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.


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

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

ABSTRACT ................................................................................................................................. 7

KURZBESCHREIBUNG ............................................................................................................. 8

CONTENTS ............................................................................................................................... 10

CHAPTER 1: INTRODUCTION ................................................................................................. 15
  1.1. MOTIVATION .................................................................................................................. 16
  1.2. RESEARCH SCOPE ...................................................................................................... 19
  1.3. STRUCTURE OF THE THESIS .................................................................................. 20

CHAPTER 2: RELATED WORK AND BASIC CONCEPTS ......................................................... 22
  2.1. REAL-TIME SYSTEMS ................................................................................................. 22
    2.1.1. Embedded real-time systems ............................................................................... 22
      2.1.1.1. Makespan ....................................................................................................... 23
    2.1.2. Dependability ...................................................................................................... 23
      2.1.2.1. Mixed-criticality systems ............................................................................ 23
      2.1.2.2. Adaptivity .................................................................................................... 24
    2.1.3. Time ..................................................................................................................... 24
    2.2. TIME-TRIGGERED SYSTEMS (TTS) ......................................................................... 24
    2.3. GLOBAL TIME BASE ............................................................................................... 24
      2.3.1. Time-triggered networks (TTNs) ...................................................................... 25
      2.3.1.1. Messages ....................................................................................................... 25
      2.3.2. Time-triggered multi-processor system-on-a-chip (MPSoC) and network-on-a-chip (NoC) ........................................................................................................... 25
      2.3.3. Time-triggered (TT) execution environments .................................................... 27
        2.3.3.1. Tasks ........................................................................................................... 27
        2.3.3.2. Worst-case execution time (WCET) ............................................................. 27
        2.3.3.3. Slack .......................................................................................................... 28
    2.4. ENERGY MANAGEMENT TECHNIQUES .................................................................. 28
      2.4.1. Clock gating and power gating ......................................................................... 28
        2.4.1.1. Dynamic voltage and frequency scaling (DVFS) ........................................... 28
        2.4.1.1.1. Frequency tuning on network-on-chip (NoC) ........................................... 28
        2.4.1.1.2. Distributed dynamic voltage and frequency scaling (DVFS) algorithm at the router and core level ................................................................. 29
        2.4.1.1.3. Worst case execution times (WCETs) and distributed dynamic voltage and frequency scaling (DVFS) ........................................................................ 29
      2.5. ENERGY MANAGEMENT FOR DIFFERENT TYPES OF RESOURCES ................. 30
        2.5.1. Network-on-chip (NoC) ................................................................................. 30
        2.5.2. Processor .......................................................................................................... 31
        2.5.3. Task procrastination and slack reclamation ....................................................... 31
    2.6. SCHEDULING FOR TIME-TRIGGERED SYSTEMS (TTS) ......................................... 32
      2.6.1. Algorithms for static ......................................................................................... 32
        2.6.1.1. Static scheduling for energy efficiency ......................................................... 33
        2.6.1.2. Static scheduling for reliability .................................................................. 33
      2.6.2. Meta-scheduling (MeS) and mode changes ......................................................... 34
      2.6.3. Optimization techniques for scheduling ............................................................. 35
        2.6.3.1. Quadratic Optimization .............................................................................. 35
        2.6.3.2. Mixed integer quadratic programming (MIQP) ........................................... 35
        2.6.3.3. Genetic algorithm (GA) and optimization techniques ................................. 35
    2.7. SCENARIO-BASED SCHEDULING IN EVENT-TRIGGERED SYSTEMS ...................... 36
CHAPTER 3: SYSTEM MODEL ................................................................. 42

3.1. OVERVIEW OF THE MODELS .................................................. 42
  3.2. Input models for a meta-scheduler (MeS) ................................. 42
    3.2.1. The physical model (PM) .............................................. 42
    3.2.2. The application model (AM) ....................................... 43
    3.2.3. The context model (CM) ........................................... 44
  3.3. The schedule model (SM) ....................................................... 45
  3.4. MODELLING OF THE OPTIMIZATION PROBLEM ....................... 46
  3.5. CHOICE OF OPTIMIZATION TECHNIQUE .................................. 46
  3.6. DECISION VARIABLES, CONSTANTS, AND CONSTRAINTS .......... 48
    3.7. DEFINITIONS ................................................................. 48
      3.7.1. Collision avoidance constraint .................................. 49
      3.7.2. Connectivity constraints ........................................ 50
        3.7.2.1. Links reliability ............................................. 50
      3.7.3. Task allocation and assignment constraints .................. 50
      3.7.4. Message duration ................................................ 50
      3.7.5. Path and visited cores ........................................ 50
      3.7.6 Task dependency constraints ..................................... 51
      3.7.7. Message deadlines ............................................... 51
    3.8. DECISION VARIABLES ..................................................... 51
      3.8.1. Slowdown factors (SDFs) ......................................... 51
      3.8.2. Hop count ........................................................... 52
    3.9. ENERGY CONSUMPTION .................................................... 53
      3.9.1. Effective Speed .................................................... 54
      3.9.2. Frequency Co-Efficient ....................................... 54
    3.10. QUADRATIZATION OF THE PRODUCT .................................. 54
    3.11. THE OBJECTIVE FUNCTION .............................................. 55
    3.12. SLACK RECOVERY TECHNIQUE ......................................... 58
      3.12.1. Makespan and slack ............................................. 59
      3.12.2. Power consumption and slack .................................. 60

CHAPTER 4: SCENARIO-BASED META-SCHEDULING (SBMeS) .................... 63

4.1. META-SCHEDULER (MeS) ......................................................... 63
4.2. SYSTEM MODEL AND ALGORITHM ........................................... 64
4.3. ARCHITECTURE OF META-SCHEDULER (MeS) ............................ 66
4.4. META-SCHEDULER (MeS) DESIGN ........................................... 68
  4.4.1. Event-driving technique ............................................. 68
  4.4.2. The schedule model (SM) tree creator ............................ 68
  4.4.3. Output generator ....................................................... 68
4.5. FUNCTIONS AND ALGORITHMS ............................................... 68
  4.5.1. Main functions and algorithms .................................... 68
4.6. SCENARIO-BASED META-SCHEDULING (SBMeS) AND FAULT MODELING ..... 69
4.6.1. Fault assumptions ........................................................................................................... 70
  4.6.1.1. Tolerance threshold range ......................................................................................... 70
4.6.2. Fault-tolerant algorithm .................................................................................................. 71
4.7. THE GOALS OF META-SCHEDULER (MES) DESIGN .......................................................... 72

CHAPTER 5: VISUALIZATION AND EVALUATION OF SCHEDULES ........................................... 75

5.1. REQUIREMENTS FOR BASIC VISUALIZATION .................................................................. 75
5.2. REQUIREMENTS FOR META-VISUALIZATION ................................................................. 75
5.3. GRAPH MAPPING .............................................................................................................. 75
5.4. GANTT MAPPING .............................................................................................................. 76
5.5. VISUALIZATION OF SCHEDULE CHANGES .................................................................... 76
5.6. ENERGY CALCULATION ................................................................................................. 77
  5.6.1. Energy consumption ...................................................................................................... 77
  5.6.2. Energy reduction ........................................................................................................... 78
5.7. MEMORY AND META-SCHEDULES ................................................................................. 79
  5.7.1. Convergence of meta-schedules for saving memory ..................................................... 80
  5.7.2. Memory consumption................................................................................................... 80
5.7.3. DELTA SCHEDULING TECHNIQUE AND DELTA TREE ............................................... 81
5.8. SUMMARY .......................................................................................................................... 82

CHAPTER 6: IMPLEMENTATION .................................................................................................. 84

6.1. SCHEMA MODELLING ...................................................................................................... 84
6.2. IMPLEMENTATION OF THE DATA MODEL .................................................................... 85
6.3. IMPLEMENTATION OF META-SCHEDULING (MES) ......................................................... 85
  6.3.1. XML ............................................................................................................................ 86
6.4. IMPLEMENTATION OF META-SCHEDULING VISUALIZATION TOOL (MESVIZ) .............. 87
  6.4.1. Outputs and formats ..................................................................................................... 88
  6.4.2. Single schedule Gantt mapping ................................................................................... 88
  6.4.3. Multi-schedule Gantt mapping .................................................................................... 89
  6.4.4. Graph mapping of meta-schedules ............................................................................. 89

CHAPTER 7: EVALUATION EXAMPLE SCENARIOS AND RESULTS ........................................... 91

7.1. EVALUATION OBJECTIVES .............................................................................................. 91
7.2. VISUALIZING SCENARIO-BASED META-SCHEDULES FOR ADAPTIVE TIME-TRIGGERED SYSTEMS (TTS) .......................................................................................... 91
  7.2.1. Simple model ............................................................................................................... 91
  7.2.1.1. Schedule model (SM) content .................................................................................... 91
  7.2.1.2. Output results ........................................................................................................... 92
  7.2.2. Complex model (CM) .................................................................................................. 92
  7.2.2.1. Schedule model (SM) content .................................................................................... 92
  7.2.2.1.2. Output results ....................................................................................................... 93
  7.2.3. Discussion .................................................................................................................. 94
7.3. EVALUATION OF META-SCHEDULING (MES) RESULTS ............................................... 96
  7.3.1. Convergence results for saving memory ....................................................................... 96
  7.3.1.1. Example 1. Sample scenario with four tasks and three messages ............................ 96
  7.3.1.2. Results for delta scheduling technique (DST) and delta tree (DT) ........................... 100
  7.3.1.3. Example 2. Sample scenario with seven tasks and five messages and $N_{s,m} = 128$ ...................................................................................................................... 100
  7.3.1.4. Example 3. Big sample with $N_{s,k} = 9$, $N_{m,s} = 8$, and $N_{s,m} = 512$ schedules .......... 102
  7.3.1.5. Discussion ................................................................................................................. 102
  7.3.2. Scenario-based meta-scheduling (SBM')S) for frequency scaling of processors .......... 103
  7.3.2.1. Decision variables .................................................................................................. 103
7.3.2.2. Slow-down factor ................................................................. 103
7.3.2.3. Objective function ............................................................. 104
7.3.2.4. Input metadata for meta-scheduling (MeS) .................................. 104
7.3.2.5. Outputs and results .......................................................... 105
7.3.2.6. Overhead .................................................................. 105
7.3.2.7. Discussion ................................................................ 105

7.3.3. Scenario-based meta-scheduling (SBMeS) for frequency scaling of processors and network-on-chip (NoC) ................................................................. 106
  7.3.3.1. Scheduling constraints ....................................................... 106
  7.3.3.2. Decision variables ............................................................ 106
  7.3.3.3. Input metadata for meta-scheduling (MeS) ................................. 106
  7.3.3.4. Outputs and results .......................................................... 108
  7.3.3.5. Discussion ................................................................ 108
  7.3.3.6. Overhead .................................................................. 110

7.3.4. Scenario-based meta-scheduling (SBMeS) for frequency scaling with flexible task-to-processor mapping ................................................................. 111
  7.3.4.1. Input models .................................................................. 111
  7.3.4.2. Outputs and results .......................................................... 112
  7.3.4.3. Discussion ................................................................ 112
  7.3.4.4. Overhead .................................................................. 114

7.3.5. Scenario-based meta-scheduling (SBMeS) for improved reliability ................................................................. 115
  7.3.5.1. Input models .................................................................. 115
  7.3.5.2. Outputs and results .......................................................... 117
  7.3.5.3. Discussion ................................................................ 119

7.4. SUMMARY .................................................................. 122

CHAPTER 8: CONCLUSION AND FURTHER RESEARCH ............................................. 124

GLOSSARY .................................................................. 126

LIST OF FIGURES ................................................................. 128

LIST OF TABLES ................................................................. 131

LIST OF EXAMPLES ............................................................... 132

LIST OF ALGORITHMS ............................................................ 133

PUBLICATIONS AND PREVIOUS WORKS ............................................................. 134

I. REFERENCES ................................................................. 135
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].

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.
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/IEV¹ 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, DO – 178B, DO – 254, and 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 AS6802 [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 mean higher power consumption: including more cores thus presents a design hurdle of architectural complexity and higher energy consumption. For example, the Kalray 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].

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].

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

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

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

The use of \textit{DVFS} to increase power efficiency of the \textit{NoC} can be implemented on different levels (i.e., at the router level or the NI level). However, in this work, the \textit{NI} serves as a platform for handling \textit{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 \textit{integer linear programming (ILP)}, as a low complexity heuristic mapping algorithm.

However, few platforms can support full stage \textit{DVFS} at the router and interconnection level. For example, in \textit{Xilinx ZYNQ – 7000}, the input clock is 100 \textit{MHz} and the PL fabric clocks generated on the software side are equal to 85 \textit{MHz} and 170 \textit{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 \textit{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 \textit{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)

\textit{WCET} is the maximum execution time of a task, which is pre-estimated or pre-computed and uses analytical or measurement-based techniques. \textit{WCET} is an essential input for scheduling in \textit{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 \textit{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.
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, community scheduler frameworks, Moab cluster suite, Maui cluster scheduler, 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 CO₂ 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., CO₂ [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 \textit{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 \textit{slowdown factors} in \textit{multi scenario-based systems} for both cores, specifically using \textit{SBMeS} for static scheduling on \textit{adaptive TT MPSoC} and \textit{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 \textit{TT} networks, the application layer supports redundancy failures [114]. \textit{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)

<table>
<thead>
<tr>
<th>Related work</th>
<th>Techniques</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Meta, Super, hybrid scheduling</td>
<td>NoC</td>
</tr>
<tr>
<td></td>
<td>Core Fault</td>
<td>MIPS 4C</td>
</tr>
<tr>
<td></td>
<td>Visualization</td>
<td>Real-time system</td>
</tr>
<tr>
<td></td>
<td>DVFS</td>
<td>Fault-tolerant</td>
</tr>
<tr>
<td></td>
<td>Core/Task</td>
<td>Energy-efficient</td>
</tr>
<tr>
<td></td>
<td>routers</td>
<td>Mixed-criticality</td>
</tr>
<tr>
<td></td>
<td>MILP/GA/ILP</td>
<td></td>
</tr>
<tr>
<td>Eitschberger [28]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Cai et al. [116]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Murshed [90]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Wang et al. [91]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>J. Hu et al. [73]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tanq et al. [46]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Guy Avni et al. [87]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Jung et al. [103]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Fohler [102]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Anup Das et al. [104]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Ishak et al. [105]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Li et al. [81]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Majd et al. [108]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Huang et al. [115]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>J. Falk et al. [112]</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Maleki et al. [25]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>This Work</td>
<td>√</td>
<td>-</td>
</tr>
</tbody>
</table>

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]

<table>
<thead>
<tr>
<th>Project</th>
<th>Simulation</th>
<th>Visualization</th>
<th>&quot;Live&quot; simulation</th>
<th>Shared resources</th>
<th>Multiple Cores</th>
<th>Sporadic tasks</th>
<th>Open-source</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alea2</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>java</td>
</tr>
<tr>
<td>ARTIST</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>Cheddar</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td>Ada</td>
</tr>
<tr>
<td>FORTISSIMO</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>gltraceviz</td>
<td></td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>GHOST</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Jedule</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>java</td>
</tr>
<tr>
<td>MAST</td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
<td></td>
<td>√</td>
<td></td>
<td>Ada</td>
</tr>
<tr>
<td>Paje</td>
<td></td>
<td>√</td>
<td></td>
<td>√</td>
<td></td>
<td>√</td>
<td></td>
<td>Objective C</td>
</tr>
<tr>
<td>Realmss</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCL</td>
</tr>
<tr>
<td>RTsim</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>unknown</td>
</tr>
<tr>
<td>RTSSIM</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>RTISSim</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Schesim</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ruby</td>
</tr>
<tr>
<td>STORM</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>java</td>
</tr>
<tr>
<td>STRESS</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>ViTE</td>
<td></td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>VizzScheduler</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>java</td>
</tr>
<tr>
<td>MeSViz</td>
<td></td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C++</td>
</tr>
</tbody>
</table>

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.
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.
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](image)

### 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.
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.

![Flowchart](image)

*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>
<thead>
<tr>
<th>Domain</th>
<th>Constant name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM</td>
<td>N_{crs}</td>
<td>ℕ</td>
<td>Number of cores</td>
</tr>
<tr>
<td></td>
<td>N_{rr}</td>
<td>ℕ</td>
<td>Number of routers</td>
</tr>
<tr>
<td></td>
<td>N_{node}</td>
<td>ℕ</td>
<td>Number of nodes</td>
</tr>
<tr>
<td></td>
<td>ALLOC</td>
<td>ℕ</td>
<td>Allocation of tasks/tasks to cores (also routers, since one core connected to one router)</td>
</tr>
<tr>
<td>PM</td>
<td>HOPS</td>
<td>ℕ</td>
<td>Number of hops for each message</td>
</tr>
<tr>
<td></td>
<td>Path</td>
<td>ℕ</td>
<td>Each row is the path of a message, the maximum number of hops (columns) given by the number of routers</td>
</tr>
<tr>
<td></td>
<td>N_{task}</td>
<td>ℕ</td>
<td>Number of tasks</td>
</tr>
<tr>
<td>AM</td>
<td>N_{msg}</td>
<td>ℕ</td>
<td>Number of messages</td>
</tr>
<tr>
<td></td>
<td>WECT/ET</td>
<td>ℕ</td>
<td>Task worst case execution time</td>
</tr>
<tr>
<td></td>
<td>MD</td>
<td>ℕ</td>
<td>Duration of message transmission (depending on message length)</td>
</tr>
<tr>
<td></td>
<td>SRC</td>
<td>ℕ</td>
<td>Source task of messages</td>
</tr>
<tr>
<td></td>
<td>DEST</td>
<td>ℕ</td>
<td>Destination task of messages</td>
</tr>
<tr>
<td></td>
<td>CON</td>
<td>ℕ</td>
<td>Connectivity of routers</td>
</tr>
<tr>
<td></td>
<td>TSDF</td>
<td>ℕ</td>
<td>SDF of all tasks</td>
</tr>
</tbody>
</table>

Table 3. Overview of input table [90]
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\text{IN}(t)**: the set of input messages of a task \( t \)

**MSG\text{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:

\[
md_{\text{total}}(m) = md(m) \cdot (hops(m) + 1) \quad (1)
\]

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

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

\[
f_{\text{sel}} \leq f_{\text{max}} \quad (2)
\]

### 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:

$$ALLOC = \begin{bmatrix} alloc_1 \\ \vdots \\ alloc_k \end{bmatrix} \in \{1, ..., |CRS|\}^k$$

$$k = \left| TSK \right|$$

(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 $m \in MSG$:

$$MD = \begin{bmatrix} md_1 \\ \vdots \\ md_n \end{bmatrix} \in \{1, ..., Max\}^n \quad | n = |MSG|$$

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

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, \( T_4 \) depends on \( T_0, T_1, T_2, \) and \( T_3 \).

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

\[
\forall t \in \text{TSK}. \ \forall m_i \in \text{MSG}_\text{IN}(t). \ \forall m_o \in \text{MSG}_\text{OUT}(t). \ \ t_{\text{inject}}(m_i) + \overline{md}_{\text{total}}(m_i) + \overline{\epsilon}(t) \leq t_{\text{inject}}(m_o) \quad (6)
\]

![Figure 12. Conceptual AM](image)

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 \( m \in \text{MSG} \):

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

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

\[
 D(m) \leq t_{\text{read,dest}}(m) \quad (8)
\]

3.8. Decision variables

3.8.1. Slowdown factors (SDFs)

An essential decision variable in our \textbf{DVFS} optimizations model is the \textit{SDF} of tasks and messages. \( tsdf(t) \) denotes the \textit{SDF} of a core for computing task \( t \) and \( msdf(m) \) denotes the \textit{SDF} for transmitting message \( m \). Assuming that the contextual parameter \( C \) is zero,
then $tsdf(t)$ is proportional to the effective execution time $\bar{e}t(t)$: $tsdf(t) \propto \bar{e}t(t)$, and $msdf(m)$ is proportional to the effective message duration $\bar{md}(m)$: $msdf(t) \propto \bar{md}(m)$.

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

$$TSDF = \begin{bmatrix} tsdf_1 \\ \vdots \\ tsdf_k \end{bmatrix} \in \{tsdf_{min}, \ldots, tsdf_{max}\}^k$$

$$k = |TSK|, \quad tsdf_{min} \geq 1 \quad (9)$$

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

$$MSDF = \begin{bmatrix} msdf_1 \\ \vdots \\ msdf_n \end{bmatrix} \in \{msdf_{min}, \ldots, msdf_{max}\}^n$$

$$n = |MSG|, \quad msdf_{min} \geq 1 \quad (10)$$

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

$$\bar{e}t(t) = et(t) \cdot tsdf(t) + C \quad (11)$$

$$\bar{md}(m) = md(m) \cdot msdf(m) + C \quad (12)$$

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

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

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:

$$HOPS = \begin{bmatrix} hops_1 \\ \vdots \\ hops_n \end{bmatrix} \in \{1, \ldots, MaxHop\}^n$$
\[ n = |MSG| \quad (14) \]

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. \textit{HOPS} in [13] and [90] is used as a decision variable to specify the number of visited routers hops, of a message m to the destination task. In the absence of cyclic paths, the maximum \textit{HOPS} count is \( MaxHop = RS + 1 \).

3.9. Energy consumption

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

\[ P = P_{\text{dyn}} + P_{\text{stc}} \quad (15) \]

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

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

\( A \) 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 \( S_{\text{effective}} \) is given as follows:

\[ S_{\text{effective}} = A \times S \quad (17) \]

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

\[ P_{\text{dyn}} \propto f \cdot U^2 \quad (18) \]

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

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

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

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

\[ et(t) \propto \frac{1}{f} \quad \text{and} \quad md(m) \propto \frac{1}{f} \quad (21) \]
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 \( f_{\text{max}} \). However, the effective execution time, denoted as \( \bar{et}(t) \), and the effective message duration, denoted as \( \bar{md}(m) \), depend on the selected clock frequency \( f_{\text{sel}} \) and a contextual parameter \( C \):

\[
\bar{et}(t) = et(t) \cdot \frac{f_{\text{max}}}{f_{\text{sel}(t)}} + C \quad (22)
\]

\[
\bar{md}(m) = md(m) \cdot \frac{f_{\text{max}}}{f_{\text{sel}(m)}} + C \quad (23)
\]

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 \( P_{\text{dyn}} \) is thus proportional to the cube of the clock frequency:

\[
P_{\text{dyn}} \propto f^3 \quad (24)
\]

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

\[
E_{T,\text{dyn}}(t) \propto f^3 \cdot et(t) \quad (25)
\]

\[
E_{m,\text{dyn}}(m) \propto f^3 \cdot md(m) \quad (26)
\]

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 \( \bar{FE}_T \) and the effective message frequency co-efficient is denoted as \( \bar{FE}_m \). 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.

\[
FE_T = et(t) \cdot f^3 \quad (27)
\]

\[
\bar{FE}_T = et(t) \cdot (f_{\text{max}}/f_{\text{sel}(t)})^3 \quad (28)
\]

\[
FE_m = md(m) \cdot f^3 \quad (29)
\]

\[
\bar{FE}_m = (m) \cdot (f_{\text{max}}/f_{\text{sel}(m)})^3 \quad (30)
\]

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

![Diagram](image)

*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
\[ F_M(m) = f^3 \cdot md(m) \] 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

\[
P_{task} \propto tsdf(t)^3 \cdot et(t) \quad (31)
\]

\[
P_{message} \propto msdf(m)^3 \cdot md(m) \quad (32)
\]

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 \( T_0 \), after reduction frequency from \( f_{max} \) to \( f_{sel(t_0)} \), changes to \( \bar{et}(t_0) \) and \( et(t_0) < \bar{et}(t_0) \).

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>
<thead>
<tr>
<th>( et(t_0) )</th>
<th>( et(t_1) )</th>
<th>( et(2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>20s</td>
<td>30s</td>
<td>10s</td>
</tr>
<tr>
<td>( f_{max} )</td>
<td>( f_{max} )</td>
<td>( f_{max} )</td>
</tr>
<tr>
<td>100 Mhz</td>
<td>100 Mhz</td>
<td>100 Mhz</td>
</tr>
<tr>
<td>tsdf(t0)</td>
<td>tsdf(t1)</td>
<td>tsdf(2)</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( \bar{et}(t_0) )</td>
<td>( \bar{et}(t_1) )</td>
<td>( \bar{et}(t_2) )</td>
</tr>
<tr>
<td>80s + C</td>
<td>150s + C</td>
<td>20s + C</td>
</tr>
</tbody>
</table>

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

![Diagram](image)

**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):
∀m ∈ MSG . msdf\textsubscript{const}(m) = 1 \quad (33)

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

∀t ∈ TSK. ∀m ∈ MSG .

\[ CP_1(m, t) = et(t) \cdot tsdf(t)^2 + md(m) \cdot msdf\textsubscript{const}(m)^2 \cdot (hops(m) + 1) \quad (34) \]

\[ RES_1 = \text{maximize} \left( \sum_{i=1}^{\lvert TSK \rvert} \sum_{j=1}^{\lvert MSG \rvert} (CP_1(m_j, t_i)) \right) \quad (35) \]

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

**Phase 2**: based on the optimization result, RES\textsubscript{1}, the second optimisation phase is completed. First, the hop counts are extracted from the intermediate result RES\textsubscript{1}:

∀m ∈ MSG . hops\textsubscript{const}(m) is extracted from RES\textsubscript{1} \quad (36)

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

∀t ∈ TSK. ∀m ∈ MSG .

\[ CP_2(m, t) = et(t) \cdot tsdf(t)^2 + md(m) \cdot msdf(m)^2 \cdot (hops\textsubscript{const}(m) + 1) \quad (37) \]

\[ RES_2 = \text{maximize} \left( \sum_{i=1}^{\lvert TSK \rvert} \sum_{j=1}^{\lvert MSG \rvert} (CP_2(m_j, t_i)) \right) \quad (38) \]

**Goal**: in this step, SM\textsubscript{0} with SS generate and maximum makespan is discovered.

**Phase 3**: finally, makespan(SSM) = makespan(SM\textsubscript{0}) are used as the timing constraint for \textit{flt(sm)} and the objective function is equal to RES\textsubscript{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).
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:

\[
\forall sm \in SSM/\{SM_0\}. \quad mks(SM_0) > mks(sm) \quad (39)
\]

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](image)

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:

\[
\text{Max}(et_{t_5}) = et(t_5) + DS_0 + SS_2 \quad (41)
\]

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

\[
et(t_5) \cdot TSDF(t_5) \leq \text{Max}(et_{t_5}) \quad (42)
\]
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)
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 schedule 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-Visualizer(AM,PM,SM)
    function Output(AM,PM,SM)
end function
end function
function Meta-Visualizer(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.
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.

$$\forall c \in CRS, \forall flt(c) \in FLT(sm, c). flt(c) = 0, \forall e_x \in Event(sm)$$

**Fault recovery function (AM, PM, 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 $SM_x$. 
For example, in $SM_1$ $ev(flt(s))$ slacks (for $T_1$ and $T_4$) and $TSDF$ are added to these tasks (e0). In $SM_2$, the $ev(flt(c))$ fault in CRS 3 $flt(c_3)$ causes $T_2$ 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 $SM_{0,1,2,3}$, $T_0$ is repeated using the same location and size, $SM_{1,2,3}$ $T_1$, and $SM_{2,3}$ $T_2$ is repeated.

### 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).
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.

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](image)

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, $FE_{avg,dyn}(SSM/\{SM_0\})$.

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

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

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:

$$FE_C(sm) = \sum_{c \in CRS} FE_C(c, sm) \quad (44)$$
The energy \( FE_R(r, sm) \) consumed at any router \( r \in RTR \) for all its messages can be calculated as,

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

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 \( f_{\text{max}} = 1 \).

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 \( FE_{C,\text{avg, dyn}} \) of all cores for all scheduling models with DS can be calculated as follows:

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

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

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

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

\[
FE_{\text{avg, dyn}} = FE_{R,\text{avg, dyn}} + FE_{C,\text{avg, dyn}} \quad (50)
\]

5.6.2. Energy reduction

\textit{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 \( sm \in (SSM/\{SM_0\}) \), 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 (\( ReFE_R(SM) \)), each SM (\( ReFE_{sm} \)) and combined (\( ReFE \)) as follows:

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

\[
ReFE_R(SM) = \frac{FE_R(SM_0) - FE_R(SM)}{FE_R(SM_0)} \cdot 100\% \quad (52)
\]
\[ \text{ReFE}_C = \frac{\text{FE}_C(SM_0) - \text{FE}_C,\text{avg, dyn}}{\text{FE}_C(SM_0)} \cdot 100\% \]  
(53)

\[ \text{ReFE}_R = \frac{\text{FE}_R(SM_0) - \text{FE}_R,\text{avg, dyn}}{\text{FE}_R(SM_0)} \cdot 100\% \]  
(54)

\[ \text{ReE}_{sm} = \frac{\text{FE}(SM_0) - \text{FE}(SM)}{\text{FE}(SM_0)} \cdot 100\% \]  
(55)

\[ \text{ReFE} = \frac{\text{FE}(SM_0) - \text{FE}_{\text{avg, dyn}}}{\text{FE}(SM_0)} \cdot 100\% \]  
(56)

5.7. Memory and meta-schedules

To avoid wasting memory and reaching the memory limits of \textit{NoCs} and \textit{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 \textit{SSM} for message duplication is reduced, but we combined (the stored memory) with a task duplication method, which made \textit{SSM} memory usage more efficient.

Figure 28 represents the effects of a core fault and slack event on \textit{SSM}, and Figure 29 shows how many share points and how much duplicated data (in this case tasks) are stored in related \textit{SMs}, thus wasting memory space. For example, \textit{SM}_2 and \textit{SM}_3 have similar amounts of data.
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, $M_{tim}$, 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 $N_{ft(c)}$, $N_{tsk} = 12$ (meaning two tasks do not have slack), and $N_{msg} = 14$, then $N_e = N_{tsk,s} + N_{ft(c)}$, which in this case is $N_e = 14$.

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

If $N_{ssm} = 2^{N_e}$ 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 $M_{ssm} = 70 \times 16384$; $TMsch = 1,146,880$.

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 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:

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

(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:

$$SavM = M_{ssm} - M_{real(ssm)}$$

(58)

**Function DTS (SM$_x$, SM$_{x+1}$)**

If SM$_x$ is Parent and SM$_{x+1}$ is Child →

(a)

For (int i = 0; i < N$_{msg}$; i++)

If (SM$_x$(M[i].inject_time) ≠ 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) ≠ SM$_{x+1}$(T[i].start_time))

OR

(SM$_x$(T[i].End_time) ≠ 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).

\[ SM_x + DTS(SM_{x+1}) \rightarrow SM_{x+1} \]

*Figure 30. Decoding schedules from delta tree (DT)*

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.

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

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

<table>
<thead>
<tr>
<th>Application Data</th>
<th>Physical Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task ID=0 ET=50 energy=[1,100]</td>
<td>Link ID=0 from node#0 to node#5</td>
</tr>
<tr>
<td>Task ID=1 ET=60 energy=[1,100]</td>
<td>Link ID=1 from node#1 to node#5</td>
</tr>
<tr>
<td>Task ID=2 ET=42 energy=[1,100]</td>
<td>Link ID=2 from node#2 to node#5</td>
</tr>
<tr>
<td>Msg. ID=0 from job#2 to job#1</td>
<td></td>
</tr>
<tr>
<td>(Msg. ID=1 from job#0 to job#4</td>
<td></td>
</tr>
<tr>
<td>(Msg. ID=2 from job#1 to job#3</td>
<td></td>
</tr>
</tbody>
</table>
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 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).

```xml
<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)](image)

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++ 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.

---

8 This concept and figure is used on SafePower D3.8 [3].
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.
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).

\[
diagram G \\
\text{digraph } G \{
0 \rightarrow 1 [\text{label} = \text{"Slack Event (Task \#6, new ET = 10)" color = green}]; \\
1 \rightarrow 2 [\text{label} = \text{"Slack Event (Task \#8, new ET = 10)" color = green}]; \\
2 \rightarrow 3 [\text{label} = \text{Slack Event (Task \#0, new ET=25) color = green}]; \\
\}
\]

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\textsuperscript{10} is used. For big data, the yEd\textsuperscript{11} graph editor and Gephi\textsuperscript{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

---

\textsuperscript{10} \url{www.graphviz.org}

\textsuperscript{11} \url{www.yworks.com}

\textsuperscript{12} \url{gephi.org}
Chapter 7: Evaluation example scenarios and results

This section presents the results of the linearized MIQP\textsuperscript{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

<table>
<thead>
<tr>
<th>Section</th>
<th>TSDF</th>
<th>MSDF</th>
<th>Multi-task</th>
<th>Dynamic-sack</th>
<th>Core fault</th>
<th>Save memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.1</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

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.

\textsuperscript{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 \textit{MeSViz} generates six Gantt maps (Figure 27) from \textit{MeS}. GVEEdit generates graph mapping with 37 nodes.

Figure 37 is a sample of 94 visualized schedules generated by \textit{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)](image)

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.

![Schedule tree with 94 schedules](image)

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.
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).

```plaintext
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**
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 $N_s = 4$, 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 $Q_msch$; 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)](image_url)
Figure 43. Schedule SM₁ with dynamic slack (DS)

Figure 44. Comparing two schedules SM₀ (Figure 42) and SM₁ (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](image)

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:

\[
M_{sm} = 5 \times 4 + 5 \times 3; \quad M_{sm} = 35.
\]

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

\[
M_{ssm} = 35 \times 16; \quad M_{ssm} = 560.
\]
In Figure 46, the data for message $M_2$ and tasks $T_2$ and $T_3$ changed; while in Figure 47, only the data of $T_3$ 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 46. Few changes from ID2 to ID3

Figure 47. Minimum changes from $SM_{37}$ to $SM_{38}$ regarding $T_3$ 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 $M_{tsk} = 5\; Bit$ and $M_{msg} = 5\; Bit$. The results of generating the schedules are presented in Table 7.

Table 7. Results of dynamic slack (DS) schedules

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

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)

<table>
<thead>
<tr>
<th>$N_{msg}$</th>
<th>$N_{tsk}$</th>
<th>$M_{sm}$</th>
<th>$N_{sm}$</th>
<th>$M_{ssm}$</th>
<th>$SavM$ (Byte)</th>
<th>$SavM%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>35</td>
<td>16</td>
<td>560</td>
<td>340</td>
<td>61%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>sm</th>
<th>(N_{ch(msg)})</th>
<th>(N_{ch(tsk)}) (\times) (Mms)</th>
<th>(N_{ch(tsk)}) (\times) (Mts)</th>
<th>(M_{sm})</th>
<th>SavM (\text{Byte})</th>
<th>Makespan (ms)</th>
<th>SavE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>25</td>
<td>30</td>
<td>55</td>
<td>5</td>
<td>1610</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>15</td>
<td>25</td>
<td>35</td>
<td>1610</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>25</td>
<td>45</td>
<td>15</td>
<td>1460</td>
</tr>
<tr>
<td>108</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>45</td>
<td>1391</td>
</tr>
<tr>
<td>109</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>50</td>
<td>1381</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1131</td>
</tr>
<tr>
<td>114</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1210</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1541</td>
</tr>
<tr>
<td>122</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1291</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1281</td>
</tr>
<tr>
<td>108</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>45</td>
<td>1391</td>
</tr>
<tr>
<td>109</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>50</td>
<td>1381</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1131</td>
</tr>
<tr>
<td>114</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1210</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>55</td>
<td>1541</td>
</tr>
</tbody>
</table>

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\% |
|-----------|---------|-----|-------|---------|utron |---|---|
| 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

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

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

<table>
<thead>
<tr>
<th>( N_{msg} )</th>
<th>( N_{tsk} )</th>
<th>( M_{sm} )</th>
<th>( N_{sm} )</th>
<th>( M_{ssm} )</th>
<th>( SavM (Byte) )</th>
<th>( SavM% )</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>9</td>
<td>80</td>
<td>512</td>
<td>40960</td>
<td>35065</td>
<td>86%</td>
</tr>
</tbody>
</table>

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 DST 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.

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

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.

\[
\forall t \in TSK. \ \forall m \in MSG
CP_3(t) = et(t) \cdot tsdf(t)^2 \quad (60)
\]

\[
RES_3 = \text{maximize} \ (\sum_{i=1}^{\mid TSK \mid} (CP_3(t_i))) \quad (61)
\]

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](image)

**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](image)

Table 13. Meta-scheduling (MeS) input constant
<table>
<thead>
<tr>
<th>Input Model</th>
<th>Input Name</th>
<th>ID</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>TSK</td>
<td>0</td>
<td>WCET=2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>WCET=4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>WCET=6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>WCET=8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>WCET=10</td>
</tr>
<tr>
<td>MSG</td>
<td></td>
<td></td>
<td>Quantity=6 ID start=0</td>
</tr>
<tr>
<td>Deadline (D(m))</td>
<td>All task= 1185</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tsdf (t)</td>
<td></td>
<td></td>
<td>All task: min =1 &amp; max =100</td>
</tr>
<tr>
<td>Slack event</td>
<td></td>
<td></td>
<td>All task= 50%</td>
</tr>
<tr>
<td>PM</td>
<td>RTR</td>
<td></td>
<td>Quantity=2 ID start=0</td>
</tr>
<tr>
<td></td>
<td>CRS</td>
<td></td>
<td>Quantity=5 ID start=6</td>
</tr>
<tr>
<td></td>
<td>Link</td>
<td></td>
<td>Quantity=6 ID start=0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Nmsg</th>
<th>Ntsk</th>
<th>Msm</th>
<th>Nsm</th>
<th>Mssm</th>
<th>SavM (Byte)</th>
<th>SavM%</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>55</td>
<td>94</td>
<td>5170</td>
<td>4037</td>
<td>78%</td>
</tr>
</tbody>
</table>

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 (SM0) with SS technique, schedule SM49 with minimum energy reduction, and schedule SM5 with maximum energy reduction using the DS technique.

Table 15. Some collected results for model example

<table>
<thead>
<tr>
<th>SM ID</th>
<th>Slack Mode</th>
<th>Task ID</th>
<th>et(t)</th>
<th>tsdf (t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Static</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>49</td>
<td>Dynamic</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

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 SM0. The
results show that the MeS algorithm can reduce energy usage by DS in robust and adaptive TTS.

Table 16. Energy results for example

<table>
<thead>
<tr>
<th>SM</th>
<th>Slack Mode</th>
<th>FE(sm)</th>
<th>ReFE(sm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Static</td>
<td>1.875</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>Dynamic</td>
<td>1.62</td>
<td>13.6%</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic</td>
<td>0.6675</td>
<td>64.4%</td>
</tr>
<tr>
<td>Average</td>
<td>Dynamic</td>
<td>1.0948</td>
<td>41.61%</td>
</tr>
</tbody>
</table>

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]

<table>
<thead>
<tr>
<th>Input Model</th>
<th>Input Name</th>
<th>ID</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>TSK</td>
<td>0</td>
<td>WCET=2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>WCET=4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>WCET=6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>WCET=8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>WCET=10</td>
</tr>
<tr>
<td>PM</td>
<td>MSG</td>
<td></td>
<td>Quantity=6 ID start=0</td>
</tr>
<tr>
<td></td>
<td>Deadline (D(m))</td>
<td>All task= 1185</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tsdf(t)</td>
<td>All task and messages: min =1 &amp; max =100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Slack event</td>
<td>All task= 50%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTR</td>
<td></td>
<td>Quantity=2 Start ID==0</td>
</tr>
<tr>
<td></td>
<td>CRS</td>
<td>Quantity=5 Start ID==6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Link</td>
<td>Quantity=6 Start ID=0</td>
<td></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>SM</th>
<th>Slack Mode</th>
<th>FE(sm)</th>
<th>ReFE_{sm}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Static</td>
<td>1.875</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>Dynamic</td>
<td>1.62</td>
<td>13.6%</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic</td>
<td>0.6675</td>
<td>64.4%</td>
</tr>
<tr>
<td>Average</td>
<td>Dynamic</td>
<td>1.0948</td>
<td>41.61%</td>
</tr>
</tbody>
</table>

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 SDF’s for both cores and routers.

Table 19. General results for a model example
<table>
<thead>
<tr>
<th>SM ID</th>
<th>CRS</th>
<th>RTR</th>
<th>Re CRS</th>
<th>Re RTR</th>
<th>Total</th>
<th>Re Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.65</td>
<td>4.03333</td>
<td>0.0000%</td>
<td>0.0000%</td>
<td>5.68333</td>
<td>0.0000%</td>
</tr>
<tr>
<td>1</td>
<td>1.3625</td>
<td>2.77778</td>
<td>17.4242%</td>
<td>31.1295%</td>
<td>4.14028</td>
<td>27.1505%</td>
</tr>
<tr>
<td>2</td>
<td>1.3325</td>
<td>3.63333</td>
<td>19.2424%</td>
<td>9.9174%</td>
<td>4.96583</td>
<td>12.6246%</td>
</tr>
<tr>
<td>5</td>
<td>0.6675</td>
<td>3.14444</td>
<td>59.5455%</td>
<td>22.0386%</td>
<td>3.81194</td>
<td>32.9277%</td>
</tr>
<tr>
<td>18</td>
<td>0.9925</td>
<td>3.14444</td>
<td>39.8485%</td>
<td>22.0386%</td>
<td>4.1369</td>
<td>27.2099%</td>
</tr>
<tr>
<td>38</td>
<td>1.0225</td>
<td>3.05556</td>
<td>24.2424%</td>
<td>4.07806</td>
<td>28.2452%</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>0.9825</td>
<td>3.05556</td>
<td>40.4545%</td>
<td>24.2424%</td>
<td>4.03806</td>
<td>28.9490%</td>
</tr>
<tr>
<td>46</td>
<td>1.1825</td>
<td>3.05556</td>
<td>28.3333%</td>
<td>24.2424%</td>
<td>4.23806</td>
<td>25.4300%</td>
</tr>
<tr>
<td>49</td>
<td>1.4625</td>
<td>3.63333</td>
<td>11.3636%</td>
<td>9.9174%</td>
<td>5.09583</td>
<td>10.3372%</td>
</tr>
<tr>
<td>63</td>
<td>0.9225</td>
<td>3.14444</td>
<td>44.0909%</td>
<td>22.0386%</td>
<td>4.06694</td>
<td>28.4409%</td>
</tr>
<tr>
<td>70</td>
<td>1.2825</td>
<td>3.14444</td>
<td>22.2727%</td>
<td>22.0386%</td>
<td>4.42694</td>
<td>22.1066%</td>
</tr>
<tr>
<td>80</td>
<td>1.1</td>
<td>3.05556</td>
<td>33.3333%</td>
<td>24.2424%</td>
<td>4.15556</td>
<td>26.8816%</td>
</tr>
<tr>
<td>93</td>
<td>1.38</td>
<td>3.05556</td>
<td>16.3636%</td>
<td>24.2424%</td>
<td>4.43556</td>
<td>21.9549%</td>
</tr>
<tr>
<td>Avg</td>
<td>1.10363</td>
<td>3.11918</td>
<td>33.1134%</td>
<td>22.6651%</td>
<td>4.22281</td>
<td>25.6983%</td>
</tr>
</tbody>
</table>

**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 \(\text{ReFE}_C(SM)\) and routers \(\text{ReFE}_R(SM)\) and average \(\text{FE}_C,\text{avg}_{.,\text{dyn}}, \text{FE}_R,\text{avg}_{.,\text{dyn}}\)

Figure 57. Total FE reduction results for schedules \(\text{ReE}_{sm}\) and average \(\text{ReFE}\)

7.3.3.6. Overhead

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

<table>
<thead>
<tr>
<th>(N_{msg})</th>
<th>(N_{tSk})</th>
<th>(M_{sm})</th>
<th>(N_{sm})</th>
<th>(M_{ssm})</th>
<th>(\text{SavM (Byte)})</th>
<th>(\text{SavM%})</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>55</td>
<td>94</td>
<td>5170</td>
<td>3980</td>
<td>77%</td>
</tr>
</tbody>
</table>

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, \(\text{SavM}\) 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>
<thead>
<tr>
<th>Input Model</th>
<th>Input Name</th>
<th>ID</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>TSK</td>
<td>0</td>
<td>WCET=2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>WCET=4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>WCET=6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>WCET=8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>WCET=10</td>
</tr>
<tr>
<td></td>
<td>MSG</td>
<td></td>
<td>Quantity=6 ID start=0</td>
</tr>
<tr>
<td></td>
<td>TSDF</td>
<td></td>
<td>All Task: min =1 &amp; max =100</td>
</tr>
<tr>
<td></td>
<td>MSDF</td>
<td></td>
<td>All Messages: min =1 &amp; max =100</td>
</tr>
<tr>
<td></td>
<td>Slack event</td>
<td></td>
<td>All Task= 50%</td>
</tr>
<tr>
<td>PM</td>
<td>RTR</td>
<td></td>
<td>Quantity=3 ID start=0</td>
</tr>
<tr>
<td></td>
<td>CRS</td>
<td></td>
<td>Quantity=5 ID start=6</td>
</tr>
<tr>
<td></td>
<td>Link</td>
<td></td>
<td>Quantity=9 ID start=0</td>
</tr>
</tbody>
</table>

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](image)

### 7.3.4.2. Outputs and results

MeS at 200 seconds generated 93 schedules for the DS scenario SSM/{SM₀} and one schedule for the SS SM₀. 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 SM₀ is used for the SS time, and the schedule SM₁ to SM₉₃ is used for DS time. As seen in the results, schedule SM₅ has lowest energy reduction and schedule SM₈₀ has the highest.

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

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

**Figure 61.** $FE(sm)$ results 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>
<thead>
<tr>
<th>$N_{msg}$</th>
<th>$N_{task}$</th>
<th>$M_{sm}$</th>
<th>$N_{sm}$</th>
<th>$M_{ssm}$</th>
<th>$SavM$ (Byte)</th>
<th>$SavM$ %</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>55</td>
<td>94</td>
<td>5170</td>
<td>4208</td>
<td>81%</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Status</th>
<th>Group</th>
<th>Scenario</th>
<th>Test</th>
<th>GRS</th>
<th>RTT</th>
<th>TSK</th>
<th>MSG</th>
<th>Links-ID</th>
<th>Slack</th>
<th>MSDF</th>
<th>TSDF</th>
<th>fit(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>I</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>I</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>I</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>I</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>I</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>II</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>II</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>II</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>II</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>II</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>II</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>II</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>II</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>II</td>
<td>8</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>II</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>II</td>
<td>9</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>II</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>II</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
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 Status12), links l8 and l9 are disabled, one task has slack, and one core failed. In I4T1 Status9, every parameter is the same, with the exception of the links connection. Comparing II8T1 Status20 to II6T1 Status14, 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

<table>
<thead>
<tr>
<th>Input Model</th>
<th>Input Name</th>
<th>ID</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM Task</td>
<td>0</td>
<td>WCET=2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>WCET=4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>WCET=6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>WCET=8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>WCET=4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>WCET=4</td>
<td></td>
</tr>
<tr>
<td>MSG Quantity=5</td>
<td>ID start=0 Du_m=4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deadline (D(m))</td>
<td>All Task: min =1 &amp; max =100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t sdf(t)</td>
<td>All Messages: min =1 &amp; max =100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slack event</td>
<td>All Task= 50%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PM CRS</td>
<td>Quantity=5 ID start=6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link</td>
<td>Quantity=10 ID start=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM (FaultEvent) type=crash type=Slack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Makespan</td>
<td>250-1000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SSM$_x$</th>
<th>Total $N_{sm}$</th>
<th>Effective $N_{sm}$</th>
<th>Real $N_{sm}$</th>
<th>Times (seconds)</th>
<th>FE$_{avg,dyn}$</th>
<th>ReFE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>191</td>
<td>127</td>
<td>64</td>
<td>416</td>
<td>2.35</td>
<td>23.63%</td>
</tr>
<tr>
<td>2</td>
<td>46</td>
<td>38</td>
<td>8</td>
<td>348</td>
<td>2.49</td>
<td>19.89%</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>23</td>
<td>4</td>
<td>342</td>
<td>2.55</td>
<td>18%</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>13</td>
<td>2</td>
<td>335</td>
<td>2.72</td>
<td>12%</td>
</tr>
<tr>
<td>5</td>
<td>383</td>
<td>255</td>
<td>128</td>
<td>945</td>
<td>2.31</td>
<td>25.56%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1146</td>
<td>762</td>
<td>256</td>
<td>3457</td>
<td>2.42</td>
<td>22.14%</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>14</td>
<td>2</td>
<td>945</td>
<td>2.96</td>
<td>4.79%</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>28</td>
<td>4</td>
<td>1800</td>
<td>2.90</td>
<td>7%</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>26</td>
<td>4</td>
<td>735</td>
<td>2.63</td>
<td>15.3%</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>52</td>
<td>8</td>
<td>2558</td>
<td>2.59</td>
<td>16.66%</td>
</tr>
<tr>
<td>11</td>
<td>120</td>
<td>104</td>
<td>16</td>
<td>4228</td>
<td>2.62</td>
<td>16%</td>
</tr>
<tr>
<td>12</td>
<td>30</td>
<td>26</td>
<td>4</td>
<td>792</td>
<td>2.63</td>
<td>15%</td>
</tr>
<tr>
<td>13</td>
<td>60</td>
<td>52</td>
<td>8</td>
<td>2565</td>
<td>2.59</td>
<td>16.66%</td>
</tr>
<tr>
<td>14</td>
<td>191</td>
<td>127</td>
<td>64</td>
<td>480</td>
<td>2.37</td>
<td>23.63%</td>
</tr>
<tr>
<td>15</td>
<td>46</td>
<td>38</td>
<td>8</td>
<td>354</td>
<td>2.49</td>
<td>19.89%</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>14</td>
<td>2</td>
<td>940</td>
<td>2.96</td>
<td>4.79%</td>
</tr>
<tr>
<td>17</td>
<td>191</td>
<td>127</td>
<td>64</td>
<td>216</td>
<td>40.25</td>
<td>1.28%</td>
</tr>
<tr>
<td>18</td>
<td>46</td>
<td>38</td>
<td>8</td>
<td>157</td>
<td>40.31</td>
<td>1.12%</td>
</tr>
<tr>
<td>19</td>
<td>27</td>
<td>23</td>
<td>4</td>
<td>137</td>
<td>40.50</td>
<td>0.6%</td>
</tr>
<tr>
<td>20</td>
<td>46</td>
<td>38</td>
<td>8</td>
<td>72</td>
<td>26.62</td>
<td>8.55%</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>13</td>
<td>4</td>
<td>53</td>
<td>28.79</td>
<td>1.13%</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>14</td>
<td>2</td>
<td>131</td>
<td>40.52</td>
<td>0.6%</td>
</tr>
<tr>
<td>23</td>
<td>32</td>
<td>28</td>
<td>4</td>
<td>223</td>
<td>40.22</td>
<td>1.31%</td>
</tr>
<tr>
<td>24</td>
<td>64</td>
<td>56</td>
<td>16</td>
<td>637</td>
<td>39.08</td>
<td>4.16%</td>
</tr>
</tbody>
</table>

**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., $c_0\text{ status}=0$) 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 $l_9$ and $l_8$, 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.

![Number of total generated schedules](image1)

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

![Time of computation for scenarios](image2)

*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.

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

<table>
<thead>
<tr>
<th>SSMx</th>
<th>Links-ID</th>
<th>Slack</th>
<th>MSDF</th>
<th>TSDF</th>
<th>f(t(G))</th>
<th>Total $N_{rm}$</th>
<th>Effective $N_{rm}$</th>
<th>Real $N_{rm}$</th>
<th>Times (seconds)</th>
<th>$FE_{avg,dyn}$</th>
<th>RefFE(SSM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>191</td>
<td>127</td>
<td>64</td>
<td>416</td>
<td>2.35</td>
<td>23.63</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>191</td>
<td>127</td>
<td>64</td>
<td>480</td>
<td>2.37</td>
<td>23.63</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>46</td>
<td>38</td>
<td>8</td>
<td>348</td>
<td>2.49</td>
<td>19.89</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>46</td>
<td>38</td>
<td>8</td>
<td>354</td>
<td>2.49</td>
<td>19.89</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>60</td>
<td>52</td>
<td>8</td>
<td>2558</td>
<td>2.59</td>
<td>16.66</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>60</td>
<td>52</td>
<td>8</td>
<td>2565</td>
<td>2.59</td>
<td>16.66</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>16</td>
<td>14</td>
<td>2</td>
<td>945</td>
<td>2.96</td>
<td>4.79</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>16</td>
<td>14</td>
<td>2</td>
<td>940</td>
<td>2.96</td>
<td>4.79</td>
</tr>
</tbody>
</table>

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 (ReFEc) and 31.12% for routers (ReFER). 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 \textit{TTS}. The results show that our DS time consideration and frequency slowdown in \textit{MeS}, compared to the SS time, produces an energy reduction of up to 57\% for cores (\textit{ReFE}_c) and 52.46\% (\textit{ReFE}_R) for routers. The energy reduction is, on average, up to 46.73\% in a single schedule and 33.88\% for \textit{NoCs}.

\textbf{F)} In section 7.3.5, compared to section 7.3.4, the \textit{SBMeS} technique and \textit{MeS} tool was improved for network design and expandability, revealing no significant impact on power consumption in the schedules when disabling redundancy links. \textit{MeS} is used in our multi-scenario-based (e.g., fault, safety, power-saving) scheduling and adaptive \textit{TT} systems problems. The results show that our DS and \textit{SDF} method and algorithm based on \textit{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

A
Adaptive, 1, 91
Adaptive TT, 20, 40, 124
Adaptivity, 24
algorithm, 17, 19, 20, 22, 24, 27, 29, 30, 31, 33, 37, 63, 77, 103, 106, 107, 108, 124
Algorithm, 18, 64, 66
AM, 42, 43, 64, 69, 75, 85, 86, 88, 91, 92, 104, 105, 107
Application model, 43
automotive, 16
automotive functions, 7

C
CM, 42, 44, 69, 85, 86, 91, 92
Collision, 49
communication, 9, 16, 17, 19, 20, 26, 30, 31, 35, 42, 106
complex integrated systems, 16
computer science, 16, 18
Connectivity, 50
Constants, 48
Constraints, 48, 50
Context model, 44
cores, 50
CPLEX, 18, 46, 48, 65, 73, 85
critical events, 33

d debug, 16, 76
d Decision Variables, 48, 51, 103, 106
dependability, 7, 27
d DFS, 15, 19, 28, 30, 31, 32, 37, 52
d DVFS, 37

e Embedded Real-Time Systems, 22
e embedded systems, 16, 17, 20, 24, 32, 33, 37
e energy consumption, 15, 17, 19, 20, 30, 31, 32, 52, 77, 103, 104, 105, 106, 108, 109
energy efficiency, 18, 19, 24, 106
energy saving, 33, 105, 108
energy-aware, 33
energy-efficiency, 7, 17, 20, 24, 30, 33, 104, 108
Energy-efficiency, 15, 17
e energy-savings, 17, 18, 22, 108, 124
Evaluation, 96
event, 18, 19, 24, 42, 64, 65, 73, 75, 76, 85, 88, 89, 106
execution time, 17, 29, 30, 32, 42, 52, 88, 92, 94

f fault events, 24
flexibility, 7
frequency scaling, 17, 24, 31
Functions, 68

g Gannt mapping, 76
global optimum, 73, 85
Graph mapping, 75, 89
heuristics, 16, 36
Heuristics, 46
Hop, 52

i IBM, 18, 35, 85
linearizing, 54

m mathematical programming, 16
MeS, 16, 45, 48, 63, 64, 69, 72, 73, 75, 76, 85, 86, 91, 92, 93, 94, 95, 96, 103, 104, 105, 106, 107, 108, 124
MES, 9, 20
Message Deadlines, 51
Message Duration, 50
Meta-Scheduler, 9, 63, 68, 70, 71, 72
meta-schedules, 7, 20, 29, 89, 91
meta-scheduling, 7
Meta-Scheduling, 1
MIQP, 19, 35, 46, 65, 73, 91, 104
Mixed-Criticality, 23
Mixed-Integer Quadratic Programming, 19
mobile phones, 15
motivation, 5
Motivation, 16
MPSoC, 9, 25, 31
multi-core architectures, 15, 17, 19, 24
multi-processor, 16
multi-scenario, 18, 63

n neighborhood search, 16
Neighborhood search, 46
NoC, 9, 17, 24, 25, 26, 29, 30, 31, 42, 50, 63, 77, 103, 108, 124
NoCs, 7, 16, 17, 20, 25, 26, 27, 30, 35, 42, 124
nodes, 18, 42, 75, 76, 86, 88, 91, 92, 93
NP-hard, 46

o objective function, 19, 35, 46
optimization, 18, 19, 37, 85
Optimization, 35, 46
optimizer, 35, 46, 73
optimizing, 22, 28
Outputs, 88, 105, 108, 112, 117

p Path, 50
Physical model, 42
platforms, 7, 124
PM, 42, 64, 69, 75, 85, 86, 91, 92, 104, 105, 106, 107, 108
problem-solving, 46
Prof. Dr. Madjid Fathi, 5
Prof. Dr. Raimund Kirner, 2, 5
Professor Dr. Roman Obermaisser, 5

q quadratic, 46
quasi-static, 17, 34, 36, 37, 63
quasi-static scheduling, 34, 37, 63

r real-time, 16, 22, 30, 40
Real-Time Systems, 22
List of Figures

Figure 1. Conceptual difference [23] .................................................................................. 22
Figure 2. Current and future trend in an embedded system [23] ........................................ 22
Figure 3. Kalray’s MPPA network-on-chip (The MPPA2® – 256 Bostan2 processor [52]) .................................................................................................................. 26
Figure 4. Simplified design flow and meta-scheduler (MeS) integration in SAFEPOWER [3] ...................................................................................................................... 27
Figure 5. A search tree for time slot [17] ........................................................................... 33
Figure 6. Conceptual physical model (PM) ........................................................................ 43
Figure 7. General physical model (PM) schema ............................................................... 43
Figure 8. General application model (AM) schema ........................................................ 44
Figure 9. General context model (CM) schema .............................................................. 45
Figure 10. Meta-scheduler (MeS) data structure schema model .................................. 46
Figure 11. The basic model of meta-scheduling (MeS) .................................................... 47
Figure 12. Conceptual AM .......................................................................................... 51
Figure 13. Quadratization technique via hops ............................................................... 55
Figure 14. The effect of TSDF on tasks et(t) and core fault ........................................... 56
Figure 15. Three-step scheduling for finding optimum solutions .................................... 57
Figure 16. a. Stack slack (SS) and b. Dynamic slack (DS) sample .................................. 59
Figure 17. Usage of dynamic slack (DS) to reduce makespan ....................................... 60
Figure 18. Usage of SDF regarding dynamic slack (DS) ............................................... 61
Figure 19. Usage of scenario-based meta-scheduling (SBMeS) on system-level design for SAFEPOWER multi-processor system-on-a-chip (MPSoC) [3] ......................... 64
Figure 20. Depth-first algorithm establishing schedule backwards with tabu-set for re-convergence (FAESB-TSR) ........................................................................ 66
Figure 21. Conceptual of the meta-scheduler (MeS) tool ............................................. 67
Figure 22. Scenario-based meta-scheduling (SBMeS) general model ............................. 70
Figure 23. States of 3 events and their effect of each SM ........................................... 72
Figure 24. Events’ effects in task scheduling ................................................................ 72
Figure 25. Schedule Gantt map ................................................................................... 76
Figure 26. Schedule tree include all data ....................................................................... 76
Figure 27. Meta-visualization of an event ..................................................................... 77
Figure 28. Events state in s schedule tree ..................................................................... 79
Figure 29. Schedules share points regarding task changes and Figure 28 ...................... 80
Figure 30. Decoding schedules from delta tree (DT) ................................................... 82
Figure 31. Delta tree (DT) data model ........................................................................... 82
Figure 32. Schema technique for standard data structure modelling ............................ 84
Figure 33. Overview of scheduling models ................................................................... 85
Figure 34. The simple architecture of meta-scheduling (MeS) ...................................... 85
Figure 35. Standardized input XML sample ................................................................. 87
Figure 36. Example a textual data with three nodes .................................................... 88
Figure 37. Static slack (SS) schedule model (SM) generated by meta-scheduling visualization tool (MeSViz) ................................................................. 92
Figure 38. Schedule tree with 94 schedules (created via meta-scheduling visualization tool (MeSViz) and GVEdi) .................................................. 93
Figure 39. Gantt map of schedule ID 44 .................................................. 94
Figure 40. Incorrect results and information found in the complex schedule .......................... 95
Figure 41. Meta-schedule Gantt map generated from meta-scheduling (MeS) class.................. 95
Figure 42. Schedule SM0 with static slack (SS) .......................................... 96
Figure 43. Schedule SM1 with dynamic slack (DS) ...................................... 97
Figure 44. Comparing two schedules SM0 (Figure 42) and SM1 (Figure 43) after slack .... 97
Figure 45. Graph output of node dependency ............................................. 98
Figure 46. Few changes from ID2 to ID3 .................................................. 99
Figure 47. Minimum changes from SM37 to SM38 regarding T3 slack .......... 99
Figure 48. Meta-visualization for Example 3 (schedules SM3 & SM4) .......... 101
Figure 49. Comparison of three different scenarios for memory saving with delta scheduling technique (DTS) .................................................. 103
Figure 50. The physical model (PM) of case study ...................................... 104
Figure 51. The application model (AM) of case study .................................. 104
Figure 52. The application model (AM) of the case study [6] ....................... 107
Figure 53. The physical model (PM) of the case study 7.3.2 ........................ 108
Figure 54. Energy consumption results for cores and routers FECs,m, FERs,m, FEC, avg,dyn, FER, avg, dyn ................................................. 109
Figure 55. FECs,m schedule models (SMs) results and average FEavg,dyn ...... 109
Figure 56. Total energy reduction results for cores ReFEC(SM) and routers ReFER(SM) and averageFEC, avg,dyn, FER, avg, dyn .................................................. 110
Figure 57. Total FE reduction results for schedules ReEsm and average ReFE ...... 110
Figure 58. The application model (AM) of the case study .......................... 111
Figure 59. The physical model (PM) of the case study ............................... 112
Figure 60. FECs,m, FERs,m results for cores and routers and average FEC, avg,dyn, FER, avg, dyn ................................................................. 113
Figure 61. FE(s,m) results and average FEavg,dyn ...................................... 113
Figure 62. FE results for cores ReFEC(SM) and routers ReFER(SM) compare to SM0 and average FEC, avg,dyn, FER, avg, dyn ............................................. 114
Figure 63. Total ReFE(s,m) in each schedule compare to SM0 and average FEavg,dyn .......................... 114
Figure 64. The physical model (PM) of the case study ................................ 115
Figure 65. The application model (AM) of case study .................................. 115
Figure 66. All nodes in a sample schedules tree .................................... 118
Figure 67. Real nodes in a sample schedules tree .................................... 119
Figure 68. Combination dynamic slack (DS) and core fault results in SM101 which generated by meta-scheduling visualization tool (MeSViz) ........... 119
Figure 69. The total number of generated schedules NSm for each scenario ... 120
Figure 70. Time of computation for scenarios ........................................ 120
Figure 71. FE results for each scenario SSMx ........................................ 121
Figure 72. Total ReFE(SMM) (percentage) results for each scenario of comparing dynamic schedules with a static schedule SMx.
List of Tables

Table 1. An overview of related works compared to scenario-based meta-scheduling (SBMeS) ................................................................. 39
Table 2. Overview of existing real-time visualizations tools [7] ......................... 40
Table 3. Overview of input table [90] ................................................................ 48
Table 4. Sample data calculated for Figure 14 .................................................. 56
Table 5. Sample raw input data before forming in XML format ......................... 84
Table 6. Difference between the designed scenarios ........................................ 91
Table 7. Results of dynamic slack (DS) schedules ........................................... 100
Table 8. Memory consumption and saving via delta scheduling technique (DST) 100
Table 9. Sample results of Example 2 ............................................................. 101
Table 10. Results of delta scheduling technique (DST) memory saving for Example 2 101
Table 11. Sample results for Example 3 .......................................................... 102
Table 12. Results of delta scheduling technique (DST) memory saving for Example 3 102
Table 13. Meta-scheduling (MeS) input constant ............................................. 104
Table 14. Results of delta scheduling technique (DST) memory saving .......... 105
Table 15. Some collected results for model example ........................................ 105
Table 16. Energy results for example .............................................................. 106
Table 17. Meta-scheduling (MeS) input constant devices [13] ......................... 107
Table 18. Energy results for Example 7.3.2 ................................................... 108
Table 19. General results for a model example ............................................... 108
Table 20. Results of delta scheduling technique (DST) memory saving .......... 110
Table 21. Meta-scheduling (MeS) input constant ........................................... 111
Table 22. General results of $FE$ for the example model .................................. 112
Table 23. Results of delta scheduling technique (DST) memory saving .......... 114
Table 24. The context model (CM) for application model (AM) and physical model (PM) scenarios ................................................................................. 116
Table 25. Meta-scheduling (MeS) input constant ........................................... 117
Table 26. Output results .................................................................................. 117
Table 27. Redundancy path routing effect ....................................................... 122
List of examples

7.3.1.1. Example 1. Sample scenario with four tasks and three messages .................. 96
7.3.1.3. Example 2. Sample scenario with seven tasks and five messages and $N_{ssm} = 128$ ........................................................................................................................................................................... 100
7.3.1.4. Example 3. Big sample with $N_{tsk} = 9$, $N_{msg} = 8$, and $N_{ssm} = 512$ schedules ........................................................................................................................................................................................................................................... 102
List of Algorithms

Algorithm 1. Main function algorithm ................................................................. 69
Algorithm 2. Fault recovery and scheduling technique ........................................ 71
Algorithm 3. DTS discover and calculating changes for messages (a) and tasks (b).... 81
Algorithm 4. XML parser ...................................................................................... 86
Algorithm 5. Slope Evaluation 1 ........................................................................... 95
Algorithm 6. Slope Evaluation 2 ........................................................................... 96
PUBLICATIONS AND PREVIOUS WORKS


I. REFERENCES


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


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


[126] L. Benini, “Platform and MPSoC Design,”


