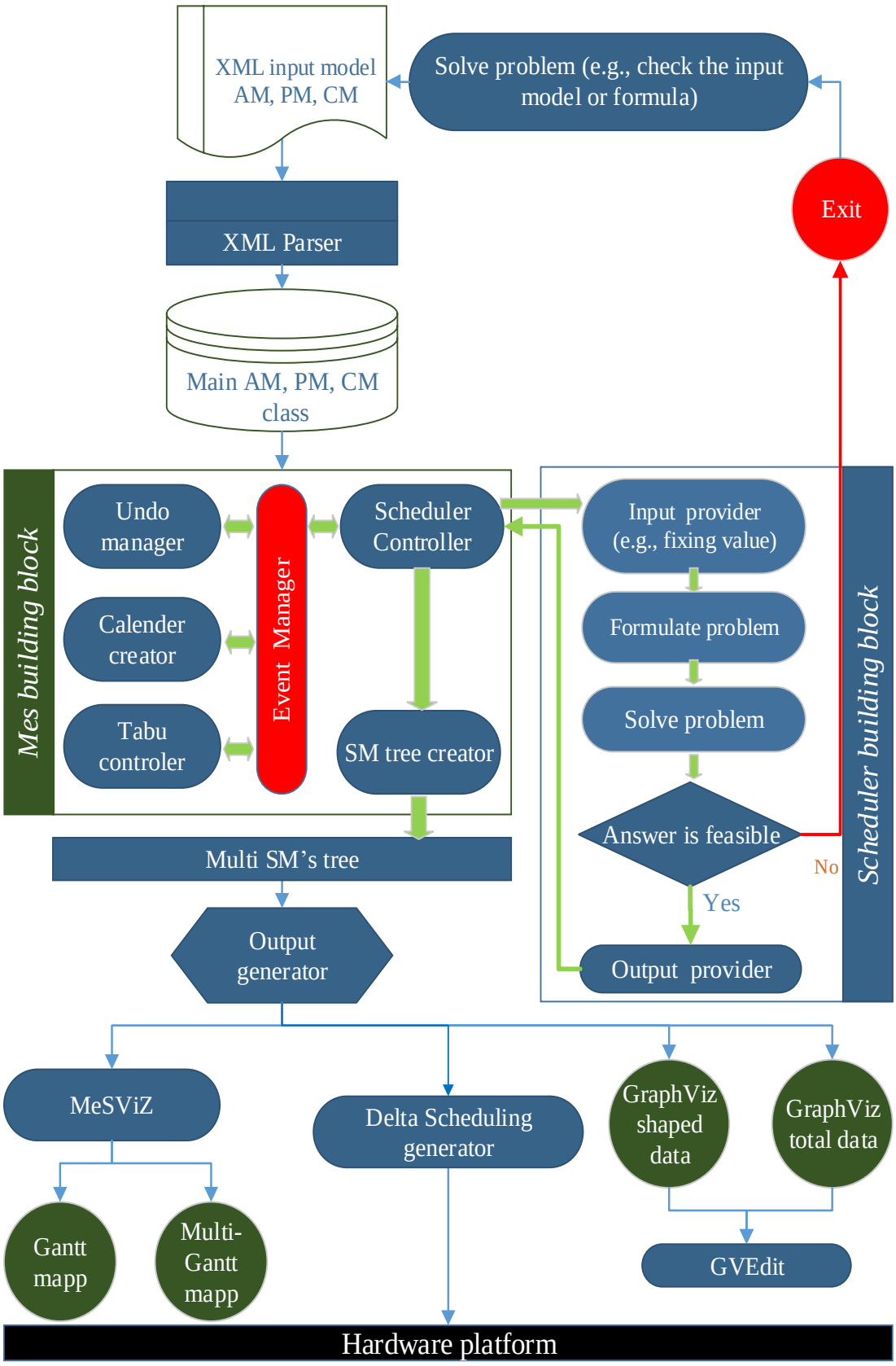


*Figure 21. Conceptual of the meta-scheduler (**MeS**) tool*

To solve the scheduling problem (i.e. energy optimization based on the slack time we use the ***SBMeS*** algorithm, as shown in Figure 20. Figure 21 represents a conceptual model of

the total process of the ***SBMeS*** technique. This includes input and output data (e.g., XML, Gantt map, Graphviz data) and their processes (e.g., XML parser, ***MeSViz***), data structures (e.g., AM, PM, and CM classes, SM tree), managers and controllers (e.g., event, undo, calendar, Tabu, scheduler), and scheduling processes (e.g., SM tree creator, scheduler, Delta Scheduling).

4.4. Meta-scheduler (MeS) design

4.4.1. Event-driving technique

Event driving and event handling are the most valuable aspects of ***SBMeS***, dependent on each type of event.

MeS begins by generating a static schedule with scheduling SM_0. To take the event into account, for each task of the AM, the *event manager* reduces the ***WCET*** by the slack time. DS is marked as an event and injected by the *calendar creator* into the calendar table.

This mechanism starting *freezing* algorithm after generating the *Calendar*, to cover each event regarding the et.

The *event manager* sends the event type (e.g., slack) and timing to the *schedule controller*. Depending on the timing of the calendar event, the s*chedule controller* will *freeze* or *unfreeze* the corresponding task in the *tabu controller*. The *tabu controller* freezes the partial schedule up to the event and leaves the rest to be completed by the scheduler.

4.4.2. The schedule model (SM) tree creator

The AM and PM are then sent to the scheduler to create new schedules by completing a frozen schedule. The *SM tree creator* generates the scheduling tree (multi-SM tree) and, depending on whether events are taken, they are either stored on the right or left side of the tree.

We build a ***SBMeS*** tree, with the top node assuming no occurrence of an event (e.g., DS). For example, each occurrence of a DS event switches to another schedule down the tree with better energy efficiency.

4.4.3. Output generator

Finally, when every event has been taken and the calendar is empty, ***MeS*** calls on the *output generator* to create two output files in *Graphviz* format: *shaped data* and *total data*. The *total data* file includes all the control parameters stored in the nodes of the scheduling graph. The *shaped data* file is a cleaned-up version of the SM tree, including only the real scheduling nodes but without the controller parameters. ***MeSViz*** is used to visualize single schedules as a Gantt map and events, depending on the schedules as a multi-Gantt map in different graphical format.

4.5. Functions and algorithms

4.5.1. Main functions and algorithms

The integration of the ***MeS*** functions is a crucial aspect of the design, speeding up significant amounts of data processing for scheduling and visualization. The XML parser processes all data in XML files to establish the AM, PM, and CM. ***MeS*** organizes the input data for the scheduler. In the final step, all schedules are visualized (*cf. Algorithm 1*).

```
function XML_Parser(xml file)
end function
function Meta − Scheduler(AM,PM,CM,SM)
    function Core − Scheduler(AM,PM,SM)
    end function
end function
function Core − Visalizer(AM,PM,SM)
    function Output(AM,PM,SM)
    end function
end function
function Meta − Visalizer(AM,PM,CM,SM)
    function Output(AM,PM,CM,SM)
    end function
end function
```

Algorithm 1. Main function algorithm

4.6. Scenario-based meta-scheduling (SBMeS) and fault modeling

In this part, we present the ***SBMeS*** model for the fault and reliability strategy and for both cores and routers on the schedules. Figure 22 shows a high-level overview of *SBMeS* and its methods and functions and the interconnection between them.

SBMeS is an offline schedule synthesizer and generator, and schedules results can be used repeatedly and cyclically during the system runtime operation. ***SBMeS*** makes heavy and complex computational demands for generating schedules offline. In other words, each operation and functionality of computational and communication parts can be implemented with a simple code and lookup table.

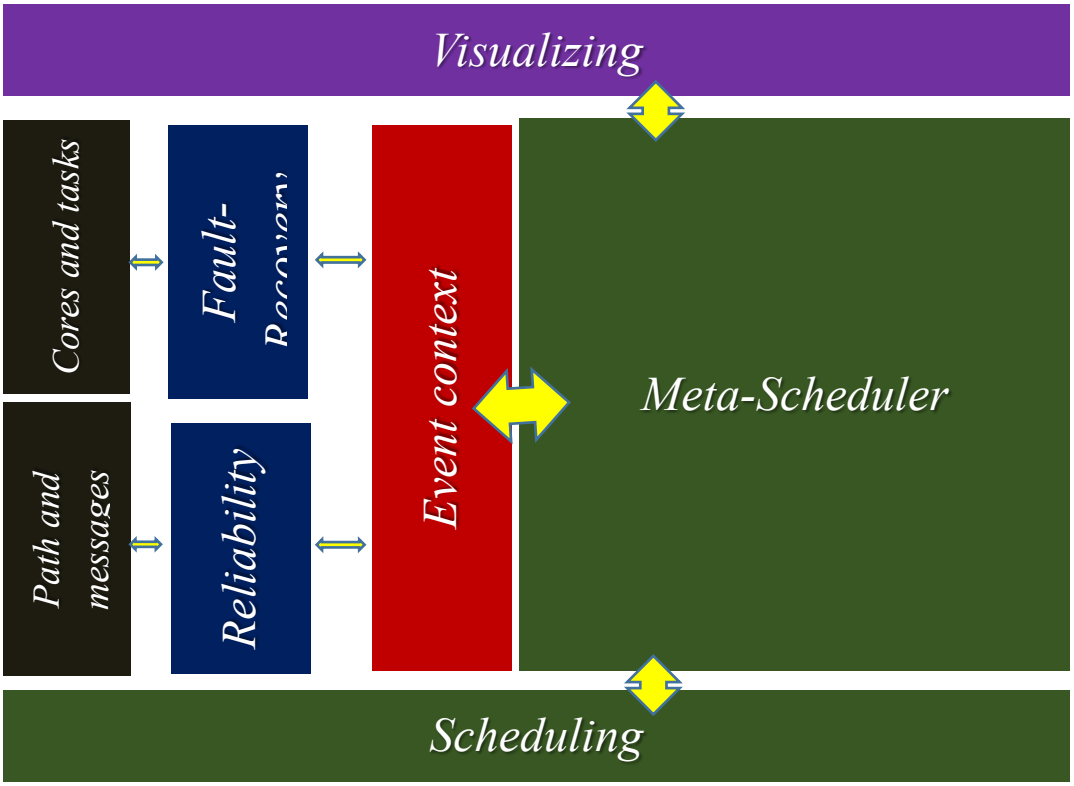


Figure 22. Scenario-based meta-scheduling (SBMeS) general model

4.6.1. Fault assumptions

In the ***SBMeS*** fault model, multiple faults can occur, and ***SBMeS*** supports a different range of faults (in hardware and/or software layers) according to the faulty definitions scenario in CM.

Therefore, each event in CM is characterized by its criticality level. We consider only core crash fault mode [127] in the PM layer to be a primary fault, and task slack as a sub-fault in the AM layer.

In our model, *CRS fault* has high criticality and priority level and slack is low. In AM, we can switch off each TSK slack and define limits for SDF for each task or message or use control deadlines for each TSK or MSG.

If any high criticality event is executed (core crashed), the system immediately switches to the related scenario with the high criticality mode [32]. As the system continues in this mode, all low criticality events (in our model slack) are limited to this status. However, further low criticality tasks continue to be executed and the system remains in high criticality mode until finishing the schedule or another low/high criticality event occurs.

We assume that, in the SAFEPOWER model, errors are detected by concept monitor and managed with the agreement layer, with each related schedule used on the platform.

4.6.1.1. Tolerance threshold range

We define TTR as "tolerance threshold range." PM is intended to tolerate the maximum number of failures $TTR = (CRS - 1)$. In our high criticality model, each CRS of the PM is mapped to a vertex in $PM = (V, E)$. Hence, if the sm scenario contains $sm(flt(c_x))$ that c_x do not work (crashed) and has a fault, then route connectivity and tasks executions should work and be computed without c_x. c_x must be omitted during the computation of the schedule and next events.

4.6.2. Fault-tolerant algorithm

When one or a series of core faults occurs, $flt(c)$, the scheduler must guarantee that the system can continue to work safely. It needs that each task and messages which are made dependent upon the failed core(s) $flt(c)$ via new schedule calculation, timing and remapping from a faulty core(s) to other healthy core(s) (migrating to the new location). Algorithm 2 shows how this mechanism works. The mechanism changes between messages, tasks, cores and, events stored in the SSM tree, which the hardware is able to use it for a new safe status.

$\forall c \in$ CRS $, \forall\ flt(c) \in\ FLT(sm, c).\ flt(c) = 1, \forall$ $e_x \in Event$ (sm)

$\boldsymbol{Fault\ recovery\ function}\ (\boldsymbol{AM}, \boldsymbol{PM}, \boldsymbol{CM})$

$flt(c_x) = 1\ \rightarrow\ \ call\ scenario\ (e_x)$

$-Push\ stack(\ AM, PM)$

$Freeze$ and $push\ every\ scheduled\ m$

and $t\ to\ table\ Freez(sm_x)$

$Push\ c_x\ to\ e_x$

$Find\ every\ healthy\ c\ which\ \ flt(c) = 0$

$Use\ Freez(sm_x)\ as\ fixed\ data\ input$

$Remove\ c_x\ \ from$ CRS

$Create\ new\ PM\ and\ AM$

$-Run\ scheduler$

$-Add\ sm\ to\ SSM$

$Event\ (sm)\ \ is\ free\ \rightarrow\ Exit$

$-Pop\ stack(\ All)$

Algorithm 2. Fault recovery and scheduling technique

Figure 23 represents the states of three events which occur in SSM. Figure 24 shows the results of each event in each SM_x.

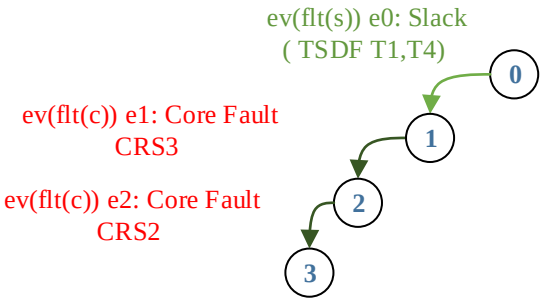


Figure 23. States of 3 events and their effect of each SM

For example, in SM_1 $ev(flt(s))$ slacks (for $T1$ and $T4$) and $TSDF$ are added to these tasks (e0). In SM_2, the $ev(flt(c))$ fault in CRS 3 $flt(c_3)$ causes $T2$ to shift to $CRS\ 0$ (e1).

In section 5.7, we will discuss how we can use this method and model to save memory by removing duplicated data. For example, in $SM_{0,1,2,3}$, T0 is repeated using the same location and size, $SM_{1,2,3}$ T1, and $SM_{2,3}$ T2 is repeated.

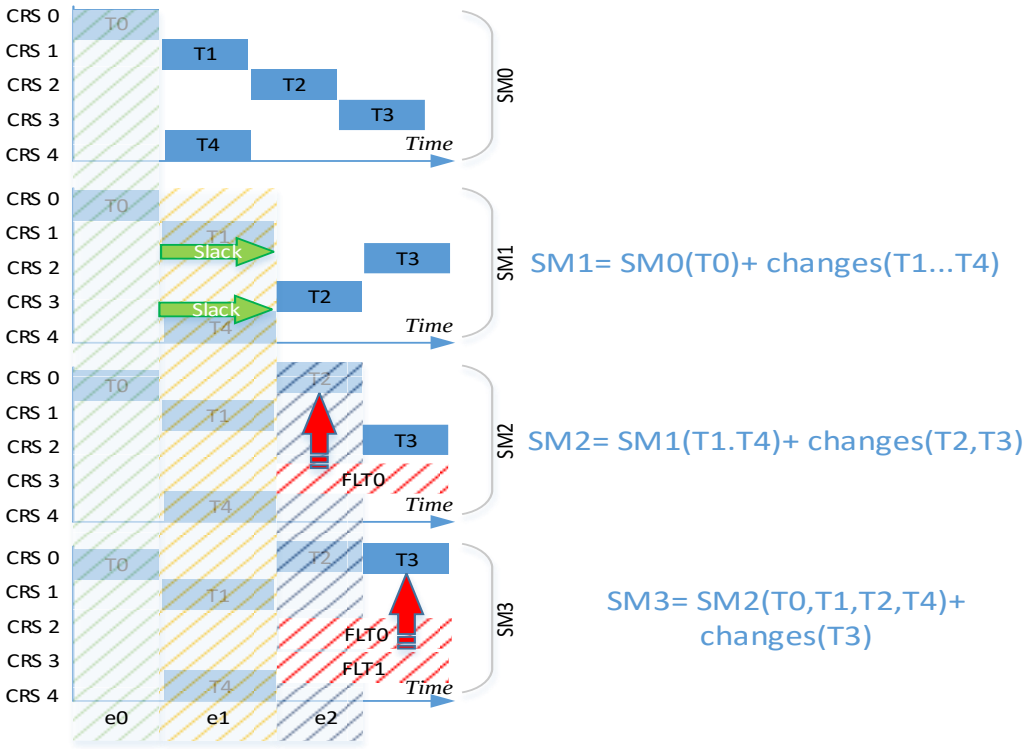


Figure 24. Events' effects in task scheduling

4.7. The goals of meta-scheduler (MeS) design

Flexible design: MeS is designed for scenario-based and static scheduling for adaptive ***TTS***. ***MeS*** enables scenario-based scheduling for fault, event, and context parameter changes.

Extensibility: MeS has a modular architecture that facilitates extensibility via XML-defined pluggable inputs. Sample components for visualization and scheduling are included in the main functions and may be easily replaced with custom functions.

A significant challenge in scheduling is finding schedules that are both feasible and optimum. The modular scheduler architecture enables the selection of several scheduling parameters to improve scheduling performance. ***MeS*** achieves optimized schedules by changing the default optimizer of CPLEX to barrier optimizers, and ***MeS*** establishes a ***MIQP*** problem to find a global optimum. The first-order optimality parameter of CPLEX (local optimum) is changed to the value optimal global. In some cases (e.g., speed up), it is possible to disable modules (e.g., power modelling, visualization).

Chapter 5: Visualization and evaluation of schedules

5.1. Requirements for basic visualization

A visualizer is needed to show the contents of MeS including AM, PM, CM, and SM. A basic schedule visualizer is capable of displaying the following items:

A) Show resources, tasks and messages location, timing.
B) Show the messages routes, routers and dependencies between the sender and receiver tasks.
C) Present textual information of messages properties (e.g., MSDF, route)
D) Textual information of tasks properties (e.g., ***TSDF***, ***WCET***)
E) Schedule ID (SM_x)
F) Generated text and image output formats (e.g., JPG, PDF)

5.2. Requirements for meta-visualization

A meta-visualizer of multi-schedule SSM first needs to access the basic data generated by the basic visualizer and then use the CM contents to combine, analyze, and generate extra details to display following items:

- parent and child schedule nodes in one scope
- detailed event information in textual and graphical format
- a report of the difference between parent and child nodes
- task and message location and $D(m)$ changes in textual and graphical forms
- the changed messages routes
- full information of total and removed schedules (e.g., invalid or pruned nodes)

5.3. Graph mapping

The Gantt chart is a straightforward solution for visualizing schedules. It is better quality and simpler than textual format for aiding understanding of schedules and their dependencies. "*This method is a common solution for human-resource scheduling, and in this work, it is referred to as "Gantt mapping" [10]* " (e.g., Figure 25).

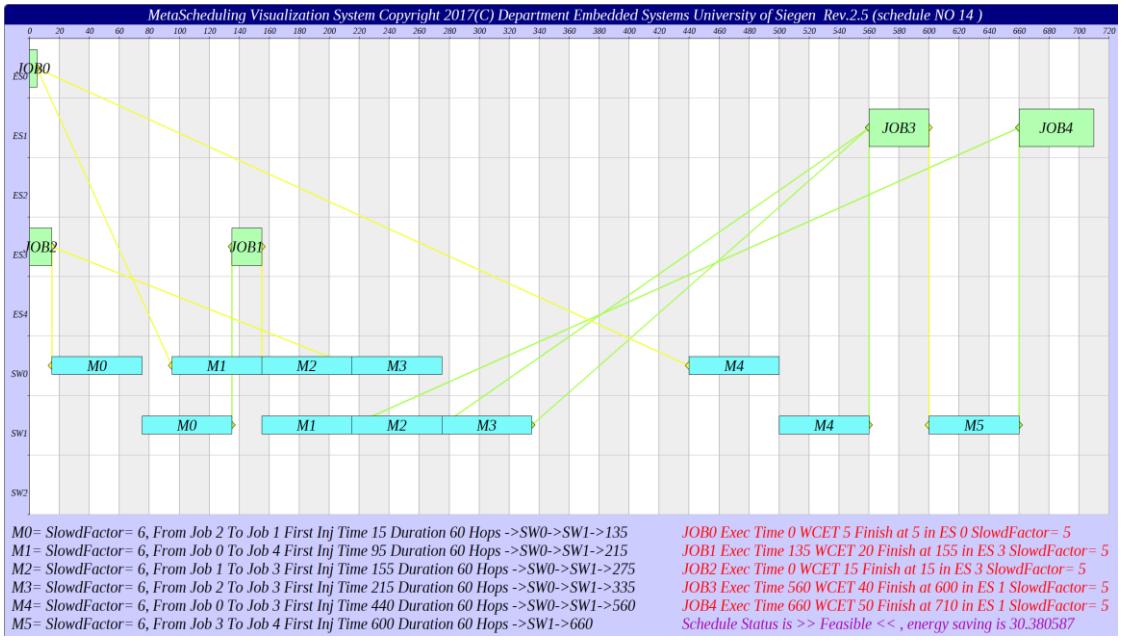


Figure 25. Schedule Gantt map

5.4. Gantt mapping

In ***MeS***, the description of events and schedules in hierarchical graphs is necessary for handling and debugging large and complex schedules. To present the schedules and event parameters, the graph model needs to be distinctive. These distinctives are the information in the nodes and edges of the graphs (e.g., ID SM_x, event). A graph mapping represents the event dependency between the schedules, as shown in Figure 26.

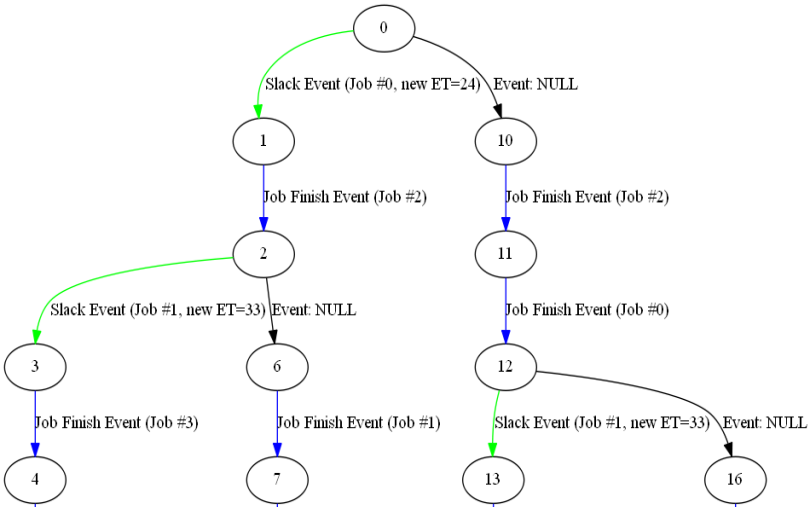


Figure 26. Schedule tree include all data

5.5. Visualization of schedule changes

After a fault (event) occurs, the scenario and system states must change. However, it is vital that the designer compare the schedules to identify the changes that occurred. ***MeS*** generates this information, but an overview of the *SM* is required: this is "meta-visualization**."** ***MeSViz*** must be able to identify when the event occurred, what kind of

changes were caused by it, and which elements were changed. Examples include tasks and messages $D(m)$ or allocation decisions, as shown in Figure 27.

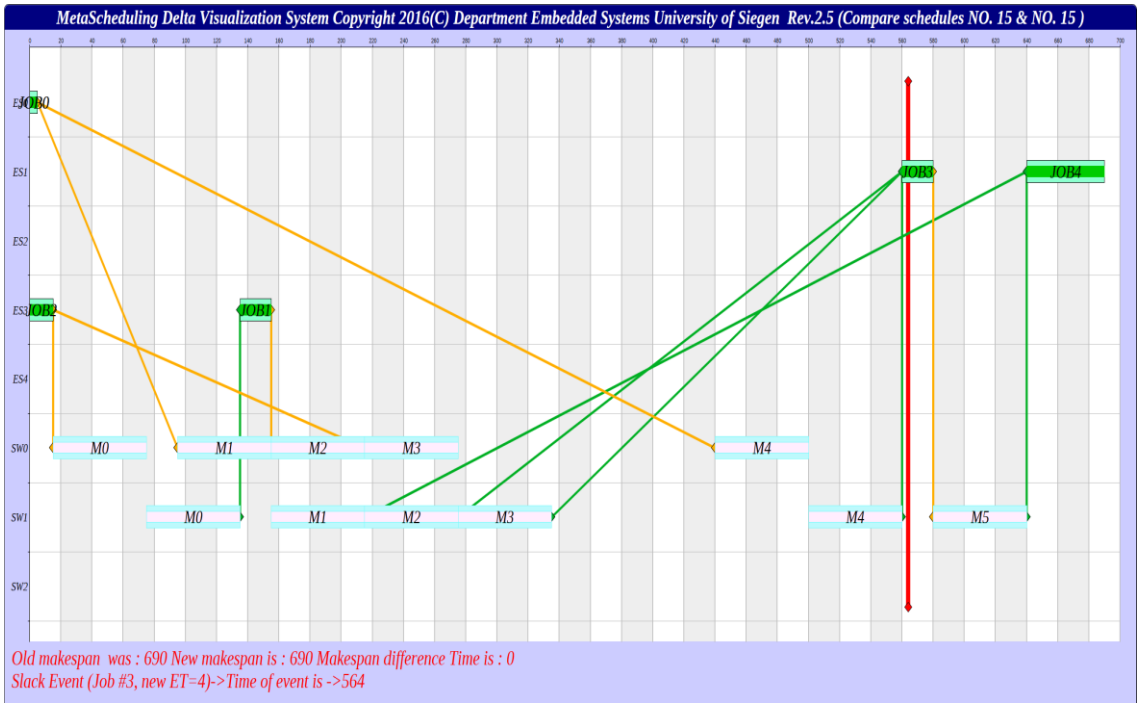


Figure 27. Meta-visualization of an event

5.6. Energy calculation

To evaluate the total energy reduction for both cores and routers for the ***NoC*** performance and deadline misses of our algorithm, ***MeSViz*** was customized to calculate and display the energy consumption of a sm and SSM.

5.6.1. Energy consumption

MeSViz is used to calculate the energy consumption $FE(sm)$ of each scheduling model sm or an average of all dynamic scheduling models, $FE_{avg,dyn}(\text{SSM}/\{SM_0\})$.

The energy $FE_C(c, sm)$ consumed at any core $c \in$ CRS for all its tasks can be calculated as follows:

$$\forall sm \in \text{SSM}.\ \forall c \in \text{CRS}.\ \ FE_C(c, sm) = \sum_{t \in \text{TSK(c,sm)}} et(t) \cdot \left(\frac{f_{max}}{tsdf(t)}\right)^3 \quad (43)$$

where $TSK(c, sm)$ is defined as the set of tasks mapped to a core c in scheduling model sm:

$$TSK(c, sm) = \{t \in TSK \mid alloc_t(sm) = c\}$$

The total energy consumption of all the cores $FE_C(sm)$ can then be calculated as follows:

$$FE_C(sm) = \sum_{c \in CRS} FE_C(c, sm) \qquad (44)$$

The energy $FE_R(r, sm)$ consumed at any router $r \in$ RTR for all its messages can be calculated as,

$$\forall sm \in \text{SSM}.\ \forall r \in RTR.$$
$$FE_R(r, sm) = \sum_{m \in MSG(r,sm)} md(m) \cdot \left(\frac{f_{max}}{msdf(m)}\right)^3 \quad (45)$$

where $MSG(r, sm)$ is defined as the set of all the messages going through router r in scheduling model sm. The total energy consumption of all the routers $FE_R(sm)$ can then be calculated as follows:

$$FE_R(sm) = \sum_{r \in RTR} FE_R(r, sm) \quad (46)$$

In every calculation, we assumed $f_{max} = 1$.

The whole energy consumption of a system for a concrete scheduling model sm, $FE(sm)$, is calculated from the energy consumption of all the routers and cores as follows:

$$FE(sm) = FE_C(sm) + FE_R(sm) \quad (47)$$

The average FE energy consumption $FE_{C,avg,dyn}$ of all cores for all scheduling models with DS can be calculated as follows:

$$FE_{C,avg,dyn} = \frac{1}{(|SSM|-1)} \cdot \sum_{sm \in (\text{SSM}/\{SM_0\})} \sum_{c=0}^{|CRS-1|} FE_C(c,sm) \quad (48)$$

The average energy consumption $FE_{R,avg,dyn}$ of all routers for all scheduling models with DS can be calculated as follows:

$$FE_{R,avg,dyn} = \frac{1}{(|SSM|-1)} \cdot \sum_{sm \in (\text{SSM}/\{SM_0\})} \sum_{r=0}^{|RTR-1|} FE_R(r,sm) \quad (49)$$

The average FE energy consumption for all scheduling models with DS can be calculated as follows:

$$FE_{avg,dyn} = FE_{R,avg,dyn} + FE_{C,avg,dyn} \quad (50)$$

5.6.2. Energy reduction

MeSViz compares the FE reduction of cores ($ReFC_C$) and routers ($ReFE_R$) combined with SM_0, which uses SS, and the average FE of all the other scheduling models with DS SM_x. The relative reduction of the average FE for the scheduling models with DS $sm \in (\text{SSM}/\{SM_0\})$, compared to the FE of the scheduling model with SS SM_0, are computed separately for SM cores $ReFE_C(SM)$, average cores ($ReFE_C$), SM routers $ReFE_R(SM)$, ($ReFE_R$), each SM ($ReFE_{sm}$) and combined ($ReFE$) as follows:

$$ReFE_C(SM) = \frac{FE_C(SM_0) - FE_C(SM)}{FE_C(SM_0)} \cdot 100\% \quad (51)$$

$$ReFE_R(SM) = \frac{FE_R(SM_0) - FE_R(SM)}{FE_R(SM_0)} \cdot 100\% \quad (52)$$

$$ReFE_C = \frac{FE_C(SM_0) - FE_{C,avg,dyn}}{FE_C(SM_0)} \cdot 100\% \quad (53)$$

$$ReFE_R = \frac{FE_R(SM_0) - FE_{R,avg,dyn}}{FE_R(SM_0)} \cdot 100\% \quad (54)$$

$$ReE_{sm} = \frac{FE(SM_0) - FE(SM)}{FE(SM_0)} \cdot 100\% \quad (55)$$

$$ReFE = \frac{FE(SM_0) - FE_{avg,dyn}}{FE(SM_0)} \cdot 100\% \quad (56)$$

5.7. Memory and meta-schedules

To avoid wasting memory and reaching the memory limits of ***NoCs*** and ***MPSoCs*** (e.g., ARM11, ARM7 4kB cache, and 256K private), we aim to extend the methods used in [25, 128] and combine them with our own approach. In [25, 128], the stored memory of SSM for message duplication is reduced, but we combined (the stored memory) with a task duplication method, which made SSM memory usage more efficient.

Figure 28 represents the effects of a core fault and slack event on SSM, and Figure 29 shows how many share points and how much duplicated data (in this case tasks) are stored in related SMs, thus wasting memory space. For example, SM_2 and SM_3 have similar amounts of data.

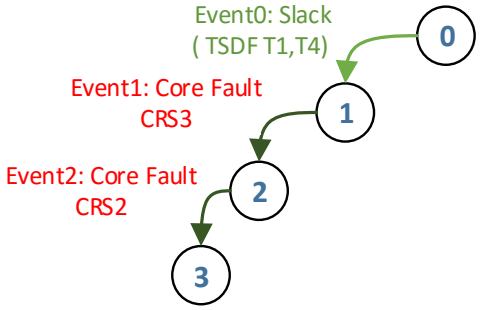


Figure 28. Events state in s schedule tree

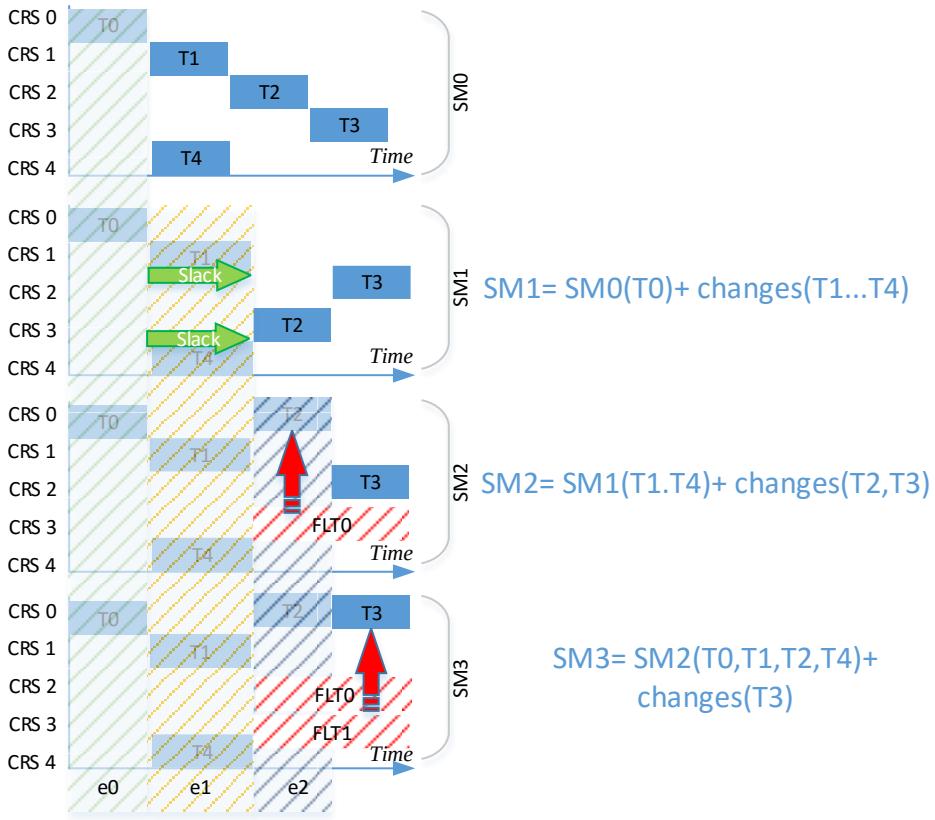


Figure 29. Schedules share points regarding task changes and Figure 28

5.7.1. Convergence of meta-schedules for saving memory

Due to the number of events, N_e, number of schedules in an SSM N_{ssm}, and memory size limitation in embedded systems, M_{lim}, increasing N_e can cause increases in consumption of memory by the stored schedules.

5.7.2. Memory consumption

If we assume the number of tasks with slack is $N_{tsk\_s} = 10$, and cores with faults is $N_{flt(c)}$, $N_{tsk} = 12$ (meaning two tasks do not have slack), and $N_{msg} = 14$, then $Ne = N_{tsk\_s} + N_{flt(c)}$, which in this case is $N_e = 14$.

We assume the memory storage needed to store each task by default is $M_{tsk} = 5\ Bite$ and for each message is $M_{msg} = 5\ Bite$.

If $N_{ssm} = 2^{Ne}$ then $N_{ssm} = 2^{14}$, which is equal to 16384 schedules. According to the platform structures (e.g., Xilinx, ARM Cortex, and Intel Arria) and operating system, the memory addressing and storage methods vary.

The memory storage space for each schedule is equal to $M_{sm} = M_{tsk} \times N_{tsk} + M_{msg} \times N_{msg}$, which in this case is equal to the following:

$$M_{sm} = 5 \times 12 + 5 \times 14; M_{sm} = 70B.$$

Total memory space for each SSM can be calculated as $M_{ssm} = M_{sm} \times N_{ssm}$, which in this case is $M_{ssm} = 70 * 16384;\ TMmsch =$ 1,146,880.

The results of M_{ssm} show that, with an increasing number of events, tasks, and messages, it is vital to find a controlling solution to reduce memory consumption for critical embedded systems.

To solve the memory consumption problem, we must look at the changes in schedules after events. Therefore, we designed and tested the *Example 1*[6] in 7.3.1.1.

If $N_{ch(msg)}$ is the number of messages changed and $N_{ch(tsk)}$is the number of tasks changed, then the total real space without data duplication $M_{real(ssm)}$that is needed for the whole scenario is equal to the following:

$$M_{real(ssm)} = \sum_{i=0}^{N_{ssm}} (N_{ch(msg)} \times M_{msg} + N_{ch(tsk)} \times M_{tsk}) \qquad (57)$$

5.7.3. Delta scheduling technique and delta tree

With *Algorithm 3*, we see the parent and child schedule changes and differences (e.g., parent schedule SM_0 and child SM_1), which we call the delta scheduling technique (DST). All ***DST*** (e.g., $DST\ (SM_0, SM_1)$) data discovered will be stored on a specific tree, known as the delta tree (DT).

The value of saving duplicated memory is calculated as follows:

$$SavM = M_{ssm} - M_{real(ssm)} \quad (58)$$

Function DTS (SM_x, SM_{x+1})

$If\ SM_x\ is\ Parent\ and\ SM_{x+1}\ is\ Child\ \rightarrow$

(a)
$For\ (int\ i = 0; i < N_{msg}; i++)$
$If\ (SM_x(\ M[i].inject\_time) \neq SM_{x+1}(M[i].inject\_time)$
{
$Add\ SM_{x+1}(M[i])\ to\ DT\ (SM_{x+1})$
$N_{ch(msg)}++;$
}
(b)
$For\ (int\ i = 0; i < N_{tsk}; i++)$
$If\ (SM_x(T[i]\ start\ time) \neq SM_{x+1}(T[i]\ start\ time)\ )$
OR
$(SM_x(T[i]\ End\ time) \neq (SM_{x+1}(T[i]\ End\ time)$
{
$Add\ (SM_{x+1}(T[i]\ End\ time)\ to\ DT\ (SM_{x+1})$
$N_{ch(tsk)}++;$
}

Algorithm 3. DTS discover and calculating changes for messages (a) and tasks (b)

[6] The part of data and results and examples are used on [128].

When the platform for the event needs to decode DT, it only requires access to the DT (SM_{x+1}) and its parent (c.f., Figure 30).

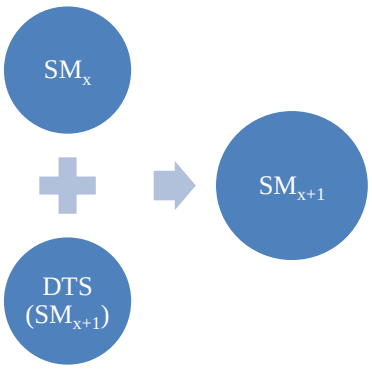


Figure 30. Decoding schedules from delta tree (DT)

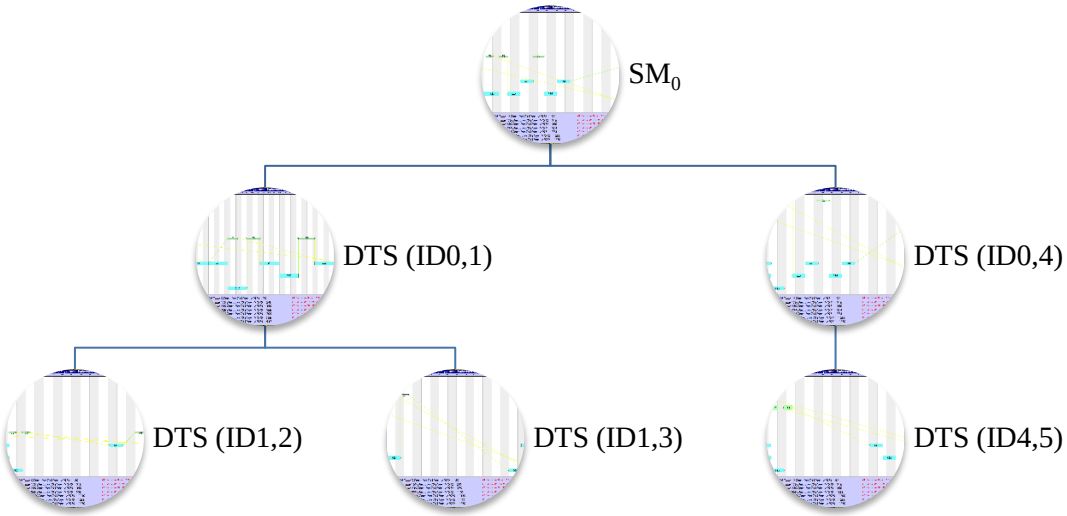


Figure 31. Delta tree (DT) data model

Therefore, rather than storing complete schedule data in the memory, we can encode graphs to DT by replacing duplicate data with changed data (cf. Figure 31). When it needs to decode $DTS(ID[n])$, adding changes to the running schedule (concerning the event and timing, as introduced in Figure 30 and [128]), it can run without delay.

In [128], we assumed memory needed to store each message is $Mms = 4$ Bite, and it only supports memory space for messages.

5.8. Summary

In this section, we introduced ***MeSViz*** algorithms and functions to help the system designer find bugs or conflicts in ***MeS*** results concerning AM, PM, CM, and ***MeS*** algorithms (e.g., objective function).

In addition, we introduced DST and DT and discussed how they can improve memory-saving capacity, evaluating them using three examples.

CHAPTER 6: IMPLEMENTATION

6.1. Schema modelling

MeS is a novel flexible architecture for scenario-based scheduling in adaptive time-triggered systems. To achieve this in embedded systems, ***MeS*** is modelled conceptually in several structures. Figure 32 illustrates how the system designer uses the schema method. The method has valuable benefits in terms of saving time and costs, making it simple and fast to every input files via a schema editor, using a general schema model for reference.

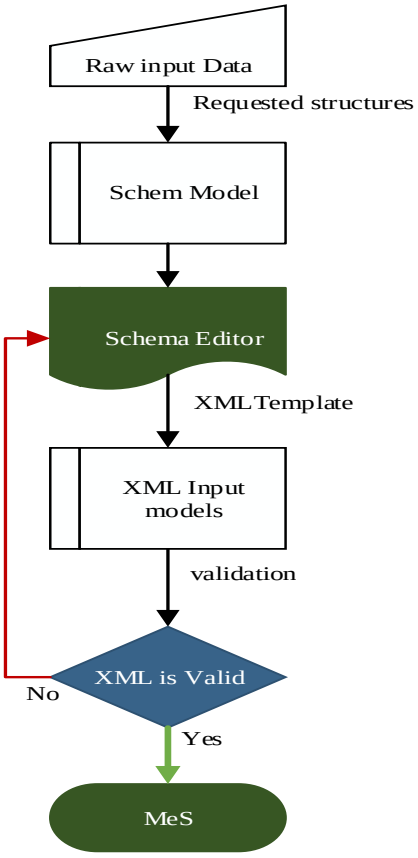


Figure 32. Schema technique for standard data structure modelling

Table 5 presents a raw data model using regular input data.

Table 5. Sample raw input data before forming in XML format

Application Data	
Task ID=0 ET=50energy=[1,100] Task ID=1 ET=60energy=[1,100] Task ID=2 ET=42energy=[1,100]	Msg. ID=0 from job#2 to job#1 Msg. ID=1 from job#0 to job#4 Msg. ID=2 from job#1 to job#3
Physical Data	
Router ID5 energy=[0,100] Router ID6 energy=[0,100] Endsystem ID0 energy=[0,100] Endsystem ID1 energy=[25,100]	Link ID=0 from node#0 to node#5 Link ID=1 from node#1 to node#5 Link ID=2 from node#2 to node#5

6.2. Implementation of the data model

To create correct data as an input for ***MeS***, data formats for the PM, AM, SM, and CM must be defined. These data formats can be expressed using an XML schema.

The Oxygen[7] XML editor was used here to describe the structure of the documents for AM, PM, and CM. The goal of an XML schema is to prepare the common standard building blocks of an XML document (cf. overview of SM in Figure 33).

$< SchedulingModel\ event\_ID = ""\ ID = ""\ makespan = ""\ >$

$< ApplicationModel > \{1,1\} </ApplicationModel >$

$< PlatformModel > \{1,1\} </PlatformModel >$

$< ContextModel\ event\_ID = "" > \{1,1\} </ContextModel >$

$</SchedulingModel >$

Figure 33. Overview of scheduling models

6.3. Implementation of meta-scheduling (MeS)

The ***MeS*** architecture, as shown in Figure 34, uses IBM ILOG CPLEX optimization C++ libraries to solve the scheduling problems and generate optimum solutions [129].

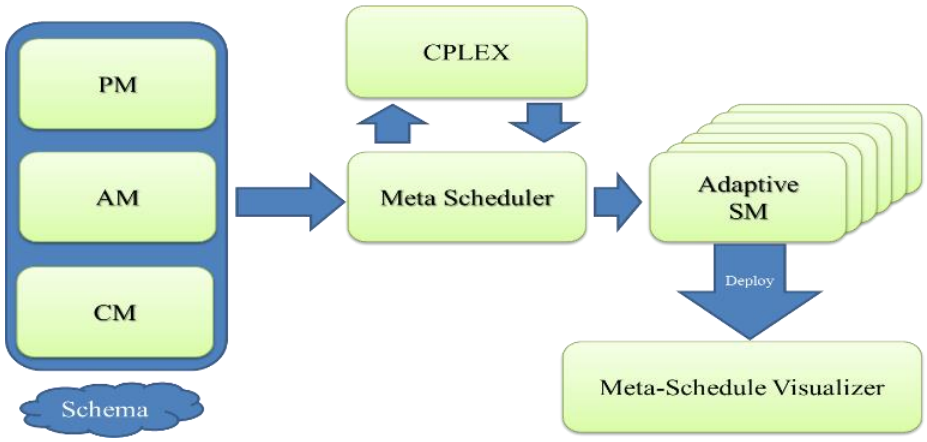


*Figure 34. The simple architecture of meta-scheduling (**MeS**)*[8]

To generate a static schedule with CPLEX, the system designer must describe the problem (c.f., section 3.5) using decision variables (c.f., section 3.8), constants, constraints, and an objective function (c.f., section 3.11). To solve the scheduling algorithms in this work, *IBM ILOG CPLEX* uses barrier optimizers. The ***MIQP*** method can obtain an optimal solution, while ***MeS*** establishes an ***MIQP*** problem to find a global optimum.

MeS uses the LibXML++ [9] parser to parse XML input data. ***MeS*** is designed in a modular and object-oriented style, with each component implemented as a separate C++ class.

[7] *Oxygen XML Editor http://www.oxygenxml.com/xml_editor.html*
[8] This concept and figure is used on Safepower D3.8 [3].
[9] *The GNOME project* - https://developer.gnome.org/libxml++-tutorial/stable/index.html

6.3.1. XML

The PM, AM, and CM elements (e.g., nodes, tasks, messages, faults) must comply with the data format of the schema.

As the first step in ***MeS***, a parser reads and processes all data in the XML files to establish the AM, PM, and CM data structures (cf. Algorithm *4*).

```
function XML_PARSER(xml file)
    Process xml file
    CREATE(AM,PM,CM)
end function
```

Algorithm 4. XML parser

All the raw input information (e.g., Table 5) is modeled in the schema format via the Oxygen XML editor to describe the structure of XML documents. This helps us to prepare the standard and flexible standard building blocks of an XML document in all ***MeS*** generations. Figure 35 is a sample of standardized input XML for use in ***MeS***, validated by the schema editor.

When the processing of the input models in XML format is complete, they are stored internally in ***MeS*** as a data structure. For defined scenarios in CM, every generated schedule and depended SM, which are stored in an SM class, is added as a node in the SM tree.

```
< ApplicationModel >
< Task ID = "0" WCET = "2" min_energy = "1" max_energy = "100" deadline = "1100"/>
< Task ID = "1" WCET = "4" min_energy = "1" max_energy = "100" deadline = "1600"/>
< Task ID = "2" WCET = "6" min_energy = "1" max_energy = "100" deadline = "1305"/>
< message ID = "0" from = "0" to = "1" size = "10" min_energy = "1" max_energy = "100" deadline = "1185"/>
< message ID = "1" from = "1" to = "2" size = "10" min_energy = "1" max_energy = "100" deadline = "1185"/>
</ApplicationModel >
< PlatformModel >
< node ID = "0" Type = "switch" min_energy = "0" max_energy = "100"/>
< node ID = "1" Type = "switch" min_energy = "0" max_energy = "100"/>
< node ID = "2" Type = "switch" min_energy = "0" max_energy = "100"/>
< node ID = "3" Type = "endsystem" min_energy = "0" max_energy = "100"/>
< node ID = "4" Type = "endsystem" min_energy = "25" max_energy = "100"/>
< link ID = "0" from = "0" to = "1"/>
< link ID = "1" from = "1" to = "2"/>
< link ID = "2" from = "3" to = "0"/>
< link ID = "3" from = "4" to = "1"/>
</PlatformModel >
< ContextModel >
< SlackEvent job = "0" NewExecutionTime = "50"/>
< SlackEvent job = "1" NewExecutionTime = "50"/>
< FaultEvent type = "crash" >
< NodeFault NodeId = "0"/>
</FaultEvent >
< FaultEvent type = "crash" >
< NodeFault NodeId = "1"/>
</FaultEvent >
</ContextModel >
```

Figure 35. Standardized input XML sample

6.4. Implementation of meta-scheduling visualization tool (MeSViz)

MeSViz uses ChartDirector [130] to design extensive chart types and the Graphviz [131] format for the GVEdit application and other graph generator tools.

When all the schedules are generated, the visualizer begins to process AM, PM, and SM for Basic Visualizer (BV). BV is the core of ***MeSViz***.

6.4.1. Outputs and formats

MeSViz generates three graphical files and two text files.

Schd_Vis_XXXXX.png: This file is the primary output of each schedule (XXXXX is the schedule ID).

MetaSchd_Vis_XX_YY.png: This file is advanced output (XX and YY are the IDs of relevant schedules), which visualizes, calculates, and presents the difference between two related schedules (before and after an event) in the tree of scheduled events (e.g., Figure 26).

Output.txt: This file includes all possibilities for nodes and events and shows all wrong or correct data. It is built by the scheduler (e.g., Figure 36).

$digraph\ G$ {

$0 -> \ 1\ [label = "Slack\ Event\ (Task\ \#6, new\ ET = 10)" color = \ green]$;

$1 -> \ 2\ [label = "Slack\ Event\ (Task\ \#8, new\ ET = 10)" color = \ green]$;

$2 \rightarrow \ 3\ [label =$ Slack Event (Task #0, new ET=25)$color = \ green]$;

}

Figure 36. Example a textual data with three nodes

Graph_Tree.png: GVEdit reads the graph tree output of ***MeS*** and generates a graph map. Figure 26 is the result of the above example.

6.4.2. Single schedule Gantt mapping

Each Gantt map contains numerous data (cf. Figure 25):

- The schedule ID (SM_x) on the top row
- The total makespan time
- The cores and routers ID on the left
- The message ID, sender, and recover task, injection time, Du, visited H and routers on the bottom row with black color
- The task ID, execution time of each task, ***WCET***, finishing time, and related core at the bottom
- Each task is aligned with the exact start and finish time and node coordination, in a green color and with its label
- Each message is aligned with the exact time and router coordination, in a blue color and with corresponding labels
- The yellow line is used by the sender, and the green line is used for the receiver task of the message

6.4.3. Multi-schedule Gantt mapping

For each event in the schedule model, a multi Gantt map is created by a combination and overlap of two schedules (parent and child) for better detection of the differences between them. This overlap model displays all the changes occurring after the event, which helps it to compare related schedules (e.g., Figure 27). These data include the following:

- The event time, indicated by a vertical red line
- The schedule elements for the parent are darker than those for the child
- Changes on makespan
- Event information (e.g., timing)
- Tracking of changes to tasks and messages
- The parent and child ID at the top

6.4.4. Graph mapping of meta-schedules

Graph mapping of schedules is the final step in visualization. For simple models – and in this work – Graphviz[10] is used. For big data, the yEd[11] graph editor and Gephi[12] can be used. Each graph shows the following:

- The schedule ID (SM_x) on each node
- Event details (e.g., event name, timing, task ID)
- Each event connected by a specific color

[10] www.graphviz.org
[11] www.yworks.com
[12] gephi.org

Chapter 7: Evaluation example scenarios and results

This section presents the results of the linearized ***MIQP***[13] model and evaluates the ***MeS*** results, visualized using ***MeSViz***. ***MeS*** and ***MeSViz*** were run on a virtual cluster machine with 12 cores of a $Intel\ Xeon\ E5-2450\ 2.2\ GHz$ and $60GB\ RAM$ on Ubuntu 14.04.5 ($GNU/Linux\ 3.13.0-100-generic\ x86\_64$).

7.1. Evaluation objectives

The goal of this section is to evaluate all the model, algorithms, and formula introduced in previous chapters. To achieve this, we designed scenarios for both different and similar domains (c.f., Table 6).

Table 6. Difference between the designed scenarios

Section	TSDF	MSDF	Multi-task	Dynamic-slack	Core fault	Save memory
7.3.1	Y	N	N	Y	N	Y
7.3.2	Y	Y	N	Y	N	Y
7.3.3	Y	Y	Y	Y	N	Y
7.3.4	Y	Y	Y	Y	N	Y
7.3.5	Y	Y	Y	Y	Y	N

7.2. Visualizing scenario-based meta-schedules for adaptive time-triggered systems (TTS)

The primary goal is to prepare a visualizer for visualizing meta-schedules as Gantt mappings. ***MeSViz*** helps engineers and developers to quickly obtain an overview of schedule behavior for different events.

7.2.1. Simple model

This model is designed with limited conditions and elements and the properties are as follows.

7.2.1.1. Schedule model (SM) content

The AM includes five tasks and $seven\ messages$, while the PM includes seven nodes ($five\ cores$ and $two\ routers$). The CM includes $five\ slack$ events with $NewExecutionTime = 50$, $five\ battery$ events with new energy levels, and $seven\ faults$ via node crashes.

[13] *linearization switch for QP, MIQP* (IBM). ibm.com/support/knowledgecenter/en/SSSA5P_12.7.0/ilog.odms.cplex.help/CPLEX/Parameters/topics/QToLin.html

7.2.1.2. Output results

The basic visualizer generates 94 Gantt maps from the basic scheduler (Figure 25), and ***MeSViz*** generates six Gantt maps (Figure 27) from ***MeS***. GVEdit generates graph mapping with 37 nodes.

Figure 37 is a sample of 94 visualized schedules generated by ***MeSViz***. It represents the information needed by the system designer to control and manage the schedule behavior. This includes the following:

- Messages and task allocations
- SDF for each task and message
- Message dependency for sender and receiver task, routing paths, injection time, and DM
- Task execution time and finishing time
- Total schedule time

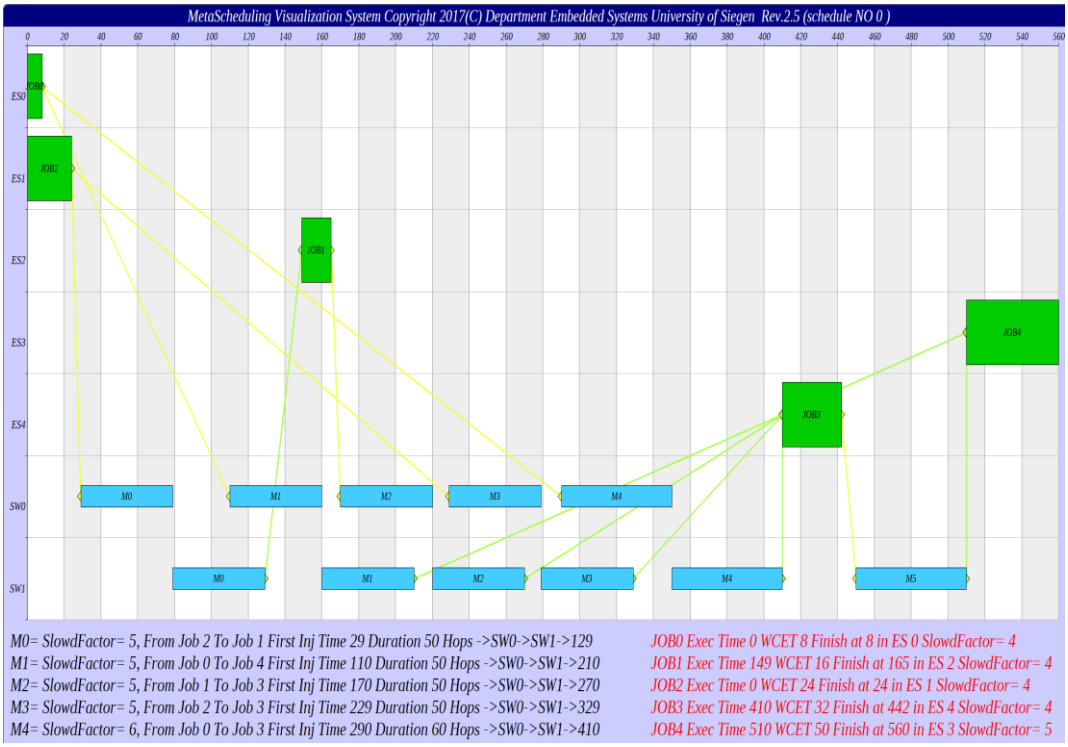


Figure 37. Static slack (SS) schedule model (SM) generated by meta-scheduling visualization tool (MeSViz)

7.2.2. Complex model (CM)

This model was designed with more elements to control performance.

7.2.2.1.1. Schedule model (SM) content

The AM included 15 tasks and 15 messages, while the PM included 20 nodes (16 cores and 4 routers). The CM included 14 slack events with NewExecutionTime, 13 faults with

node crashes, 14 faults with link faults and babbling idiots, and 14 faults with message omission events.

7.2.2.1.2. Output results

BV generated 3048 Gantt maps from the basic scheduler, and the meta-visualizer generated 35 meta-visual Gantt maps from ***MeS***. GVEdit generated a graph map with 37 nodes.

Figure 38 represents the schedule tree, with each node containing an SM identifier and each edge representing the schedule status (e.g., $Status = 0 ... invalid$ and $Status = 1 ... valid$), the energy reduction value, occurred events (e.g., slack), task identifier, and new execution time.

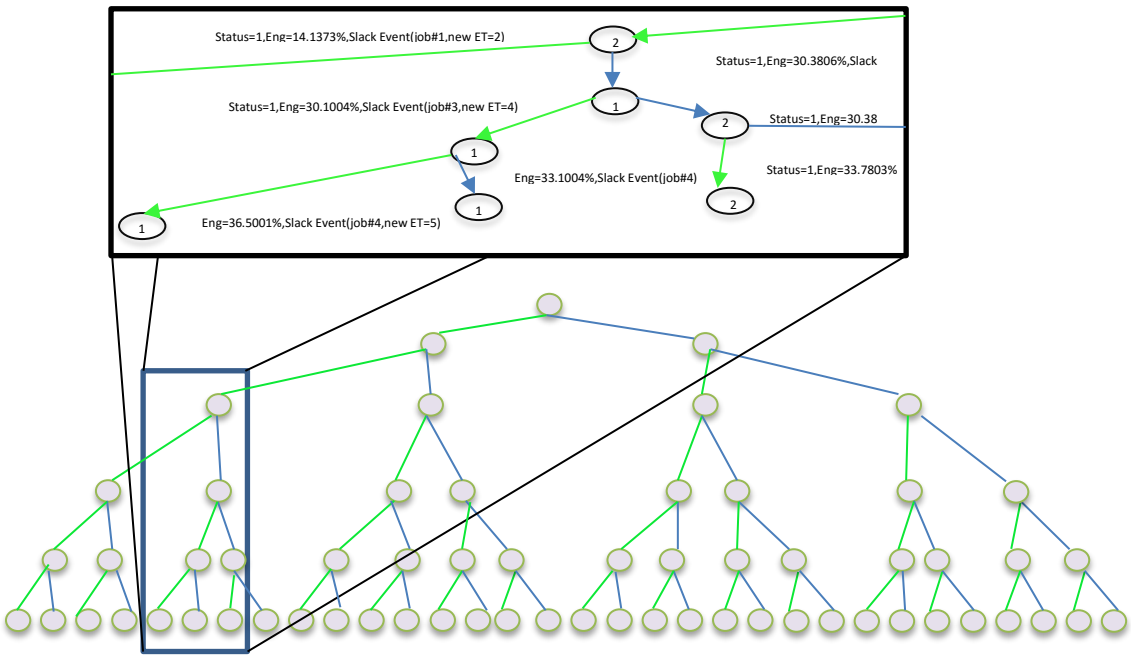


Figure 38. Schedule tree with 94 schedules (created via meta-scheduling visualization tool (MeSViz) and GVEdit)

Figure 39 shows the Gantt map for one specific schedule with SM_{44}, generated by ***MeS*** and visualized using ***MeSViz***. Figure 39 includes all the data useful for debugging and implementation.

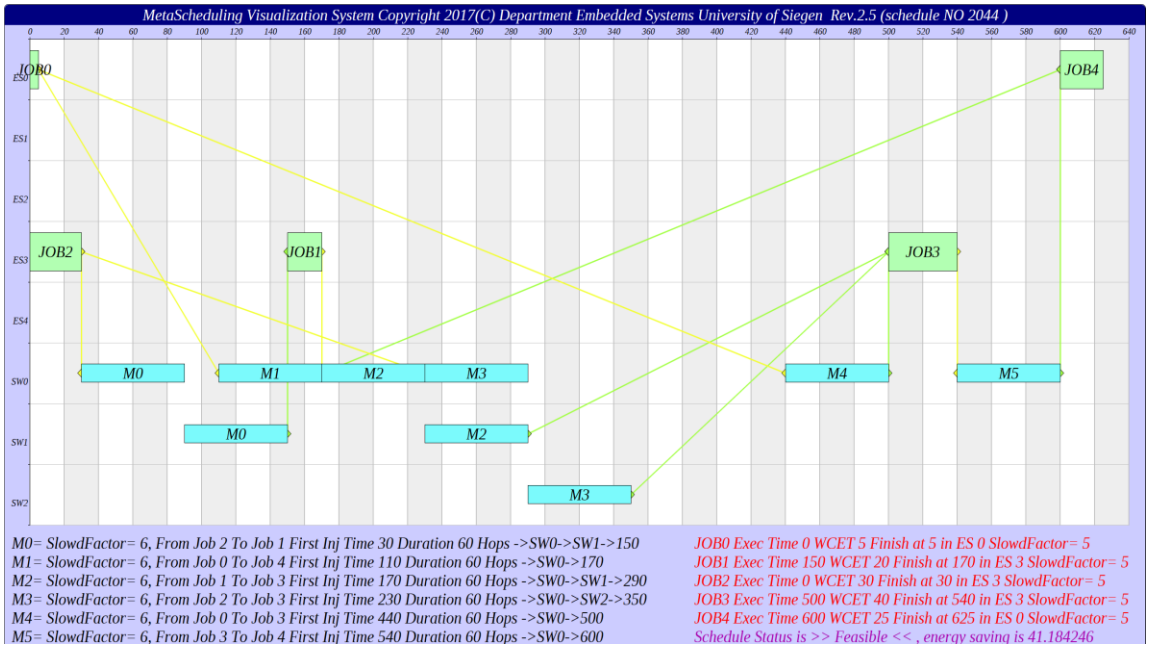


Figure 39. Gantt map of schedule ID 44

For example, it shows that, of five cores, only two are used. It indicates how the tasks and messages are allocated and depend on one another, the values of the $TSDF$ for each task and $MSDF$ message, the scheduled status, the energy reduction rate, the message path from one hop to another, and the timing of messages and tasks.

7.2.3. Discussion

MeSViz generates Gantt maps and graph maps. After analyzing outputs, we identified several design faults.

Incorrect configuration: MeSViz helps to detect incorrect input data (e.g., a message does not connect to a task or makes a wrong loop between cores and routers). If the ***MeS*** has faulty constraints or conditions, generated schedules will be incorrect. A design fault is illustrated in Figure 40, where $T14$ is the sender of $M14$ and $T11$ is the receiver of $M14$. The problem is the execution time of $T11$, which finishes earlier than $M14$.

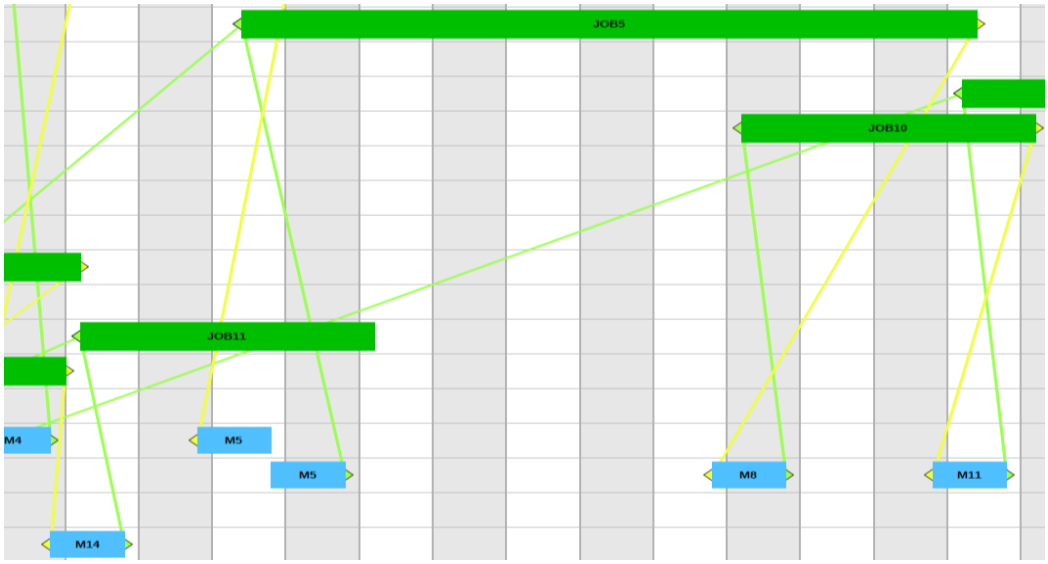


Figure 40. Incorrect results and information found in the complex schedule

MeS incorrect data**:* When ***MeS creates incorrect outputs, incorrect results are shown in ***MeSViz*** – as seen, for example, in Figure 41. $M12$, in the old schedule, has the correct connection to the sender and receiver ($T12$ and $T13$), but the new schedule shows a crossed line.

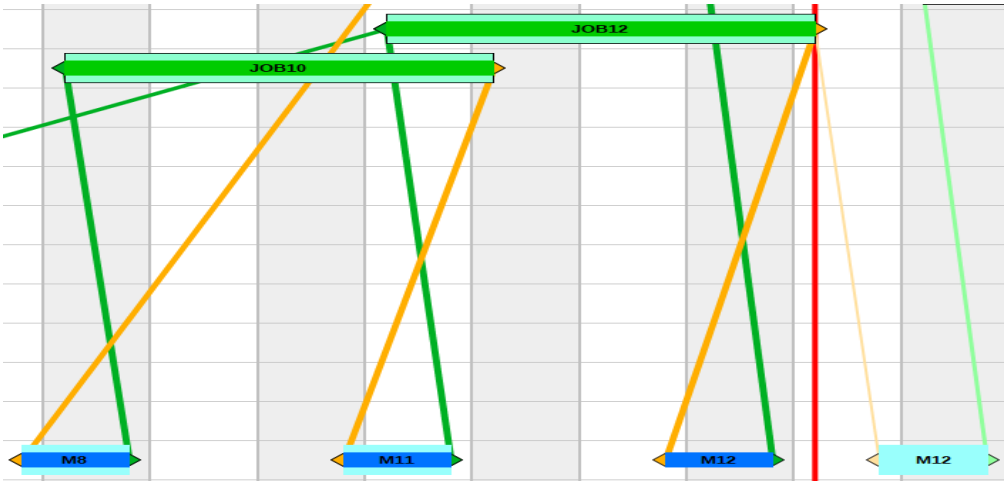


Figure 41. Meta-schedule Gantt map generated from meta-scheduling (MeS) class

For evaluation schedules, we must control the direction and slope of the connections between the sender and receiver of each message (cf. *Algorithm 5* and *Algorithm 6*).

```
if line Slope from sender to message is negative
    Or
    line Slope from message to receiver is negative then
    schedule is not valid
end if
```

Algorithm 5. Slope Evaluation 1

while line slope from all senders to recivers is **left to right**
Or
line is **Vertical** **do**
schedule **is** valid
end while

Algorithm 6. Slope Evaluation 2

In Figure 40 and Figure 41, the system designer can easily find any task or message not correctly aligned.

7.3. Evaluation of meta-scheduling (MeS) results

7.3.1. Convergence results for saving memory

7.3.1.1. Example 1. Sample scenario with four tasks and three messages

In this sample, $N_{tsk\_s} = 4, N_{cf} = 0, N_{tsk} = 4$ (meaning all tasks have slack) and $N_{msg} = 3$, then $N_e = 4$, and therefore $N_{sm} = 2^4$ .As seen in Figure 45 the number of real schedules that are saved nodes is 16, which is equal to $Qmsch$; although the numbering of the IDs is different and is related to the controller parameters in the scheduler (e.g., finishing tasks and not representing quantity of schedules).

The results of this example are presented in Figure 42 and Figure 43. Figure 42 represents the results of the static stack as the starting node SM_0, while Figure 43 gives the results of DS.

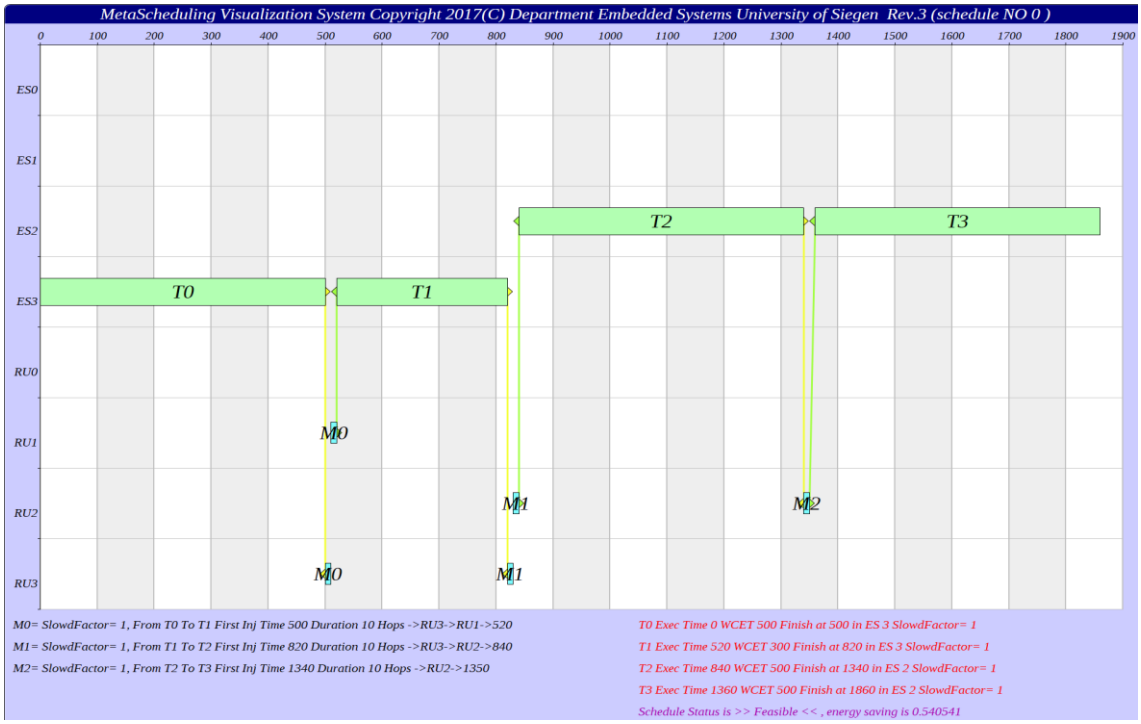


Figure 42. Schedule SM_0 with static slack (SS)

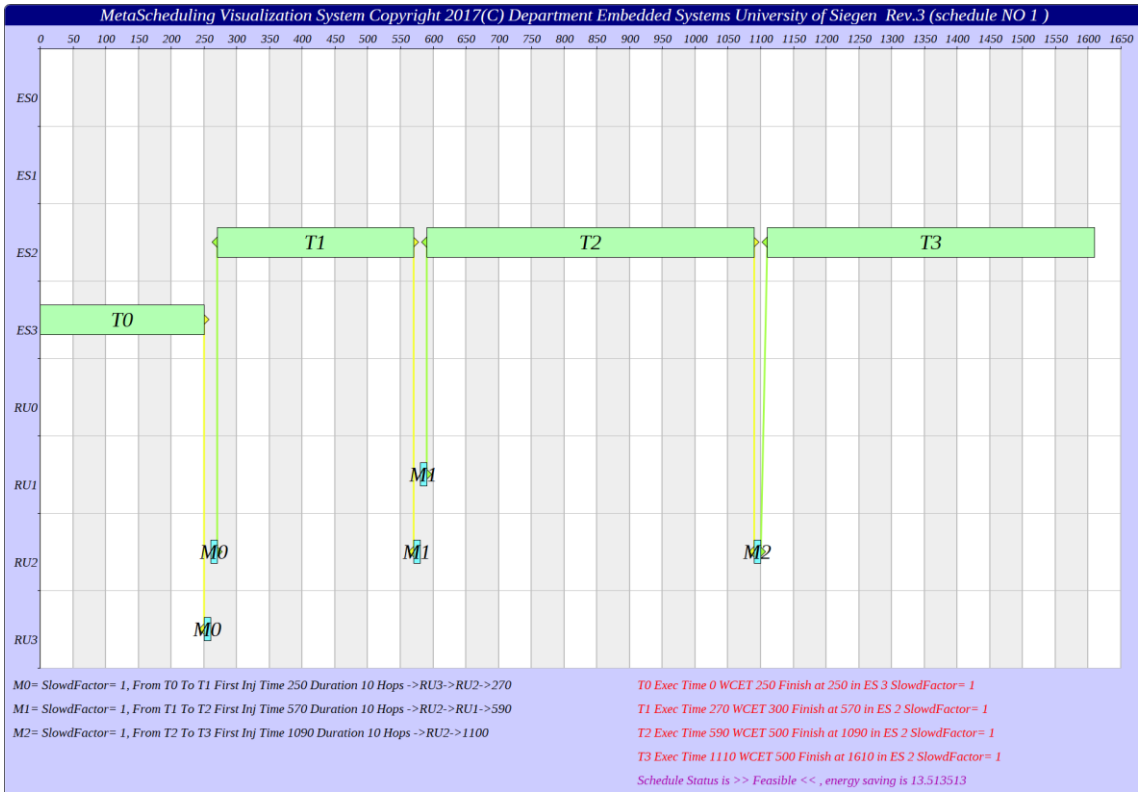


Figure 43. Schedule SM_1 with dynamic slack (DS)

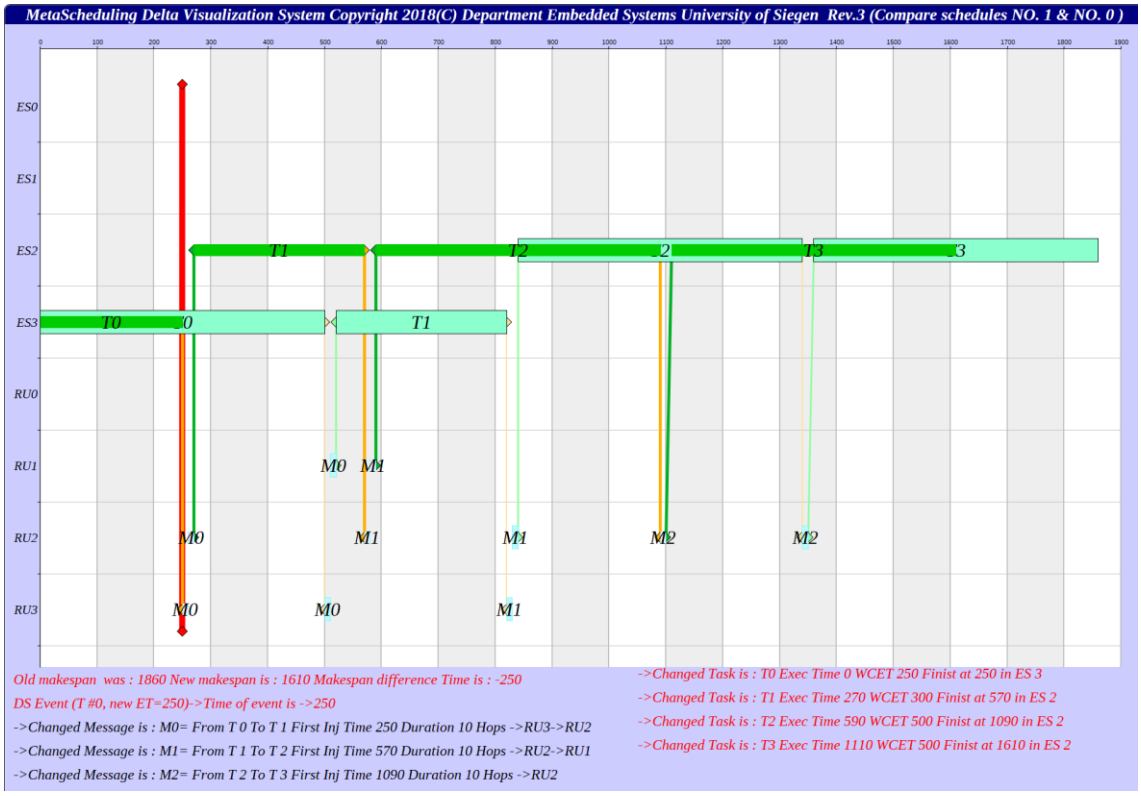


Figure 44. Comparing two schedules SM_0 (Figure 42) and SM_1 (Figure 43) after slack

Figure 44 represents a comparison between SS and DS. Figure 45 is a graph of node dependency after DS has occurred for each task, and it is generated by Graphviz using the output code below.

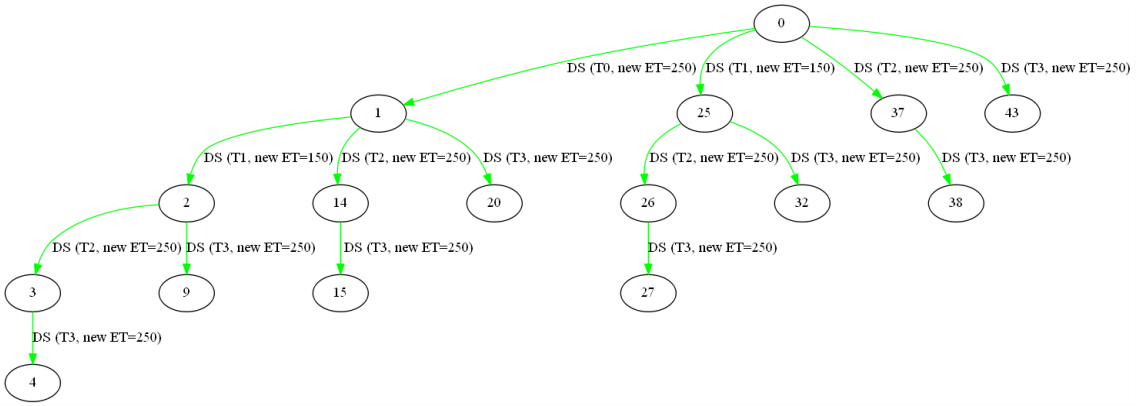


Figure 45. Graph output of node dependency

The vertical event pointer line (in red color) in Figure 44 highlights that the DS event for T0 on time $250\mu s$ and other information indicates makespan from $1860\mu s$ is reduced to $1610\mu s$ and $T0$ WCET value from $500\mu s$ is reduced to $250\mu s$. The important part of Figure 44 refers to three changes to messages and four to tasks.

The memory storage for each schedule is equal to the following:

$$M_{sm} = 5 \times 4 + 5 \times 3; M_{sm} = 35.$$

The total memory space for the whole scenario in this case is equal to the following:

$$M_{ssm} = 35 * 16; M_{ssm} = 560.$$

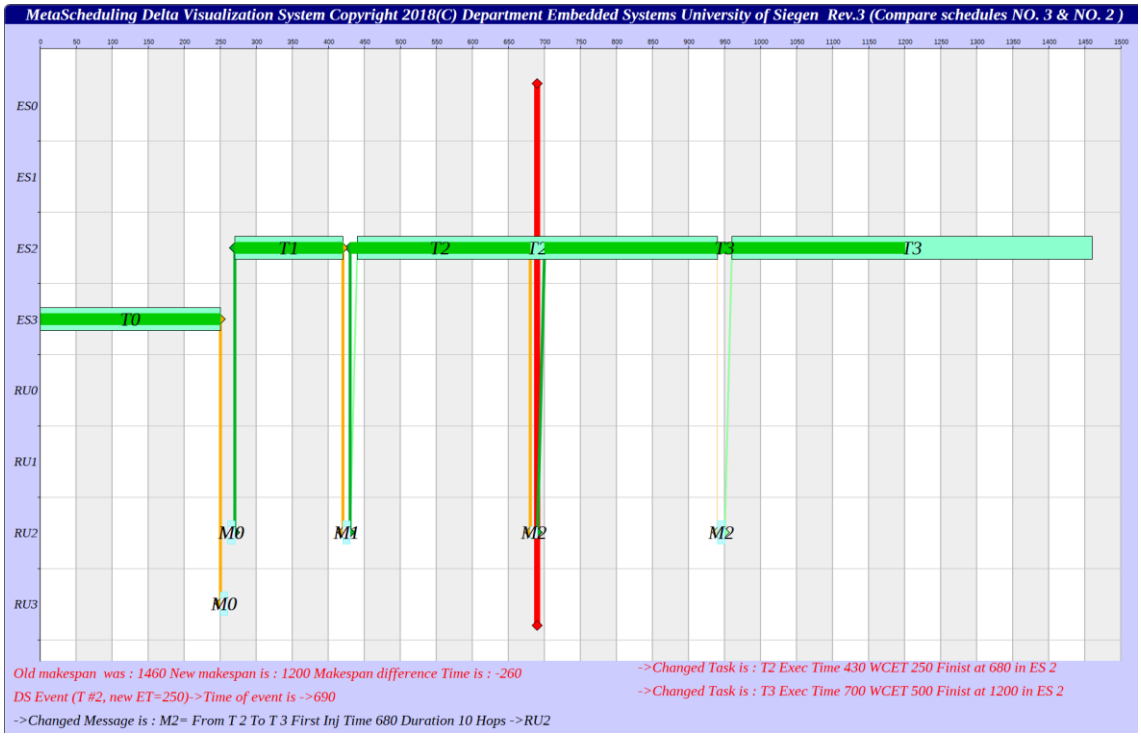


Figure 46. Few changes from ID2 to ID3

In Figure 46, the data for message $M2$ and tasks $T2$ and $T3$ changed; while in Figure 47, only the data of $T3$ changed. The data for other tasks and messages remain the same, with duplicated data for storage time, which leads to a waste of memory.

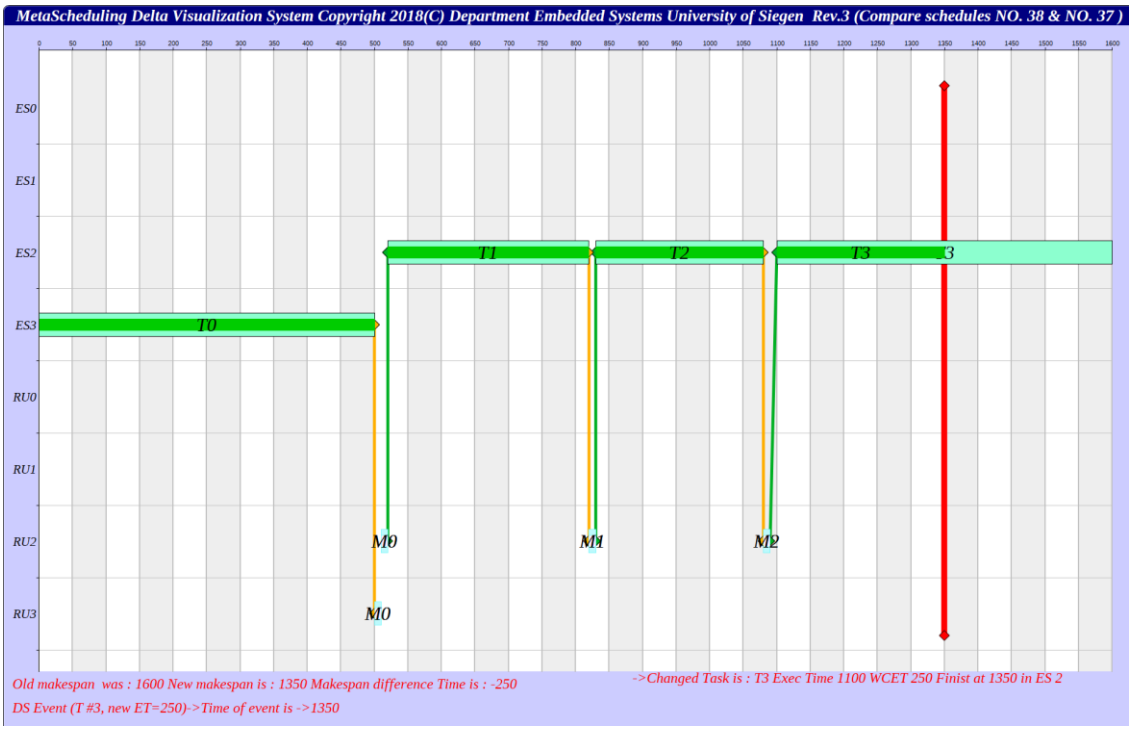


Figure 47. Minimum changes from SM_{37} to SM_{38} regarding T3 slack

7.3.1.2. Results for delta scheduling technique (DST) and delta tree (DT)

In the next examples (7.3.1.3,7.3.1.4), we extended the model to both messages and tasks in the same AM, PM, and CM, and we assumed that $M_{tsk} = 5\ Bite$ and $M_{msg} = 5\ Bite$. The results of generating the schedules are presented in Table 7.

Table 7. Results of dynamic slack (DS) schedules

sm	$N_{ch(msg)}$	$N_{ch(tsk)}$	$N_{ch(msg)} \times Mms$	$N_{ch(tsk)} \times Mts$	M_{sm}	$SavM\ Byte$	$Makespan$ (ms)	$SavE$
1	3	4	15	20	35	0	1610	14%
2	2	3	10	15	25	10	1460	22%
3	1	2	5	10	15	20	1200	36%
4	0	1	0	5	5	30	950	49%
9	0	1	0	5	5	30	1200	36%
14	1	2	5	10	15	20	1350	28%
15	0	1	0	5	5	30	1100	41%
20	0	1	0	5	5	30	1350	28%
25	2	3	10	15	25	10	1710	9%
26	1	2	5	10	15	20	1450	22%
27	0	1	0	5	5	30	1200	36%
32	0	1	0	5	5	30	1450	22%
37	1	2	5	10	15	20	1600	14%
38	0	1	0	5	5	30	1350	28%
43	0	1	0	5	5	30	1600	14%

The results in Table 8 show that, using DST, we are able to save 61% of the memory without using data compression techniques, which need more computational resources and consume more energy.

Table 8. Memory consumption and saving via delta scheduling technique (DST)

N_{msg}	N_{tsk}	M_{sm}	N_{sm}	M_{ssm}	$SavM\ (Byte)$	$SavM\%$
3	4	35	16	560	340	61%

7.3.1.3. Example 2. Sample scenario with seven tasks and five messages and $N_{ssm} = 128$

In this example, we extended scenario Example 2 to $N_{tsk} = 7$, and $N_{msg} = 5$.

Table 9. Sample results of the 128 generated schedule, and Figure 48 show how the slack event affects changes from father to child schedule.

Table 9. Sample results of Example 2

sm	$N_{ch(msg)}$	$N_{ch(tsk)}$	$N_{ch(msg)} \times Mms$	$N_{ch(tsk)} \times Mts$	M_{sm}	$SavM$ $Byte$	$Makespan$ (ms)	$SavE$
1	5	6	25	30	55	5	1610	7%
2	2	3	10	15	25	35	1610	17%
3	4	5	20	25	45	15	1460	22%
108	1	2	5	10	15	45	1391	23%
109	0	2	0	10	10	50	1381	31%
110	0	1	0	5	5	55	1131	38%
114	0	1	0	5	5	55	1210	31%
120	0	1	0	5	5	55	1541	26%
122	0	1	0	5	5	55	1291	34%
128	0	1	0	5	5	55	1281	26%
108	1	2	5	10	15	45	1391	23%
109	0	2	0	10	10	50	1381	31%
110	0	1	0	5	5	55	1131	38%
114	0	1	0	5	5	55	1210	31%
120	0	1	0	5	5	55	1541	26%

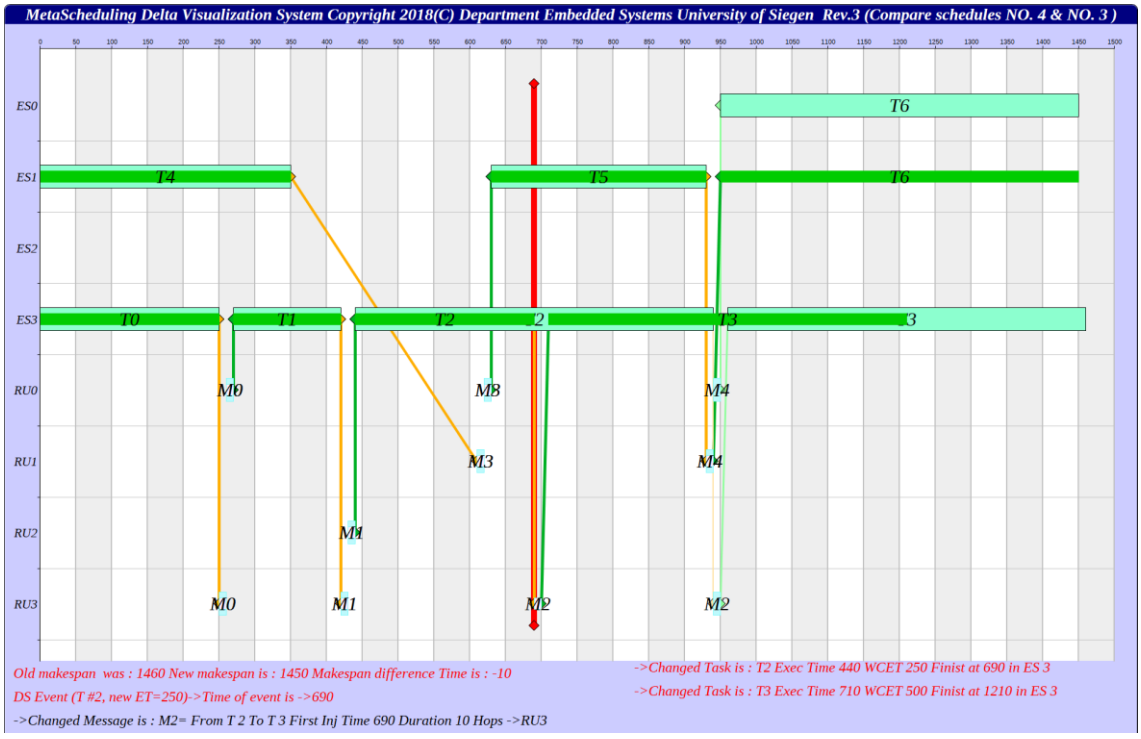


Figure 48. Meta-visualization for Example 3 (schedules SM_3 & SM_4)

Table 10. Results of delta scheduling technique (DST) memory saving for Example 2

N_{msg}	N_{tsk}	M_{sm}	N_{sm}	M_{ssm}	$SavM$ $(Byte)$	$SavM\%$
5	7	60	128	7680	6335	82%

The results in Table 10 indicate that, using DST, we are able to save approximately 82% of the memory space.

7.3.1.4. Example 3. Big sample with $N_{tsk} = 9$, $N_{msg} = 8$, and $N_{ssm} = 512$ schedules

The last test was designed with $N_{tsk} = 9$, $N_{tsk\_s} = 9$, and $N_{msg} = 8$ message.

Table 11. Sample results for Example 3

sm	$N_{ch(msg)}$	$N_{ch(tsk)}$	$N_{ch(msg)} \times Mms$	$N_{ch(tsk)} \times Mts$	M_{sm}	$SavM$	$Makespan$	$SavE$
1	4	5	20	25	45	35	1860	2%
2	3	4	15	20	35	45	1733	10%
3	1	2	5	10	15	65	1733	14%
138	2	3	10	15	25	55	1663	22%
139	0	2	0	10	10	70	1653	29%
197	0	2	0	10	10	70	1543	24%
204	0	2	0	10	10	70	1704	21%
245	0	2	0	10	10	70	1350	36%
246	0	1	0	5	5	75	1100	43%
326	0	1	0	5	5	75	1213	37%
434	0	1	0	5	5	75	1200	42%
469	0	1	0	5	5	75	1710	27%
471	0	1	0	5	5	75	1460	34%
509	0	2	0	10	10	70	1600	29%
510	0	1	0	5	5	75	1350	36%

Table 12. Results of delta scheduling technique (DST) memory saving for Example 3

N_{msg}	N_{tsk}	M_{sm}	N_{sm}	M_{ssm}	$SavM\ (Byte)$	$SavM\%$
7	9	80	512	40960	35065	86%

The results in Table 12 indicate that, using DST, we are able to save approximately 82% of memory space.

7.3.1.5. Discussion

Compared to the DeltaGraph generator [25, 128] results for communication, which show the memory is optimized by more than 50% in the best case, our results in Figure 49 indicate that DTS can be optimized by 61% to 86% in the best case (c.f., sections 7.3.1.1,7.3.1.3, and 7.3.1.4).

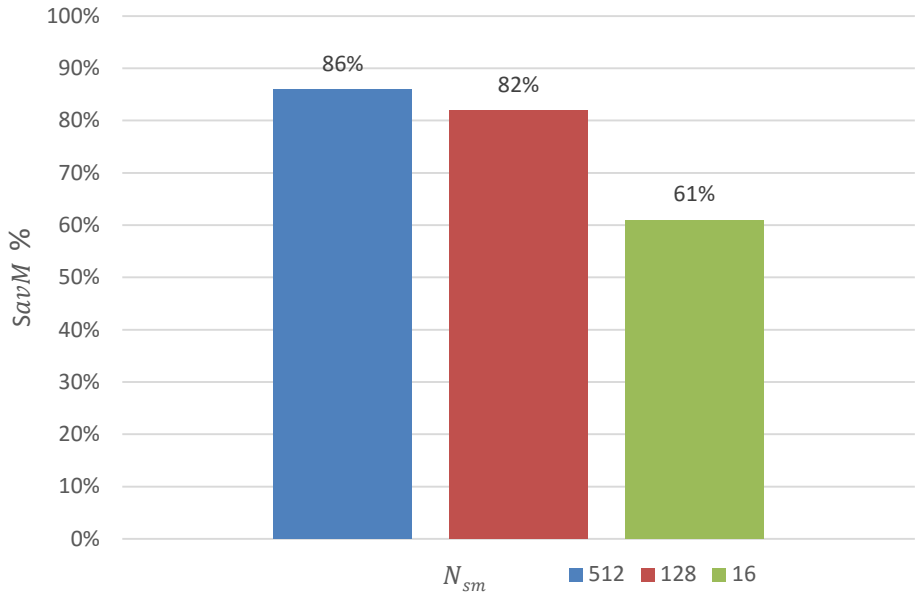


Figure 49. Comparison of three different scenarios for memory saving with delta scheduling technique (DTS)

Figure 49 shows that by increasing the N_{ssm} with DST, the quantity of $N_{ch(msg)}$ and $N_{ch(tsk)}$ increases, which causes the system to save memory and $SavM$ increases. Therefore, $M_{real(ssm)}$ better guarantees that the permissible memory limit will not be exceeded.

$$DST\ is\ active\ \rightarrow\ M_{real(ssm)} <\ M_{ssm}\ < M_{lim} \quad (59)$$

7.3.2. Scenario-based meta-scheduling (SBMeS) for frequency scaling of processors

Our goal in this section is to optimally schedule the messages and tasks, such that the time constraints (i.e., task and message deadlines) are satisfied, while total energy consumption is decreased. A novel algorithm is introduced for scenario-based mapping and scheduling of tasks and messages onto the ***NoC*** platform. The objective is to minimize energy consumption due to the scenarios.

7.3.2.1. Decision variables

The first step to solving the scheduling problem in ***MeS*** is defining the scenarios, decision variables, constants, and constraints.

7.3.2.2. Slow-down factor

In this case study, we are using only $TSDF$ as a decision value.

7.3.2.3. Objective function

The objective is to maximize energy efficiency. In other words, we wish to minimize energy consumption by increasing task execution times due to the *TSDF* by reducing the frequency of the cores.

$$\forall t \in \text{TSK}.\ \forall \text{m} \in MSG$$

$$CP_3(t) = et(t) \cdot tsdf(t)^2 \qquad (60)$$

$$RES_3 = maximize\ (\sum_{i=1}^{|TSK|}(CP_3(t_i)) \qquad (61)$$

7.3.2.4. Input metadata for meta-scheduling (MeS)

This section discusses the results of the ***MIQP*** model described above and evaluates the ***MeS*** results, as visualized with ***MeSViz***.

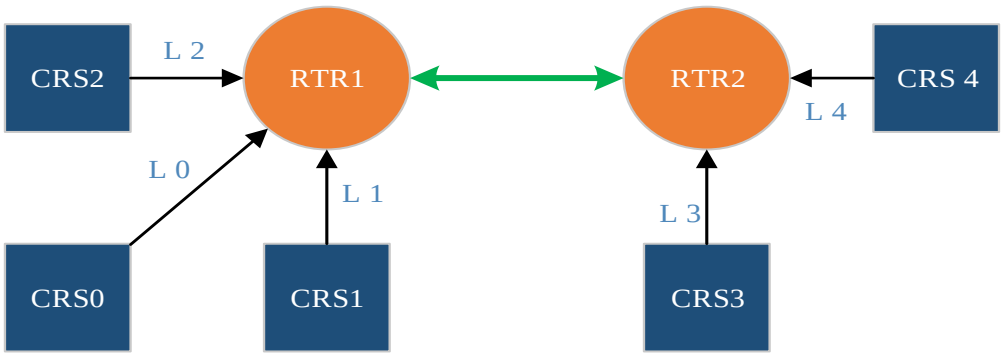


Figure 50. The physical model (PM) of case study

Input models: The SM of the case which was designed and tested is shown in Table 15 and Figure 51 for AM and Figure 50 for PM.

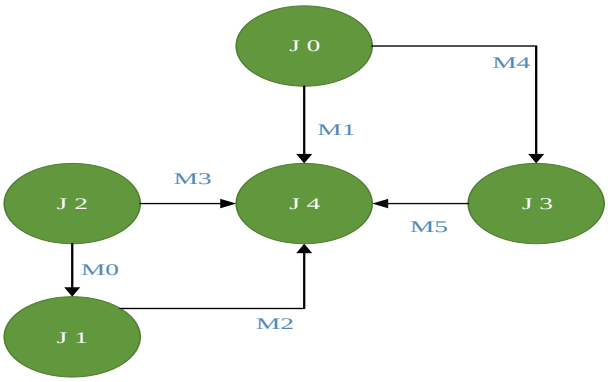


Figure 51. The application model (AM) of case study

Table 13. Meta-scheduling (MeS) input constant

Input Model	Input Name	ID	Data
AM	TSK	0	WCET=2
		1	WCET=4
		2	WCET=6
		3	WCET=8
		4	WCET=10
	MSG		Quantity=6 ID start=0
	Deadline ($D(m)$)		All task= 1185
	$tsdf(t)$		All task: min =1 & max =100
	Slack event		All task= 50%
PM	RTR		Quantity=2 ID start=0
	CRS		Quantity=5 ID start=6
	$Link$		Quantity=6 ID start=0

7.3.2.5. Outputs and results

Output results: ***MeS*** generates 94 schedules. ***MeSViz*** generates 31 meta-visual Gantt maps from ***MeS***. GVEdit generates a graph map with 94 SMs.

7.3.2.6. Overhead

Table 14. Results of delta scheduling technique (DST) memory saving

N_{msg}	N_{tsk}	M_{sm}	N_{sm}	M_{ssm}	$SavM\ (Byte)$	$SavM\%$
6	5	55	94	5170	4037	78%

The results presented in Table 14 show that, using DST, we are able to save approximately 78% of the memory space.

7.3.2.7. Discussion

Three samples of collected data from SM outputs are shown in Table 15: the first schedule (SM_0) with SS technique, schedule SM_{49} with minimum energy reduction, and schedule SM_5 with maximum energy reduction using the DS technique.

Table 15. Some collected results for model example

SM ID	*Slack Mode*	*Task ID*	$et(t)$	$tsdf(t)$
0	Static	0	2	4
		1	4	4
		2	6	4
		3	8	4
		4	10	4
49 (Minimum $SavE_{sm}$)	Dynamic	0	1	4
		1	4	4
		2	3	5
		3	8	4
		4	10	4
5 (Maximum $SavE_{sm}$)	Dynamic	0	1	4
		1	2	4
		2	3	5
		3	4	5
		4	5	5

In Table 16, the slack mode, energy consumption, and energy-saving of each SM and the average energy consumption (total DS SMs) are compared to those of the SS SM_0. The

results show that the ***MeS*** algorithm can reduce energy usage by DS in robust and adaptive ***TTS***.

Table 16. Energy results for example

SM	*Slack Mode*	$\boldsymbol{FE(sm)}$	$\boldsymbol{ReFE_{sm}}$
0	Static	1.875	0
49	Dynamic	1.62	13.6%
5	Dynamic	0.6675	64.4%
Average	Dynamic	1.0948	41.61%

7.3.3. Scenario-based meta-scheduling (SBMeS) for frequency scaling of processors and network-on-chip (NoC)

Communication and task deadlines are critical constraints that scheduling algorithms and scheduling tools for real-time systems must satisfy. Scheduling algorithms must also have a significant impact on energy efficiency. For example, the authors in [22] worked on task scheduling and time constraints to save energy without degrading performance. However, computing costs and task deadlines are not guaranteed in many algorithms.

In [132], "as soon as possible" (ASAP) and "as late as possible" (ALAP) techniques for task scheduling are used, but communication scheduling is not considered. Many studies explain that an interconnection network should be designed to be power-aware and energy-efficient for satisfying system-level requirements for energy and power (e.g., [133]).

In this case, we scheduled each task on a single core of the multi-core processor, such that the time constraints (i.e., task and message deadlines) were satisfied, while the total energy consumption was decreased for both cores and routers. Our primary objective was to find and minimize the total energy consumption by maximizing the $SDFs$ for computations and communication activities using DS and SS event scenarios.

7.3.3.1. Scheduling constraints

Task assignment: In this case study, the task assignment strategy involved assigning one task to one core.

7.3.3.2. Decision variables

We used both *TSDF* and *MSDF*.

7.3.3.3. Input metadata for meta-scheduling (MeS)

The input data for the case, which was designed and tested, is shown in Table 17. In the PM presented in Figure 53, we used a sample network based on the ***MPSoC*** design.

The ***MeS*** scheduling and routing methodology is independent of the hardware platform and can support different network topologies (e.g., mesh, direct network, indirect network, balanced tree).

To compare and measure the efficiency and effectiveness of our method, we used equal AM and PM structures, as seen in section 7.3.2. The AM and scheduler algorithm were also extended to use $SDFs$ for routers.

Table 17. Meta-scheduling (MeS) input constant devices [13]

Input Model	Input Name	ID	Data
AM	TSK	0	WCET=2
		1	WCET=4
		2	WCET=6
		3	WCET=8
		4	WCET=10
	MSG		Quantity=6 ID start=0
	Deadline ($D(m)$)		All task= 1185
	$tsdf(t)$		All task and messages: min =1 & max =100
	Slack event		All task= 50%
PM	RTR		Quantity=2 Start ID==0
	CRS		Quantity=5 Start ID==6
	$Link$		Quantity=6 Start ID=0

Figure 52 represents the dependencies of the tasks t and messages m, which are used in the AM. Figure 53 represents the $cores$ CRS, $routers$ RTR, and their connectivity ($Link$), which are used in the PM.

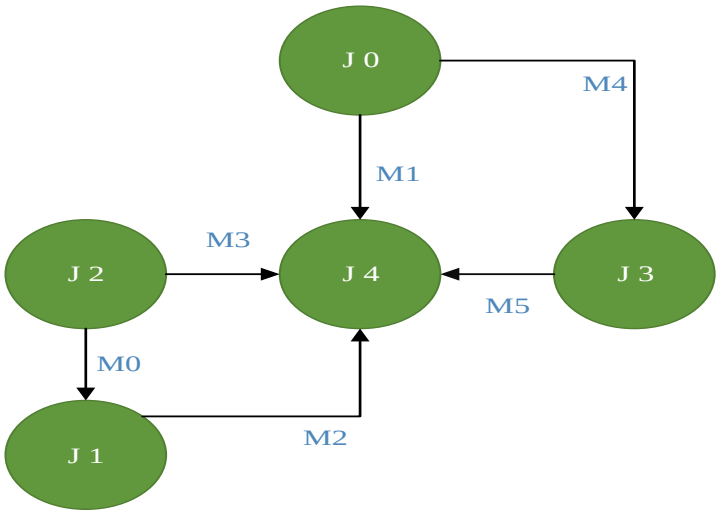


Figure 52. The application model (AM) of the case study [6]

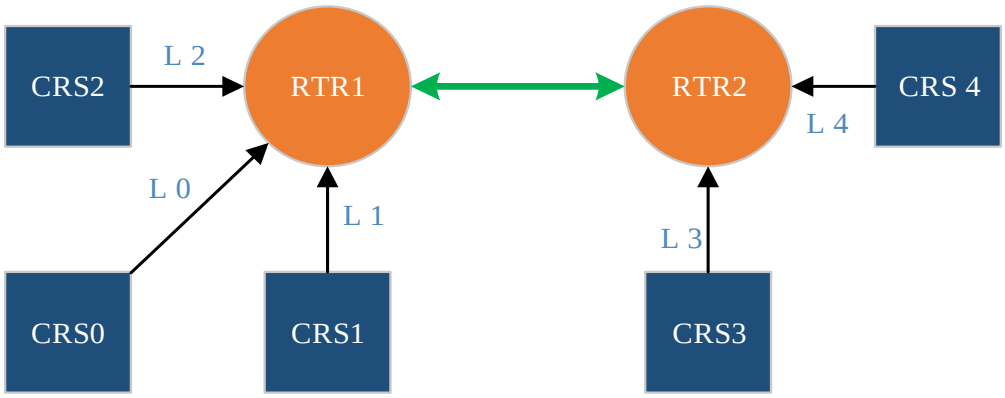


Figure 53. The physical model (PM) of the case study 7.3.2

7.3.3.4. Outputs and results

MeS generates 94 schedules for slack events. GVEdit generates a graph map with 94 SMs.

7.3.3.5. Discussion

In Table 19, the results of 14 samples of collected data from SM outputs are presented. The first schedule was generated using the SS technique, with schedule SM_{49} seeing minimum energy reduction, while schedules (SM_1 to SM_{93}) used the DS technique. The slack mode, energy consumption, and energy reduction s of each SM and average energy consumption (total DS SMs) are compared to the SS SM, as presented in Figure 54, Figure 55, Figure 56, and Figure 57.

The results in 7.3.2, presented in Table 18, indicate an average 41.61% energy reduction at cores only. However, in this work, it is observed that energy reduction in cores is reduced by 8.5% compared to the results in 7.3.2. In other words, energy reduction is 33.11% for cores, while routers reach 22.66%.

Table 18. Energy results for Example 7.3.2

SM	*Slack Mode*	$\boldsymbol{FE(sm)}$	$\boldsymbol{ReFE_{sm}}$
0	Static	1.875	0
49	Dynamic	1.62	13.6%
5	Dynamic	0.6675	64.4%
Average	Dynamic	1.0948	41.61%

These results confirm our extended ***SBMeS*** algorithm method, with the combination of DS and $SDFs$ reduces energy consumption in the ***TT MPSoCs***. The system has better energy efficiency in the ***NoC***, while being robust and adaptive in the use of SDFs for both cores and routers.

Table 19. General results for a model example

SM ID	CRS	RTR	Re CRS	Re RTR	Total	Re Total
0	1.65	4.03333	0.0000%	0.0000%	5.68333	0.0000%
1	1.3625	2.77778	17.4242%	31.1295%	4.14028	27.1505%
2	1.3325	3.63333	19.2424%	9.9174%	4.96583	12.6246%
5	0.6675	3.14444	59.5455%	22.0386%	3.81194	32.9277%
18	0.9925	3.1444	39.8485%	22.0386%	4.1369	27.2099%
38	1.0225	3.05556	38.0303%	24.2424%	4.07806	28.2452%
44	0.9825	3.05556	40.4545%	24.2424%	4.03806	28.9490%
46	1.1825	3.05556	28.3333%	24.2424%	4.23806	25.4300%
49	1.4625	3.63333	11.3636%	9.9174%	5.09583	10.3372%
63	0.9225	3.14444	44.0909%	22.0386%	4.06694	28.4409%
70	1.2825	3.14444	22.2727%	22.0386%	4.42694	22.1066%
80	1.1	3.05556	33.3333%	24.2424%	4.15556	26.8816%
93	1.38	3.05556	16.3636%	24.2424%	4.43556	21.9549%
Avg	1.10363	3.11918	33.1134%	22.6651%	4.22281	25.6983%

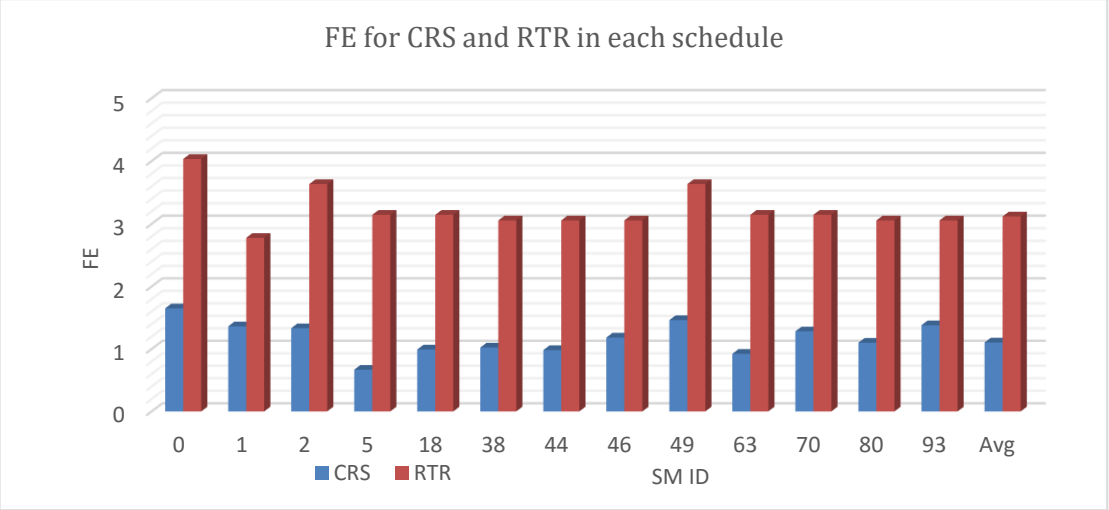


Figure 54. *Energy consumption results for cores and routers* $FE_C(sm)$, $FE_R(sm)$, $FE_{C,avg,dyn}$, $FE_{R,avg,dyn}$

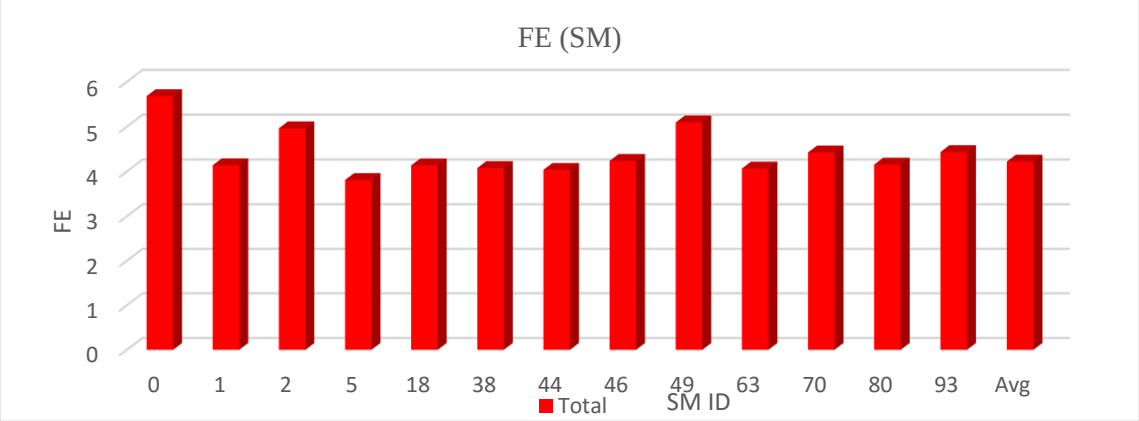


Figure 55. $FE(sm)$ *schedule models(SMs) results and average* $FE_{avg,dyn}$

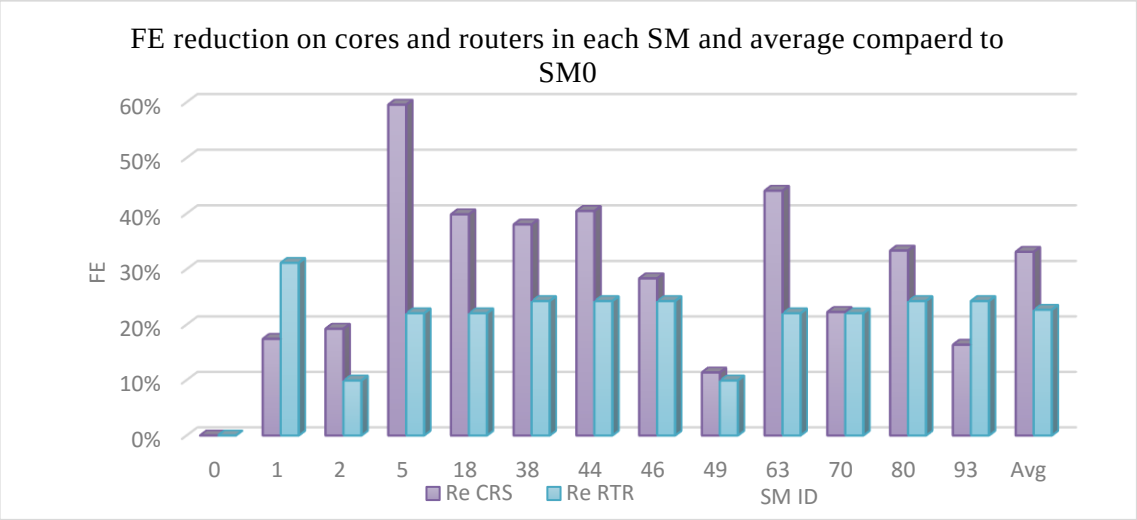


Figure 56. Total energy reduction results for cores$ReFE_C(SM)$ and routers $ReFE_R(SM)$ and average$FE_{C,avg,dyn}$, $FE_{R,avg,dyn}$

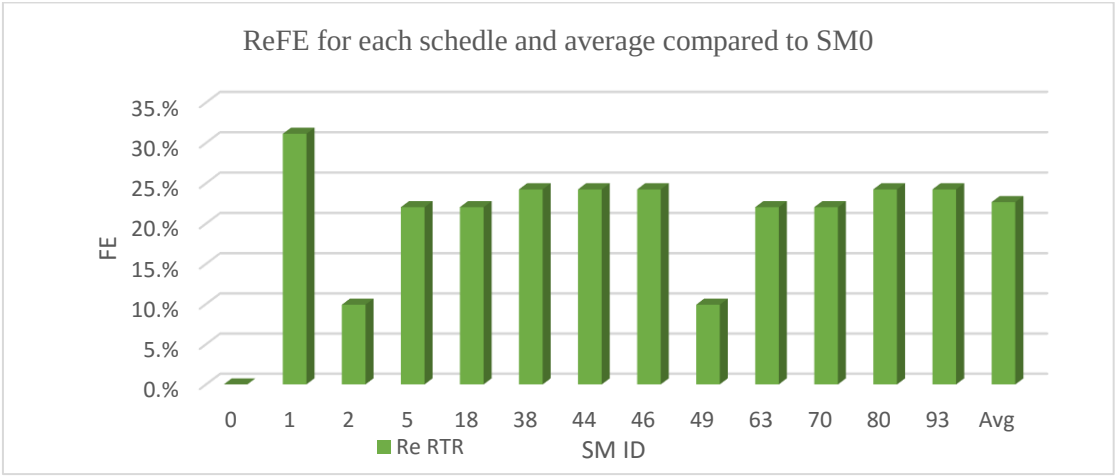


Figure 57.Total FE reduction results for schedules ReE_{sm} and average $ReFE$

7.3.3.6. Overhead

Table 20. Results of delta scheduling technique (DST) memory saving

N_{msg}	N_{tsk}	M_{sm}	N_{sm}	M_{ssm}	$SavM\ (Byte)$	$SavM\%$
6	5	55	94	5170	3980	77%

The results in Table *20* show that, using DST, we are able to save approximately 77% of the memory space. Compare to overhead in 7.3.2.6, due to use of $MSDF$ and $TSDF$ in this case, $SavM$ about 1% reduced.

7.3.4. Scenario-based meta-scheduling (SBMeS) for frequency scaling with flexible task-to-processor mapping

Unlike in sections 7.3.3 and 7.3.2, where only a single task was assigned per core, the algorithm presented in this section is able to assign multiple tasks per core on a multi-core platform. In addition, this section does not introduce a new architecture for ***SBMeS***, but rather an improved, extended, and more flexible and reliable ***SBMeS*** architecture for ***MPSoCs*** and ***NoCs***.

In this section, we examine the importance of the methods and techniques discussed. This experiment and its results are intended to answer the question of why we require the ***SBMeS*** and DS technique and special energy reduction schemes for both cores and routers.

7.3.4.1. Input models

The input data for the designed use case are shown in Table 17. Figure 58 concerns the AM, while Figure 59 and Figure 53 concern the PM of the case study.

Table 21. Meta-scheduling (MeS) input constant

Input Model	Input Name	ID	Data
AM	TSK	0	WCET=2
		1	WCET=4
		2	WCET=6
		3	WCET=8
		4	WCET=10
	MSG	Quantity=6 ID start=0	
	$TSDF$	All Task: min =1 & max =100	
	$MSDF$	All Messages : min =1 & max =100	
	Slack event	All Task= 50%	
	RTR	Quantity=3 ID start=0	
PM	CRS	Quantity=5 ID start=6	
	$Link$	Quantity=9 ID start=0	

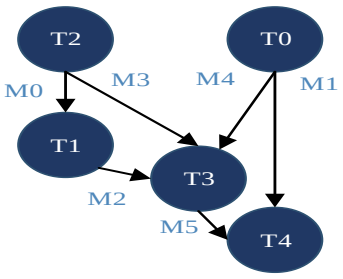


Figure 58. The application model (AM) of the case study

In the AM (Figure 58), $T2$ and $T0$ are starting tasks and $T4$ is the final task. In PM, $ES2$ with links $L4$ and $L6$ is connected to two hops, $H1$ and $H2$. The connection of $ES2$ via

two links provides greater reliability for the system. This example shows how ***MeS*** can be used to model reliability for safety-critical systems.

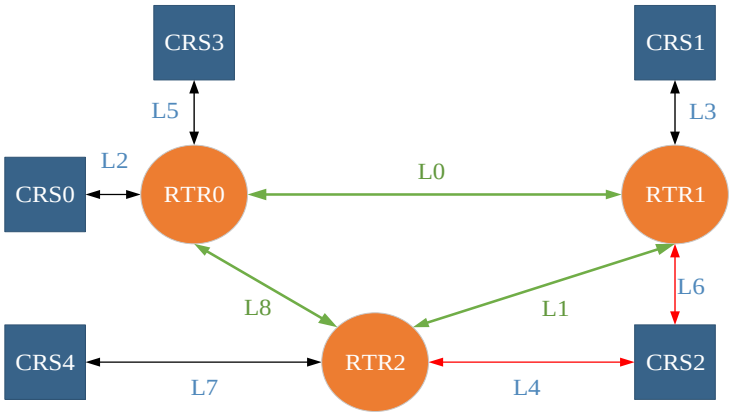


Figure 59. The physical model (PM) of the case study

7.3.4.2. Outputs and results

MeS at 200 seconds generated 93 schedules for the DS scenario SSM/$\{SM_0\}$ and one schedule for the SS SM_0. GVEdit created a graph map from *MeSViz* output with 94 SMs.

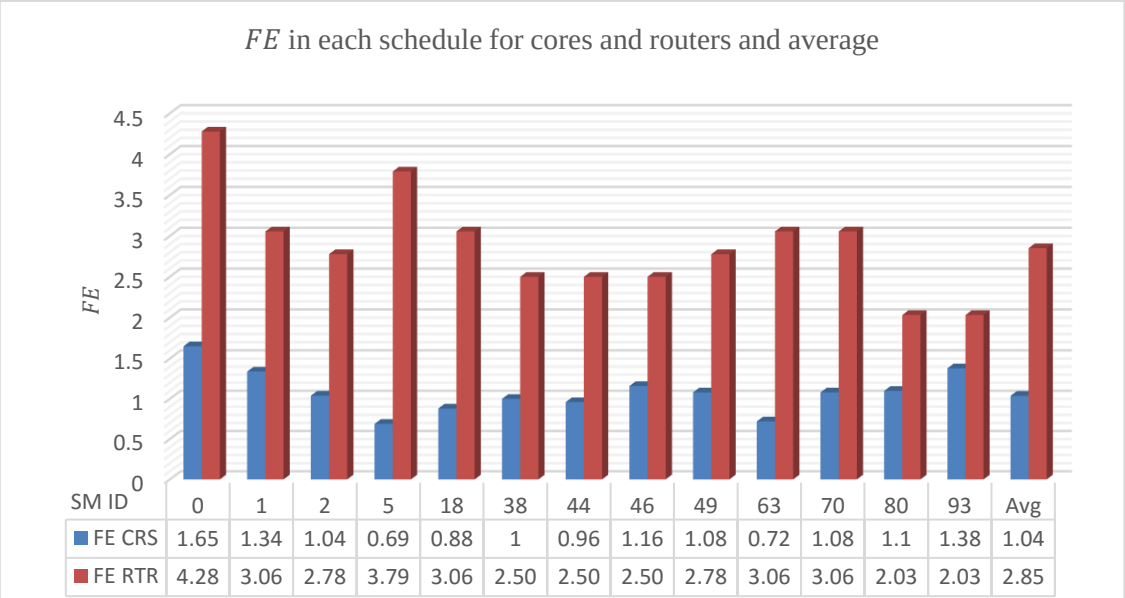
7.3.4.3. Discussion

In Table 22, the results of 14 samples of collected data from SM outputs are presented. The schedule SM_0 is used for the SS time, and the schedule SM_1 to SM_{93} is used for DS time. As seen in the results, schedule SM_5 has lowest energy reduction and schedule SM_{80} has the highest.

Table 22. General results of FE for the example model

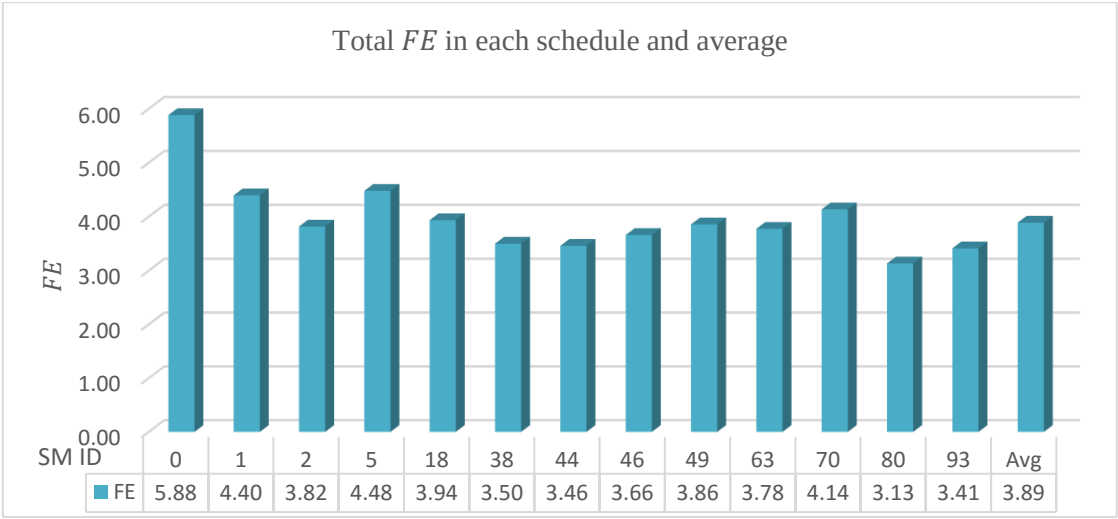
SM ID	FE CRS	FE RTR	ReFE Core	ReFE RTR	FE	ReFE Total
0	1.65	4.28	0.00%	0.00%	5.88	0.00%
1	1.34	3.06	16.51%	28.57%	4.40	25.28%
2	1.04	2.78	35.20%	35.06%	3.82	35.10%
5	0.69	3.79	57.01%	11.43%	4.48	23.86%
18	0.88	3.06	45.17%	28.57%	3.94	33.10%
38	1	2.50	37.69%	41.56%	3.50	40.50%
44	0.96	2.50	40.19%	41.56%	3.46	41.18%
46	1.16	2.50	27.73%	41.56%	3.66	37.78%
49	1.08	2.78	32.71%	35.06%	3.86	34.42%
63	0.72	3.06	55.14%	28.57%	3.78	35.82%
70	1.08	3.06	32.71%	28.57%	4.14	29.70%
80	1.1	2.03	31.46%	52.47%	3.13	46.74%
93	1.38	2.03	14.02%	52.47%	3.41	41.98%
Avg	**1.04**	**2.85**	**35.26%**	**33.37%**	**3.89**	**33.89%**

The FE for static SM_0 and DS modes SSM/SM_0 results, FE reductions $ReFE(SSM/SM_0)$, an average of FE (for cores and routers) and $ReFE_{total}$, as compared to the SM_0, are shown in Figure 60, Figure 61, Figure 62, and Figure 63.



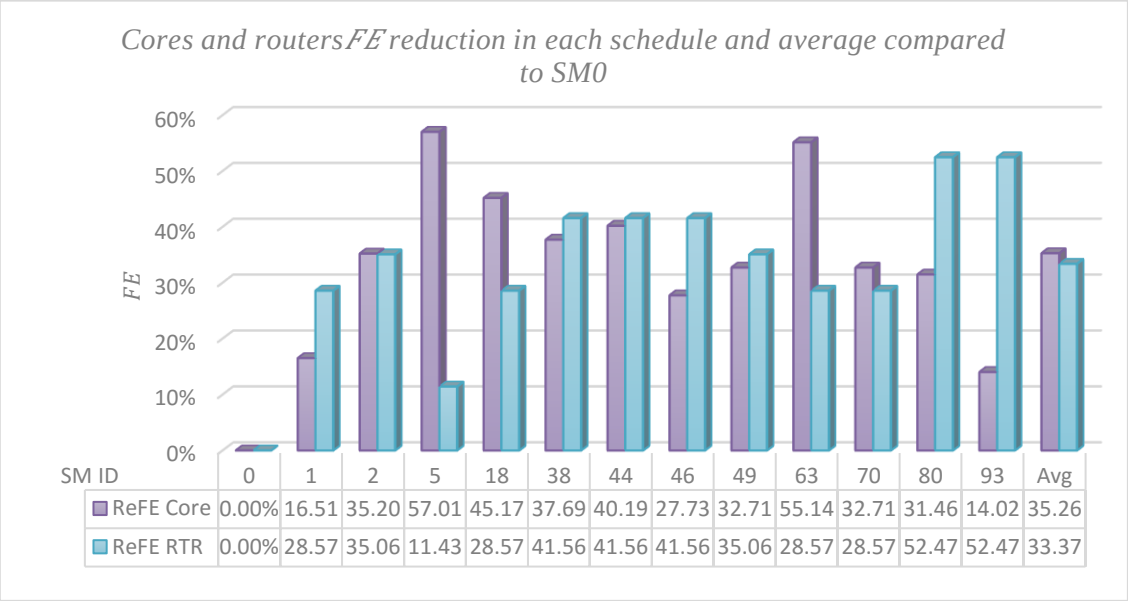
SM ID	0	1	2	5	18	38	44	46	49	63	70	80	93	Avg
FE CRS	1.65	1.34	1.04	0.69	0.88	1	0.96	1.16	1.08	0.72	1.08	1.1	1.38	1.04
FE RTR	4.28	3.06	2.78	3.79	3.06	2.50	2.50	2.50	2.78	3.06	3.06	2.03	2.03	2.85

Figure 60. $FE_C(sm), FE_R(sm)$ results for cores and routers and average $FE_{C,avg,dyn}$, $FE_{R,avg,dyn}$



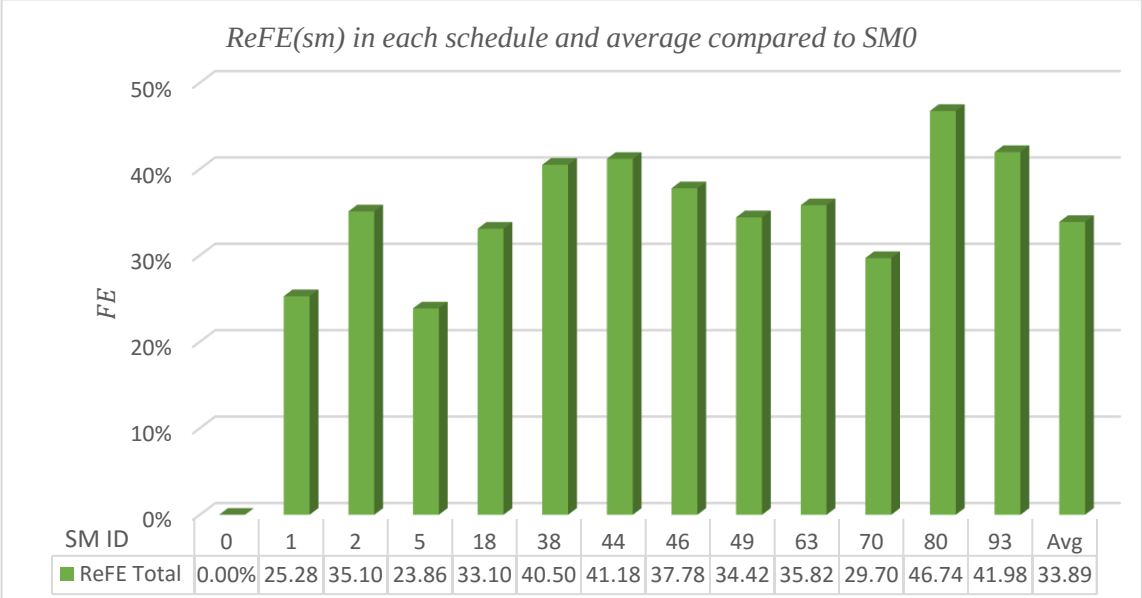
SM ID	0	1	2	5	18	38	44	46	49	63	70	80	93	Avg
FE	5.88	4.40	3.82	4.48	3.94	3.50	3.46	3.66	3.86	3.78	4.14	3.13	3.41	3.89

Figure 61. $FE(sm)$ results and average $FE_{avg,dyn}$



SM ID	0	1	2	5	18	38	44	46	49	63	70	80	93	Avg
ReFE Core	0.00%	16.51	35.20	57.01	45.17	37.69	40.19	27.73	32.71	55.14	32.71	31.46	14.02	35.26
ReFE RTR	0.00%	28.57	35.06	11.43	28.57	41.56	41.56	41.56	35.06	28.57	28.57	52.47	52.47	33.37

Figure 62. FE results for cores $ReFE_C(SM)$ and routers $ReFE_R(SM)$ compare to SM_0 and average $FE_{C,avg,dyn}$, $FE_{R,avg,dyn}$



SM ID	0	1	2	5	18	38	44	46	49	63	70	80	93	Avg
ReFE Total	0.00%	25.28	35.10	23.86	33.10	40.50	41.18	37.78	34.42	35.82	29.70	46.74	41.98	33.89

Figure 63. Total $ReFE(sm)$ in each schedule compare to SM_0 and average $FE_{avg,dyn}$

These results confirm our theory that, compared to SS time, DS time together with our ***MeS*** algorithm can reduce power consumption of ***NoCs*** by using frequency scaling for both cores and routers.

7.3.4.4. Overhead

Table 23. Results of delta scheduling technique (DST) memory saving

N_{msg}	N_{tsk}	M_{sm}	N_{sm}	M_{ssm}	$SavM\ (Byte)$	$SavM\%$
6	5	55	94	5170	4208	81%

The results in Table 23 indicate that, using DST, we are able to save approximately 81% of the memory space.

7.3.5. Scenario-based meta-scheduling (SBMeS) for improved reliability

7.3.5.1. Input models

The base input data for all cases are shown in Table 17. Figure 65 is for AM and Figure 64 for PM.

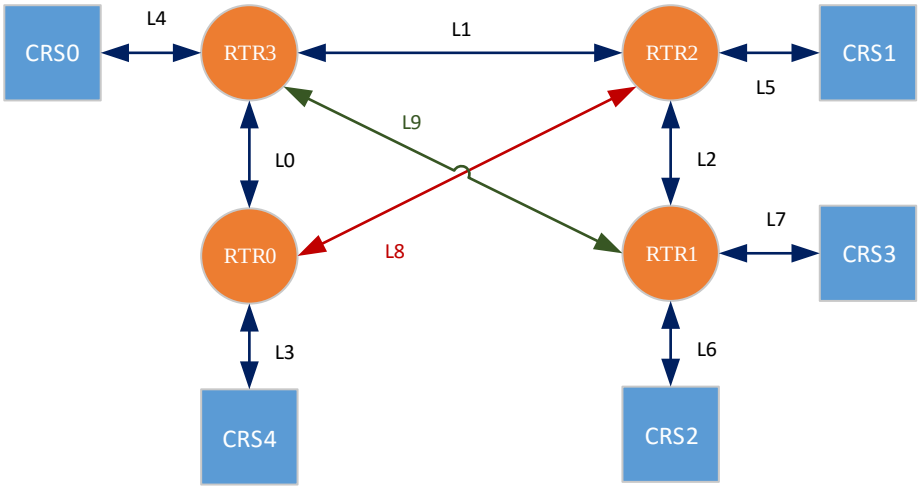


Figure 64. The physical model (PM) of the case study

In the AM, $T0$ is the first tasks and $T4$ and $T5$ are the last task. In the PM, $l8$ and $l9$ are used for reliability and flexibility in routing.

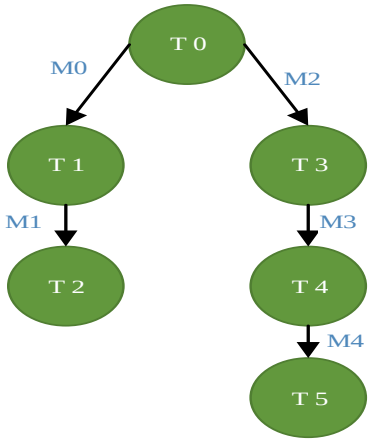


Figure 65. The application model (AM) of case study

Table 24. The context model (CM) for application model (AM) and physical model (PM) scenarios

$Status_x$	Group	Scenario	Test	CRS	RTR	TSK	MSG	Links-ID	Slack	$MSDF$	$TSDF$	$flt(c)$
1	I	1	1	5	4	6	5	9	6	6	6	0
2	I	1	2	5	4	6	5	9	3	6	6	0
3	I	1	3	5	4	6	5	9	2	6	6	0
4	I	1	4	5	4	6	5	9	1	6	6	0
5	I	2	1	5	4	6	5	9	6	6	6	1
6	I	2	2	5	4	6	5	9	6	6	6	2
7	I	3	1	5	4	6	5	9	0	6	6	1
8	I	3	2	5	4	6	5	9	0	6	6	2
9	I	4	1	5	4	6	5	9	1	6	6	1
10	I	4	2	5	4	6	5	9	1	6	6	2
11	I	4	3	5	4	6	5	9	1	6	6	3
12	II	5	1	5	4	6	5	7	1	6	6	1
13	II	5	2	5	4	6	5	7	1	6	6	2
14	II	6	1	5	4	6	5	7	6	6	6	0
15	II	6	2	5	4	6	5	7	3	6	6	0
16	II	6	3	5	4	6	5	7	0	6	6	1
17	II	7	1	5	4	6	5	7	6	1	6	0
18	II	7	2	5	4	6	5	7	3	1	6	0
19	II	7	3	5	4	6	5	7	1	1	6	0
20	II	8	1	5	4	6	5	7	3	6	1	0
21	II	8	2	5	4	6	5	7	1	6	1	0
22	II	9	1	5	4	6	5	7	0	1	6	1
23	II	9	2	5	4	6	5	7	0	1	6	2
24	II	9	3	5	4	6	5	7	0	1	6	3

Table 24 shows the 24 scenario for CM, PM, and AM used in 24 statuses. Nine scenarios (groups of the same color) in two groups . Group I includes all links and group II without two links $l8$ and $l9$.

For example, in group II, scenario five, test one (II5T1 $Status_{12}$), links $l8$ and $l9$ are disabled, one task has slack, and one core failed. In I4T1 $Status_9$, every parameter is the same, with the exception of the links connection. Comparing II8T1 $Status_{20}$ to II6T1 $Status_{14}$, it has one core fault without any slack, and the coefficient MSlowDF for both is one, which means TSDF is disabled.

Table 25. Meta-scheduling (MeS) input constant

Input Model	Input Name	ID	Data
AM	Task	0	WCET=2
		1	WCET=4
		2	WCET=6
		3	WCET=8
		4	WCET=4
		5	WCET=4
	MSG	Quantity=5ID start=0 Du_m=4	
	Deadline ($D(m)$)	All Task: min =1 & max =100	
	$tsdf(t)$	All Messages : min =1 & max =100	
	Slack event	All Task= 50%	
	$Hop(RTR)$	Quantity=4 ID start=0	
PM	CRS	Quantity=5 ID start=6	
	$Link$	Quantity=10 ID start=0	
CM (FaultEvent)	type=crash	type= Slack	
Makespan	250-1000		

7.3.5.2. Outputs and results

Output results*: *MeS generates 13 to 1146 schedules for different scenarios. Table 26 presents the results of SSM_x for each scenario $Status_x$; thus, for each scenario, total run time measured by $Times = Mes + MesVis$ run times (second), *total* $ReFE$ (SS schedule energy consumption, compared to an average of DS schedules) and FE (all cores and routers) $FE_{avg,dyn}$ of each scenario are measured and computed.

Table 26. Output results

SSM_x	Total N_{sm}	Effective N_{sm}	Real N_{sm}	Times (seconds)	$FE_{avg,dyn}$	$ReFE$
1	191	127	64	416	2.35	23,63%
2	46	38	8	348	2.49	19,89%
3	27	23	4	342	2.55	18%
4	15	13	2	335	2.72	12%
5	383	255	128	945	2.31	25,56%

6	1146	762	256	3457	2.42	22,14%
7	16	14	2	945	2.96	4,79%
8	32	28	4	1800	2.90	7%
9	30	26	4	735	2.63	15,3%
10	60	52	8	2558	2.59	16,66%
11	120	104	16	4228	2.62	16%
12	30	26	4	792	2.63	15%
13	60	52	8	2565	2.59	16,66%
14	191	127	64	480	2,37	23,63%
15	46	38	8	354	2.49	19,89%
16	16	14	2	940	2.96	4,79%
17	191	127	64	216	40.25	1,28%
18	46	38	8	157	40.31	1,12%
19	27	23	4	137	40.50	0,6%
20	46	38	8	72	26.62	8,55%
21	15	13	4	53	28.79	1,13%
22	16	14	2	131	40.52	0,6%
23	32	28	4	223	40.22	1,31%
24	64	56	16	637	39.08	4,16%

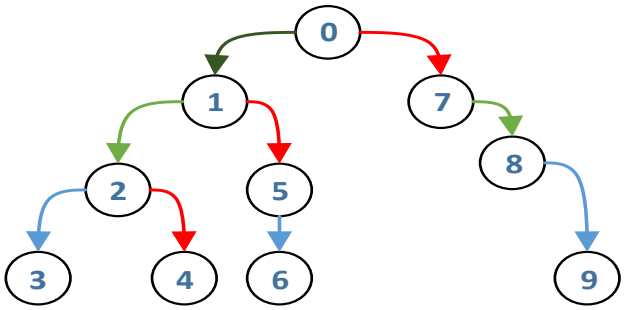


Figure 66. All nodes in a sample schedules tree

Figure 66 is the result of a sample schedule of every schedule and event controller node (e.g., valid and invalid). For example, red lines are related to schedule event cycles and node controllers that the scheduler requires (used only in total schedules). The blue line is related to the controller finishing each task (total and effective schedules). The green is related to real schedules, used in end system platforms (in total, effective, and real schedules). Figure 67 shows the final real schedules after removing every control node in an ***SBMeS*** system.

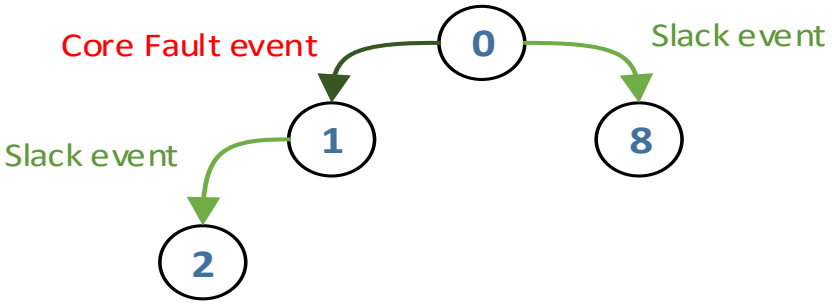


Figure 67. Real nodes in a sample schedules tree

Figure 68 shows schedule SM_{101}, generated by ***MeS*** and visualized via ***MeSViz***. Figure 68 is generated for one core fault (e.g., c_0 status=0) and six task slack, and presents all the data needed for debugging and implementation. For example, it shows that, of five cores, only four are used, and one core is disabled due to a fault. It also concerns how the tasks and messages are allocated and dependent on one another, the rate of SDF for each task and message, the scheduled status and energy saving rate compared to SS schedule SM_0, the message path from one hop to another, and the timing of messages and tasks.

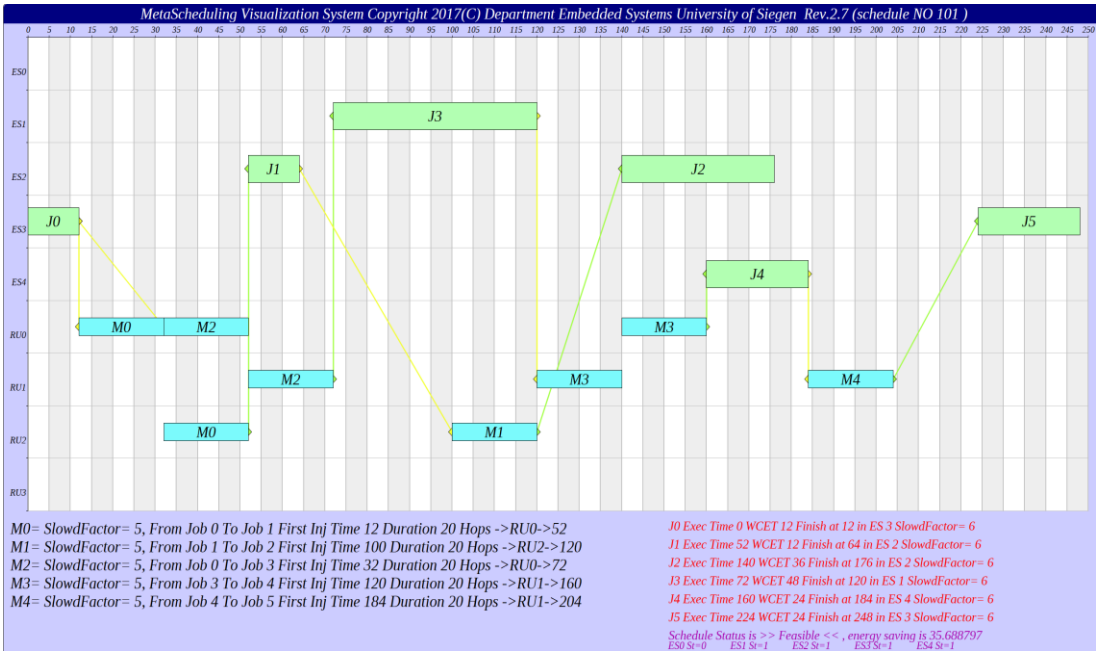


Figure 68. Combination dynamic slack (DS) and core fault results in SM_{101} which generated by meta-scheduling visualization tool (MeSViz)

7.3.5.3. Discussion

Table 26 represents the results of 11 statuses categorized in the two main groups, with (I) and without (II) reliability and flexibility on routing $l9$ and $l8$, and nine scenarios run a total of 24 times. The schedule SM_0 is used for SS, while others use the DS technique.

The results in Figure 69 indicate that, increasing the number of faults and scenario conditions (e.g., combination core faults and slack), the quantity of schedules increases

(e.g., $SSM_{0,6,5,14}$). Figure 70 indicates that, in a core fault scenario (e.g., $SSM_{6,11,13}$), the time for computation and generating schedules is increased, compared to that of DS.

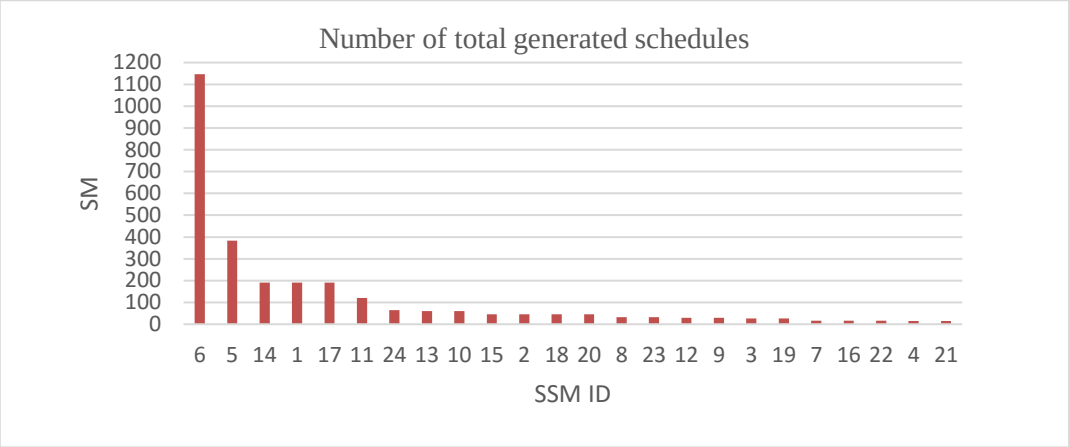


Figure 69. The total number of generated schedules N_{sm} for each scenario

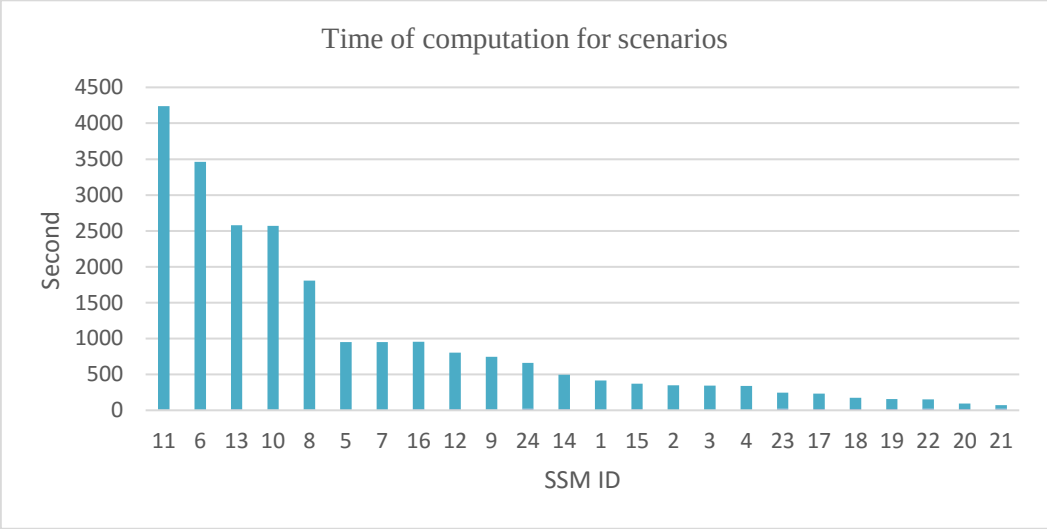


Figure 70.Time of computation for scenarios

Figure 71 indicates that, when disabling *MSDF*, the *FE* of scenarios (e.g., $SSM_{18,19,22}$) is on average 1544.84% higher than for enabled *MSDF* and *TSDF* (e.g., $SSM_{5,6,7,12}$). In addition, compared with disabled *TSDF* scenarios (e.g., $SSM_{21,22}$), with disabled *MSDF*, consumes 30.99% more than *FE*. The *FE* of disabled *TSDF* scenarios (e.g., $SSM_{21,22}$) is, on average, 1066% higher than for enabled *MSDF* and *TSDF* (e.g., $SSM_{5,6,7,12}$).

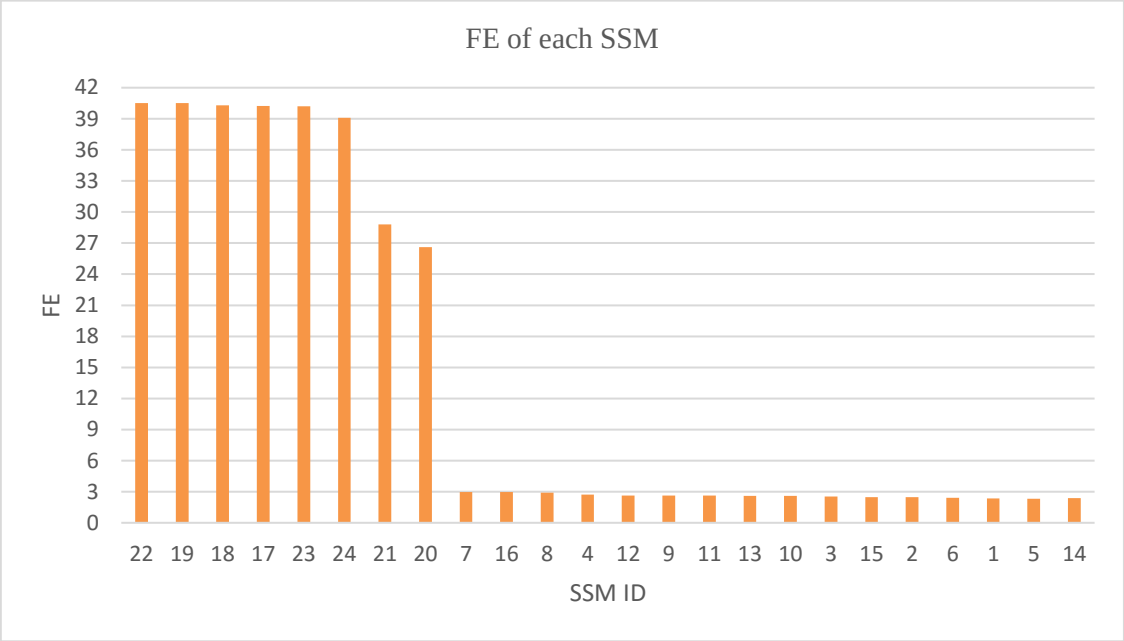


Figure 71. FE results for each scenario SSM_x

Figure 72 illustrates that DS, of all the scenarios, has the strongest effect on energy efficiency. For example, in scenarios $SSM_{1,5,14}$, all tasks have DS, and $SSM_{19,22}$ does not have DS. In addition, increasing rates of failure in the cores (e.g., $SSM_{7,8,16,23,24}$), even without DS, other techniques (e.g., core or router frequency scaling) can save more energy than SS.

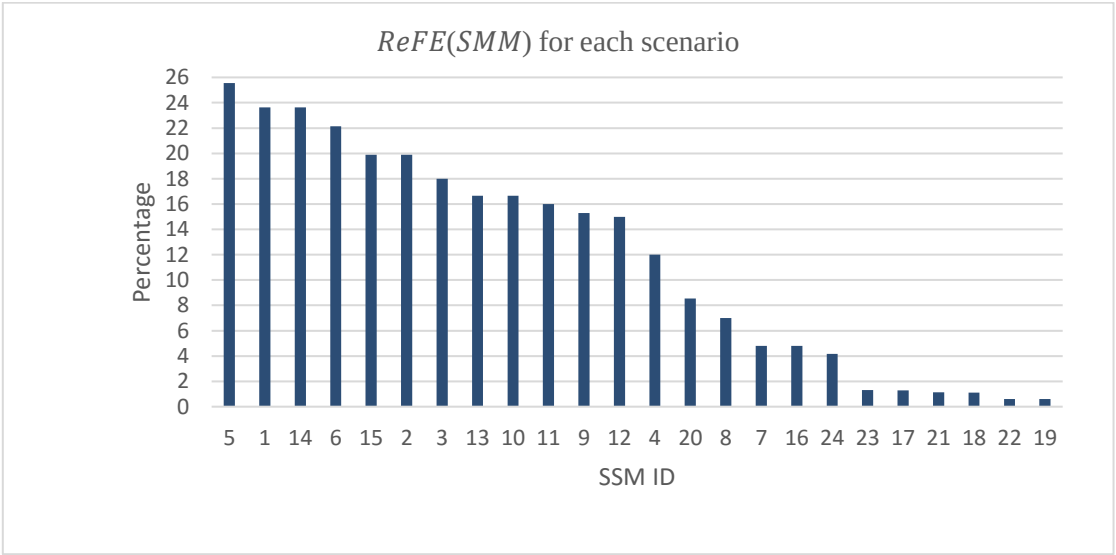


Figure 72. Total $ReFE(SMM)$ (percentage) results for each scenario of comparing dynamic schedules with a static schedule SM_x

The results in Table 27 indicate that disabling or enabling $l8$ and $l9$ has no effect on energy saving or energy consumption, other than a small amount in SSM_1 and SSM_{114} (at a rate of $0.02\ FE$), but it increases computation and generating time – except for a small amount in SSM_7 and SSM_{16}, where it is decreased at a rate of $5s$.

Table 27. Redundancy path routing effect

SSM_x	Links-ID	Slack	$MSDF$	$TSDF$	$flt(c)$	Total N_{sm}	Effective- N_{sm}	Real- N_{sm}	Times (seconds)	$FE_{avg,dyn}$	$ReFE(SSM)$
1	9	6	6	6	0	191	127	64	416	2.35	23,63
14	7	6	6	6	0	191	127	64	480	2,37	23,63
2	9	3	6	6	0	46	38	8	348	2.49	19,89
15	7	3	6	6	0	46	38	8	354	2.49	19,89
10	9	1	6	6	2	60	52	8	2558	2.59	16,66
13	7	1	6	6	2	60	52	8	2565	2.59	16,66
7	9	0	6	6	1	16	14	2	945	2.96	4,79
16	7	0	6	6	1	16	14	2	940	2.96	4,79

These results confirm our theory that the ***SBMeS*** method and ***MeS*** algorithm are able to balance and perform trade-offs between fault-tolerance, reliability, and energy efficiency in adaptive ***TT MPSoCs***. In addition, compared to SS, DS can reduce power consumption in ***MPSOCs*** and has better energy efficiency in robust and adaptive ***TTS*** when using SDF for cores and routers.

7.4. Summary

This section presented five case studies in which developing models and techniques were designed and evaluated. Each case study helped to evaluate the ***SBMeS*** technique, ***MeS***, and the MeSViz tools. These case studies have all been published in journals or included in conferences papers, are currently under review, or are currently awaiting publication.

Chapter 8: Conclusion and further research

This chapter concludes the thesis and highlights problems and recommendations for future research.

We have proposed a new energy-efficient ***SBMeS*** algorithm for an ***NoC***-based ***MPSoC***. The domain of embedded systems based on ***MPSoCs*** and ***NoCs*** has many challenges due to the emergence of new applications, such as drones, e-cars, and smart-cars. Scheduling plays a vital role in real-time systems and ***TTS***, through management and control of different platform and hardware services in a perfectly timed system.

This dissertation introduced a ***SBMeS*** model for safety and mixed-critical systems of ***MPSoCs*** and ***NoCs***, based on ***adaptive TT*** that could support different scenarios for fault-tolerance systems. The ***SBMeS*** model was designed first to work and support slacks and for energy efficiency and was then extended to support core faults.

The formulation was based on ***MIQP*** and all scenarios were investigated using ***CIPLEX***. The summary of the results is as follows:

A) ***MeSViz*** can visualize meta-schedules in adaptive ***TT MPSoC*** and ***NoCs***. The tool provides a systematic means of realizing a meta-visualizer. It helps to easily validate schedules to find problems in schedules (e.g., bugs, errors, overlaps, and collisions, and their hierarchy).

B) In section 7.3.1, our DTS algorithm was seen to save 86% of memory for 512 schedules. This method is used for sections 7.3.2.6, 7.3.3.67.3.4.4)

C) In section 7.3.2, our algorithm minimizes the frequency of cores by maximizing the SDF of tasks $TSDF$ in different scenarios. Thereby, it reduces the dynamic power consumption of the ***NoC*** framework. ***MeS*** can be used in scenario-based (fault, safety, power-saving) scheduling and adaptivity ***TTS***. ***MeSViz*** is used to evaluate ***MeS*** results, and there is evidence of accuracy and performance for ***MeS*** algorithms. The simulation results show that our DS algorithm, compared to the SS, produces, on average, a maximum of 64.4% in a single schedule and 41.61% energy reduction for ***NoCs***.

D) In section 7.3.3, our algorithm minimizes the frequency of cores and routers by maximizing the SDF of execution times for each task and the DM for each message in the whole path in different scenarios. As a result, it reduces the dynamic power consumption of the ***NoC*** framework for both routers and cores. The novel ***SBMeS*** technique is expandable and reduces the dynamic power consumption in the scenario-based schedules. The simulation results show that our DS and SDF method, compared to the SS, produces an energy reduction of up to 59.57% for cores ($ReFE_c$) and 31.12% for routers ($ReFE_R$). The energy reduction is, on average of ***SSM***, up to 32.92% in a single schedule and 25.69% for ***NoCs***.

E) As seen in section 7.3.4, the novel optimization technique of our ***MeS*** tool is independent of network design, is expandable, and can be used to reduce and optimize dynamic power consumption in the schedules. ***MeS*** is used for our multi-scenario-

based (e.g., fault, safety, power-saving) scheduling on adaptive ***TTS***. The results show that our DS time consideration and frequency slowdown in ***MeS***, compared to the SS time, produces an energy reduction of up to 57% for cores ($ReFE_c$) and 52.46% ($ReFE_R$) for routers. The energy reduction is, on average, up to 46.73% in a single schedule and 33.88% for ***NoCs***.

F) In section 7.3.5, compared to section 7.3.4, the ***SBMeS*** technique and ***MeS*** tool was improved for network design and expandability, revealing no significant impact on power consumption in the schedules when disabling redundancy links. ***MeS*** is used in our multi-scenario-based (e.g., fault, safety, power-saving) scheduling and adaptive ***TT*** systems problems. The results show that our DS and SDF method and algorithm based on ***MeS***, compared to the SS, produce an energy reduction of up to 25.56%, on average. The energy reduction in the worst-case is 0.6%.

Glossary

S

T

V

W

X

List of Figures

List of Tables

List of examples

List of Algorithms

PUBLICATIONS AND PREVIOUS WORKS

R. Obermaisser. H. Ahmadian, A. Maleki, Y. Bebawy, A. Lenz, A, B. Sorkhpour " Adaptive Time-Triggered Multi-Core Architecture," *Designs*, vol. 3, no. 1, p. 7, 2019.

B. Sorkhpour, R. Obermaisser, and Y. Bebawy, Eds., *Optimization of Frequency-Scaling in Time-Triggered Multi-Core Architectures using Scenario-Based Meta-Scheduling*: "in Proceedings of *AmE 2019-Automotive meets Electronics; 10th GMM-Symposium* VDE, 2019.

B. Sorkhpour and R. Obermaisser, "MeSViz: Visualizing Scenario-based Meta-Schedules for Adaptive Time-Triggered Systems," in *AmE 2018-Automotive meets Electronics; 9th GMM-Symposium*, 2018, pp. 1–6.

B. Sorkhpour, A. Murshed, and R. Obermaisser, "Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems," in *Knowledge-Based Engineering and Innovation (KBEI), 2017 IEEE 4th International Conference on*, 2017, pp. 143–150.

I. REFERENCES

[1] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*: Springer Science & Business Media, 2011.

[2] J. Theis, G. Fohler, and S. Baruah, "Schedule table generation for time-triggered mixed criticality systems," *Proc. WMC, RTSS*, pp. 79–84, 2013.

[3] Safepower, *D3.8 User guide of the heterogeneous MPSoC design.* [Online] Available: http://safepower-project.eu/wp-content/uploads/2019/01/D3.8-User_guide_of_the_heterogeneous_MPSoC_design_v1-0_final.pdf.

[4] DW BUSINESS, *BMW increases R&D spending on e-cars, autonomous vehicles*.

[5] S. R. Sakhare and M. S. Ali, "Genetic algorithm based adaptive scheduling algorithm for real time operating systems," *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 3, pp. 91–97, 2012.

[6] R. Obermaisser, Ed., *Time-triggered communication*. Boca Raton, FL: CRC Press, 2012.

[7] P. Munk, "Visualization of scheduling in real-time embedded systems," University of Stuttgart, Institute of Software Technology, Department of Programming Languages and Compilers, 20103.

[8] S. Hunold, R. Hoffmann, and F. Suter, "Jedule: A Tool for Visualizing Schedules of Parallel Applications," in *2010 International Conference on Parallel Processing Workshops (ICPPW)*, San Diego, CA, USA, pp. 169–178.

[9] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *36th International symposium on microarchitecture*, San Diego, CA, USA, 2003, pp. 105–116.

[10] B. Sorkhpour and R. Obermaisser, "MeSViz: Visualizing Scenario-based Meta-Schedules for Adaptive Time-Triggered Systems," in *AmE 2018-Automotive meets Electronics; 9th GMM-Symposium*, 2018, pp. 1–6.

[11] A. C. Persya and T. R. G. Nair, "Model based design of super schedulers managing catastrophic scenario in hard real time systems," in *2013 International Conference on Information Communication and Embedded Systems (ICICES)*, Chennai, 2013, pp. 1149–1155.

[12] B. Sorkhpour, O. Roman, and Y. Bebawy, Eds., *Optimization of Frequency-Scaling in Time-Triggered Multi-Core Architectures using Scenario-Based Meta-Scheduling*: VDE, 2019.

[13] B. Sorkhpour, A. Murshed, and R. Obermaisser, "Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems," in *Knowledge-Based*

Engineering and Innovation (KBEI), 2017 IEEE 4th International Conference on, 2017, pp. 143–150.

[14] J. Huang, C. Buckl, A. Raabe, and A. Knoll, "Energy-Aware Task Allocation for Network-on-Chip Based Heterogeneous Multiprocessor Systems," in *2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*, Ayia Napa, Cyprus, 2011, pp. 447–454.

[15] S. Prabhu, B. Grot, P. Gratz, and J. Hu, "Ocin tsim-DVFS aware simulator for NoCs," *Proc. SAW*, vol. 1, 2010.

[16] Roman Obermaisser *et al.*, "Adaptive Time-Triggered Multi-Core Architecture," *Designs*, vol. 3, no. 1, p. 7, 2019.

[17] H. Kopetz, Ed., *Real-Time Systems: Design Principles for Distributed Embedded Applications (Real-Time Systems Series) // Real-time systems: Design principles for distributed embedded applications*, 2nd ed. New York: Springer, 2011.

[18] F. Guan, L. Peng, L. Perneel, H. Fayyad-Kazan, and M. Timmerman, "A Design That Incorporates Adaptive Reservation into Mixed-Criticality Systems," *Scientific Programming*, vol. 2017, 2017.

[19] Y. Lin, Y.-l. Zhou, S.-t. Fan, and Y.-m. Jia, "Analysis on Time Triggered Flexible Scheduling with Safety-Critical System," in *Chinese Intelligent Systems Conference*, 2017, pp. 495–504.

[20] IEEE, "TTP - A Time-Triggered Protocol For Fault-tolerant Real-time System - Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposi,"

[21] J. Cortadella, A. Kondratyev, L. Lavagno, C. Passerone, and Y. Watanabe, "Quasi-static scheduling of independent tasks for reactive systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 10, pp. 1492–1514, 2005.

[22] R. Rajaei, S. Hessabi, and B. V. Vahdat, "An energy-aware methodology for mapping and scheduling of concurrent applications in MPSOC architectures," in *Electrical Engineering (ICEE), 2011 19th Iranian Conference on*, 2011, pp. 1–6.

[23] A. Menna, "Allocation, Assignment and Scheduling for Multi-processor System on Chip," PhD, Universit ´e des Sciences et Technologies de Lille, 2006.

[24] A. Murshed, "Scheduling Event-Triggered and Time-Triggered Applications with Optimal Reliability and Predictability on Networked Multi-Core Chips,"

[25] A. Maleki, H. Ahmadian, and R. Obermaisser, "Fault-Tolerant and Energy-Efficient Communication in Mixed-Criticality Networks-on-Chips," in *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2018, pp. 1–7.

[26] P. Eitschberger, S. Holmbacka, and J. Keller, “Trade-Off Between Performance, Fault Tolerance and Energy Consumption in Duplication-Based Taskgraph Scheduling,” in *Architecture of Computing Systems – ARCS 2018*.

[27] R. Lent, “Grid Scheduling with Makespan and Energy-Based Goals,” *Journal of Grid Computing*, vol. 13, no. 4, pp. 527–546, 2015.

[28] P. Eitschberger, “Energy-efficient and Fault-tolerant Scheduling for Manycores and Grids,” Fakultät für Mathematik und Informatik, FernUniversität in Hagen, Hagen, 2017.

[29] A. Murshed, “Scheduling event-triggered and time-triggered applications with optimal reliability and predictability on networked multi-core chips,” Dissertation, Embedded Systems, Universität Siegen, Siegen, 2018.

[30] E. Dubrova, *Fault-Tolerant Design*. New York, NY: Springer; Imprint, 2013.

[31] A. Avizienis, J.-C. Laprie, B. Randell, and others, *Fundamental concepts of dependability*: University of Newcastle upon Tyne, Computing Science, 2001.

[32] I. Bate, A. Burns, and R. I. Davis, “A Bailout Protocol for Mixed Criticality Systems,” in *2015 27th Euromicro Conference on Real-Time Systems: ECRTS 2015 : proceedings*, Lund, Sweden, 2015, pp. 259–268.

[33] A. Burns and R. Davis, “Mixed criticality systems-a review,” *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.

[34] B. Hu *et al.*, “FFOB: efficient online mode-switch procrastination in mixed-criticality systems,” *Real-Time Syst*, vol. 79, no. 1, p. 39, 2018.

[35] H. Isakovic and R. Grosu, “A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform,” in *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications*: IGI Global, 2018, pp. 1153–1178.

[36] B. Sorkhpour and R. Obermaisser, “MeSViz: Visualizing Scenario-based Meta-Schedules for Adaptive Time-Triggered Systems,” in *AmE 2018-Automotive meets Electronics; 9th GMM-Symposium*, 2018, pp. 1–6.

[37] B. Hu, “Schedulability Analysis of General Task Model and Demand Aware Scheduling in Mixed-Criticality Systems,” Technische Universität München.

[38] H. Ahmadian, F. Nekouei, and R. Obermaisser, “Fault recovery and adaptation in time-triggered Networks-on-Chips for mixed-criticality systems,” in *12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, (ReCoSoC 2017): July 12-14, 2017, Madrid, Spain : proceedings*, Madrid, Spain, 2017, pp. 1–8.

[39] R. Trüb, G. Giannopoulou, A. Tretter, and L. Thiele, “Implementation of partitioned mixed-criticality scheduling on a multi-core platform,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 122, 2017.

[40] C. Schöler, “Novel scheduling strategies for future NoC and MPSoC architectures,” 2017.

[41] M. I. Huse, “FlexRay Analysis, Configuration Parameter Estimation, and Adversaries,” NTNU.

[42] W. Steiner, “Synthesis of Static Communication Schedules for Mixed-Criticality Systems,” in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing workshops (ISORCW): 28-31 March 2011, Newport Beach, California, USA ; proceedings*, Newport Beach, CA, USA, 2011, pp. 11–18.

[43] G. Marchetto, S. Tahir, and M. Grosso, “A blocking probability study for the aethereal network-on-chip,” in *Proceedings of 2016 11th International Design & Test Symposium (IDT): December 18-20-2016, Hammamet, Tunisia*, Hammamet, Tunisia, 2016, pp. 104–109.

[44] M. Ruff, “Evolution of local interconnect network (LIN) solutions,” in *VTC2003-Fall Orlando: 2003 IEEE 58th Vehicular Technology Conference : proceedings : 6-9 October, 2003, Orlando, Florida, USA*, Orlando, FL, USA, 2004, 3382-3389 Vol.5.

[45] R. B. Atitallah, S. Niar, A. Greiner, S. Meftali, and J. L. Dekeyser, “Estimating Energy Consumption for an MPSoC Architectural Exploration,” in *Lecture Notes in Computer Science, Architecture of Computing Systems - ARCS 2006*, W. Grass, B. Sick, and K. Waldschmidt, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 298–310.

[46] U. U. Tariq, H. Wu, and S. Abd Ishak, “Energy-Aware Scheduling of Conditional Task Graphs on NoC-Based MPSoCs,” in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[47] *Compiler-Directed Frequency and Voltage Scaling for a Multiple Clock Domain*: ACM Press.

[48] A. B. Mehta, “Clock Domain Crossing (CDC) Verification,” in *ASIC/SoC Functional Design Verification*.

[49] *Mecanismo de controle de QoS através de DFS em MPSOCS*: Pontifícia Universidade Católica do Rio Grande do Sul; Porto Alegre, 2014.

[50] Marc Boyer, Benoît Dupont de Dinechin, Amaury Graillat, and Lionel Havet, “Computing Routes and Delay Bounds for the Network-on-Chip of the Kalray MPPA2 Processor,”

[51] B. D. de Dinechin and A. Graillat, “Feed-Forward Routing for the Wormhole Switching Network-on-Chip of the Kalray MPPA2 Processor,” in *Proceedings of the 10th International Workshop on Network on Chip Architectures - NoCArc'17*, Cambridge, MA, USA, 2017, pp. 1–6.

[52] KALRAY Corporation, *Kalray’s MPPA network-on-chip.* [Online] Available: http://www.kalrayinc.com/portfolio/processors/.

[53] I. Lee, J. Y. T. Leung, and S. H. Son, *Handbook of real-time and embedded systems*: CRC Press, 2007.

[54] Wikipedia, *XtratuM - Wikipedia.* [Online] Available: https://en.wikipedia.org/w/index.php?oldid=877711274. Accessed on: Feb. 11 2019.

[55] I. Ripoll *et al.*, “Configuration and Scheduling tools for TSP systems based on XtratuM,” *Data Systems In Aerospace (DASIA 2010)*, 2010.

[56] V. Brocal *et al.*, “Xoncrete: a scheduling tool for partitioned real-time systems,” *Embedded Real-Time Software and Systems*, 2010.

[57] R. Jejurikar and R. Gupta, “Dynamic slack reclamation with procrastination scheduling in real-time embedded systems,” in *Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 111–116.

[58] H. Li, S. Bhunia, Y. Chen, T. N. Vijaykumar, and K. Roy, “Deterministic clock gating for microprocessor power reduction,” in *The 9th international symposium on high-performance computer architecture*, Anaheim, CA, USA, 2003, pp. 113–122.

[59] H. Matsutani, M. Koibuchi, H. Nakamura, and H. Amano, “Run-Time Power-Gating Techniques for Low-Power On-Chip Networks,” in *Low Power Networks-on-Chip*.

[60] P. W. Cook *et al.*, “Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors,” *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[61] W. Kim, J. Kim, and S. L. Min, “Dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis,” in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, 2002, pp. 788–794.

[62] D. M. Brooks *et al.*, “Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors,” *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[63] S. Prabhu, *Ocin_tsim - A DVFS Aware Simulator for NoC Design Space Exploration and Optimization*. [College Station, Tex.]: [Texas A & M University], 2010.

[64] A. Bianco, P. Giaccone, and N. Li, “Exploiting Dynamic Voltage and Frequency Scaling in networks on chip,” in *IEEE 13th International Conference on High*

Performance Switching and Routing (HPSR), 2012, Belgrade, Serbia, 2012, pp. 229–234.

[65] M. Caria, F. Carpio, A. Jukan, and M. Hoffmann, “Migration to energy efficient routers: Where to start?,” in *IEEE International Conference on Communications (ICC), 2014: 10-14 June 2014, Sydney, Australia*, Sydney, NSW, 2014, pp. 4300–4306.

[66] D. M. Brooks *et al.*, “Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors,” *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[67] S. Chai, Y. Li, J. Wang, and C. Wu, “An energy-efficient scheduling algorithm for computation-intensive tasks on NoC-based MPSoCs,” *Journal of Computational Information Systems*, vol. 9, no. 5, pp. 1817–1826, 2013.

[68] P. K. Sharma, S. Biswas, and P. Mitra, “Energy efficient heuristic application mapping for 2-D mesh-based network-on-chip,” *Microprocessors and Microsystems*, vol. 64, pp. 88–100, 2019.

[69] H. M. Kamali, K. Z. Azar, and S. Hessabi, “DuCNoC: A High-Throughput FPGA-Based NoC Simulator Using Dual-Clock Lightweight Router Micro-Architecture,” *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 208–221, 2018.

[70] H. Farrokhbakht, H. M. Kamali, and S. Hessabi, “SMART,” in *Proceedings of the Eleventh IEEE/ACM International Symposium on Networks-on-Chip - NOCS '17*, Seoul, Republic of Korea, 2017, pp. 1–8.

[71] W. Hu, X. Tang, B. Xie, T. Chen, and D. Wang, “An Efficient Power-Aware Optimization for Task Scheduling on NoC-based Many-core System,” in *2010 10th IEEE International Conference on Computer and Information Technology*, Bradford, United Kingdom, 2010, pp. 171–178.

[72] H. F. Sheikh and I. Ahmad, “Simultaneous optimization of performance, energy and temperature for DAG scheduling in multi-core processors,” in *Green Computing Conference (IGCC), 2012 International*, 2012, pp. 1–6.

[73] J. Hu and R. Marculescu, “Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints,” in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2004, pp. 234–239.

[74] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, “darkNoC,” in *Proceedings of the 51st Annual Design Automation Conference*, San Francisco, CA, USA, 2014, pp. 1–6.

[75] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, “Dynamic and aggressive scheduling techniques for power-aware real-time systems,” in *22nd IEEE real-time systems symposium: (RTSS 2001)*, London, UK, 2001, pp. 95–105.

[76] R. Jejurikar and R. Gupta, “Dynamic slack reclamation with procrastination scheduling in real-time embedded systems,” in *DAC 42*, San Diego, California, USA, 2005, p. 111.

[77] G. Ma, L. Gu, and N. Li, “Scenario-Based Proactive Robust Optimization for Critical-Chain Project Scheduling,” *J. Constr. Eng. Manage.*, vol. 141, no. 10, p. 4015030, 2015.

[78] H. K. Mondal and S. Deb, “Power-and performance-aware on-chip interconnection architectures for many-core systems,” IIIT-Delhi.

[79] J. Wang *et al.*, “Designing Voltage-Frequency Island Aware Power-Efficient NoC through Slack Optimization,” in *International Conference on Information Science and Applications (ICISA), 2014: 6-9 May 2014, Seoul, South Korea*, Seoul, South Korea, 2014, pp. 1–4.

[80] K. Han, J.-J. Lee, J. Lee, W. Lee, and M. Pedram, “TEI-NoC: Optimizing Ultralow Power NoCs Exploiting the Temperature Effect Inversion,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 2, pp. 458–471, 2018.

[81] D. Li and J. Wu, “Energy-efficient contention-aware application mapping and scheduling on NoC-based MPSoCs,” *Journal of Parallel and Distributed Computing*, vol. 96, pp. 1–11, 2016.

[82] W. Y. Lee, Y. W. Ko, H. Lee, and H. Kim, “Energy-efficient scheduling of a real-time task on dvfs-enabled multi-cores,” in *Proceedings of the 2009 International Conference on Hybrid Information Technology*, 2009, pp. 273–277.

[83] B. Sprunt, L. Sha, and J. Lehoczky, “Aperiodic task scheduling for Hard-Real-Time systems,” *Real-Time Syst*, vol. 1, no. 1, pp. 27–60, 1989.

[84] J. K. Strosnider, J. P. Lehoczky, and L. Sha, “The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments,” *IEEE Trans. Comput.*, vol. 44, no. 1, pp. 73–91, 1995.

[85] N. Chatterjee, S. Paul, and S. Chattopadhyay, “Task mapping and scheduling for network-on-chip based multi-core platform with transient faults,” *Journal of Systems Architecture*, vol. 83, pp. 34–56, 2018.

[86] R. N. Mahapatra and W. Zhao, “An energy-efficient slack distribution technique for multimode distributed real-time embedded systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 7, pp. 650–662, 2005.

[87] G. Avni, S. Guha, and G. Rodriguez-Navas, “Synthesizing time-triggered schedules for switched networks with faulty links,” in *Proceedings of the 13th International Conference on Embedded Software*, Pittsburgh, Pennsylvania, 2016, pp. 1–10.

[88] F. Benhamou, *Principle and Practice of Constraint Programming - CP 2006: 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*. Berlin Heidelberg: Springer-Verlag, 2006.

[89] *Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems*, 2011.

[90] A. Murshed, R. Obermaisser, H. Ahmadian, and A. Khalifeh, “Scheduling and allocation of time-triggered and event-triggered services for multi-core processors with networks-on-a-chip,” pp. 1424–1431.

[91] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, “Variation-aware task allocation and scheduling for MPSoC,” in *IEEE/ACM International Conference on Computer-Aided Design, 2007*, San Jose, CA, USA, 2007, pp. 598–603.

[92] D. Mirzoyan, B. Akesson, and K. Goossens, “Process-variation-aware mapping of best-effort and real-time streaming applications to MPSoCs,” *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2s, pp. 1–24, 2014.

[93] C. MacLean and G. COWIE, *Data flow graph*: Google Patents.

[94] S. K. Baruah, A. Burns, and R. I. Davis, “Response-Time Analysis for Mixed Criticality Systems,” in *IEEE 32nd Real-Time Systems Symposium (RTSS), 2011*, Vienna, Austria, 2011, pp. 34–43.

[95] A. Burns and S. Baruah, “Timing Faults and Mixed Criticality Systems,” in *Lecture notes in computer science, 0302-9743*, 6875. Festschrift, *Dependable and historic computing: Essays dedicated to Brian Randell on the occasion of his 75th birthday / Cliff B. Jones, John L. Lloyd (eds.)*, B. Randell, C. B. Jones, and J. L. Lloyd, Eds., Heidelberg: Springer, 2011, pp. 147–166.

[96] P. Ekberg and W. Yi, “Outstanding Paper Award: Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks,” in *Proceedings of The 24th Euromicro Conference on Real-Time Systems: 10-13 July 2012, Pisa, Italy*, Pisa, Italy, 2012, pp. 135–144.

[97] M. R. Garey, D. S. Johnson, and L. Stockmeyer, “Some simplified NP-complete problems,” in *Proceedings of the sixth annual ACM symposium on Theory of computing - STOC '74*, Seattle, Washington, United States, 1974, pp. 47–63.

[98] L. Su *et al.*, “Synthesizing Fault-Tolerant Schedule for Time-Triggered Network Without Hot Backup,” *IEEE Trans. Ind. Electron.*, vol. 66, no. 2, pp. 1345–1355, 2019.

[99] A. Carvalho Junior, M. Bruschi, C. Santana, and J. Santana, “Green Cloud Meta-Scheduling : A Flexible and Automatic Approach,” (eng), *Journal of Grid Computing : From Grids to Cloud Federations*, vol. 14, no. 1, pp. 109–126, http://dx.doi.org/10.1007/s10723-015-9333-z, 2016.

[100] T. Tiendrebeogo, “Prospect of Reduction of the GreenHouse Gas Emission by ICT in Africa,” in *e-Infrastructure and e-Services*.

[101] Deutsche Welle (www.dw.com), *Carmaker BMW to invest heavily in battery cell center | DW | 24.11.2017*. [Online] Available: https://p.dw.com/p/2oD3x. Accessed on: Dec. 03 2018.

[102] G. Fohler, “Changing operational modes in the context of pre run-time scheduling,” *IEICE transactions on information and systems*, vol. 76, no. 11, pp. 1333–1340, 1993.

[103] H. Jung, H. Oh, and S. Ha, “Multiprocessor scheduling of a multi-mode dataflow graph considering mode transition delay,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, p. 37, 2017.

[104] A. Das, A. Kumar, and B. Veeravalli, “Energy-Aware Communication and Remapping of Tasks for Reliable Multimedia Multiprocessor Systems,” in *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 2012*, Singapore, Singapore, 2012, pp. 564–571.

[105] S. A. Ishak, H. Wu, and U. U. Tariq, “Energy-Aware Task Scheduling on Heterogeneous NoC-Based MPSoCs,” in *IEEE 35th IEEE International Conference on Computer Design: ICCD 2017 : 5-8 November 2017 Boston, MA, USA : proceedings*, Boston, MA, 2017, pp. 165–168.

[106] C. A. Floudas and V. Visweswaran, “Quadratic Optimization,” in *Nonconvex Optimization and Its Applications*, vol. 2, *Handbook of Global Optimization*, R. Horst and P. M. Pardalos, Eds., Boston, MA, s.l.: Springer US, 1995, pp. 217–269.

[107] R. Lazimy, “Mixed-integer quadratic programming,” *Mathematical Programming*, vol. 22, no. 1, pp. 332–349, 1982.

[108] A. Majd, G. Sahebi, M. Daneshtalab, and E. Troubitsyna, “Optimizing scheduling for heterogeneous computing systems using combinatorial meta-heuristic solution,” in *2017 IEEE SmartWorld: Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI) : 2017 conference proceedings : San Francisco Bay Area, California, USA, August 4-8, 2017*, San Francisco, CA, 2017, pp. 1–8.

[109] B. Xing and W.-J. Gao, “Imperialist Competitive Algorithm,” in *Intelligent Systems Reference Library, Innovative computational intelligence: A rough guide to 134 clever algorithms*, B. Xing and W.-J. Gao, Eds., New York NY: Springer Berlin Heidelberg, 2013, pp. 203–209.

[110] J. D. Foster, A. M. Berry, N. Boland, and H. Waterer, “Comparison of Mixed-Integer Programming and Genetic Algorithm Methods for Distributed Generation Planning,” *IEEE Trans. Power Syst.*, vol. 29, no. 2, pp. 833–843, 2014.

[111] J. Yin, P. Zhou, A. Holey, S. S. Sapatnekar, and A. Zhai, “Energy-efficient non-minimal path on-chip interconnection network for heterogeneous systems,” in *ISPLED'12: Proceedings of the international symposium on low power electronics and design*, Redondo Beach, California, USA, 2012, p. 57.

[112] J. Falk *et al.*, “Quasi-static scheduling of data flow graphs in the presence of limited channel capacities,” in *The 13th IEEE Symposium on Embedded Systems for Real-time Multimedia: October 8-9, 2015, Amsterdam, Netherlands*, Amsterdam, Netherlands, 2015, pp. 1–10.

[113] T. Wei, P. Mishra, K. Wu, and J. Zhou, “Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems,” *Journal of Systems and Software*, vol. 85, no. 6, pp. 1386–1399, 2012.

[114] M. J. Ryan, “A Case Study on the Impact of Convergence on Physical Architectures—The Tactical Communications System,”

[115] Y. Huang and D. P. Palomar, “Randomized Algorithms for Optimal Solutions of Double-Sided QCQP With Applications in Signal Processing,” *IEEE Trans. Signal Process.*, vol. 62, no. 5, pp. 1093–1108, 2014.

[116] X. Cai, W. Hu, T. Ma, and R. Ma, “A hybrid scheduling algorithm for reconfigurable processor architecture,” in *Proceedings of the 13th IEEE Conference on Industrial Electronics and Applications (ICIEA 2018): 31 May-2 June 2018 Wuhan, China*, Wuhan, 2018, pp. 745–749.

[117] P.-A. Hsiung and J.-S. Shen, *Dynamic reconfigurable network-on-chip design: Innovations for computational processing and communication*. Hershey, Pa.: IGI Global, 2010.

[118] R. Misener and C. A. Floudas, “Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations,” *Mathematical Programming*, vol. 136, no. 1, pp. 155–182, 2012.

[119] D. Axehill, “Applications of integer quadratic programming in control and communication,” Institutionen för systemteknik, 2005.

[120] A. Nemirovskii, “Several NP-hard problems arising in robust stability analysis,” *Math. Control Signal Systems*, vol. 6, no. 2, pp. 99–105, 1993.

[121] A. Sarwar, “Cmos power consumption and cpd calculation,” *Proceeding: Design Considerations for Logic Products*, 1997.

[122] S. Kaxiras and M. Martonosi, “Computer Architecture Techniques for Power-Efficiency,” *Synthesis Lectures on Computer Architecture*, vol. 3, no. 1, pp. 1–207, 2008.

[123] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, “Recent Advances in Quadratic Programming Algorithms for Nonlinear Model Predictive Control,” *Vietnam Journal of Mathematics*, vol. 46, no. 4, pp. 863–882, 2018.

[124] R. Fourer, “Strategies for “Not Linear” Optimization,” Houston, TX, Mar. 6 2014.

[125] L. A. Cortes, P. Eles, and Z. Peng, “Quasi-Static Scheduling for Multiprocessor Real-Time Systems with Hard and Soft Tasks,” in *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications: 17-19 August 2005, Hong Kong, China : proceedings*, Hong Kong, China, 2005, pp. 422–428.

[126] L. Benini, “Platform and MPSoC Design,”

[127] R. Obermaisser and P. Peti, “A Fault Hypothesis for Integrated Architectures,” in *Proceedings of the Fourth Workshop on Intelligent Solutions in Embedded Systems: Vienna University of Technology, Vienna, Austria, 2006 June 30*, Vienna, Austria, 2005, pp. 1–18.

[128] R. Obermaisser *et al.*, “Adaptive Time-Triggered Multi-Core Architecture,” *Designs*, vol. 3, no. 1, p. 7, https://www.mdpi.com/2411-9660/3/1/7/pdf, 2019.

[129] IBM, *IBM ILOG CPLEX Optimization Studio CPLEX User’s Manual*: IBM, 1987-2016.

[130] *Chart Component and Control Library for .NET (C#/VB), Java, C++, ASP, COM, PHP, Perl, Python, Ruby, ColdFusion.* [Online] Available: https://www.advsofteng.com/product.html. Accessed on: Jan. 10 2019.

[131] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, “Graphviz—open source graph drawing tools,” in *International Symposium on Graph Drawing*, 2001, pp. 483–484.

[132] T. Lei and S. Kumar, “Algorithms and tools for network on chip based system design,” in *Chip in sampa*, Sao Paulo, Brazil, 2003, pp. 163–168.

[133] G. D. Micheli and L. Benini, “Powering networks on chips: energy-efficient and reliable interconnect design for SoCs,” in *isss*, 2001, pp. 33–38.