

Tradeoff Analysis for Dependable Real-Time Embedded Systems during the Early Design Phases

Junhe Gan

DTU



Kongens Lyngby 2014
PhD-2014-330

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Building 303B, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253031, Fax +45 45881399
compute@compute.dtu.dk
www.compute.dtu.dk
PhD-2014-330

To My Family

Summary

Embedded systems are becoming increasingly complex and have tight competing constraints in terms of performance, cost, energy consumption, dependability, flexibility, security, etc. The objective of this thesis is to propose design methods and tools for supporting the tradeoff analysis of competing design objectives during the early design phases, which are characterized by uncertainties. We consider safety-critical real-time applications modeled as task graphs, to be implemented on distributed heterogeneous architectures consisting of processing elements (PEs), interconnected by a shared communication channel. Tasks are scheduled using fixed-priority preemptive scheduling, and we use non-preemptive scheduling for messages.

As a first step, we address the problem of function-to-task decomposition. In this context we have assumed that the application functionality is captured by a set of functional blocks, with different safety requirements. We propose a Genetic Algorithm-based metaheuristic to solve the function-to-task decomposition problem. Our algorithm also decides the mapping of tasks to the PEs of a distributed architecture and the reliability of each PE in the architecture, such that the safety and integrity constraints are satisfied, the schedulability of the real-time application is guaranteed and the overall development and product unit costs are minimized.

Next, we investigate tradeoffs between performance, energy and reliability. Addressing energy and reliability simultaneously is especially challenging, since lowering the voltage to reduce the energy consumption has been shown to increase the transient fault rate. We are interested to tolerate transient faults and we use task replication for recovery. We propose a Tabu Search-based approach, which decides the mapping of tasks to processing elements, as well as the processor voltage and frequency levels for executing each task, such that transient faults are tolerated, the real-time constraints of the application are satisfied, and the energy consumed is minimized.

In this thesis, we target the early design phases, when decisions have a high impact on the subsequent implementation choices. However, due to a lack of information, the early design phases are characterized by uncertainties, e.g., in the worst-case execution times (WCETs), in the functionality requirements, or in the hardware component costs. In this context, we select the hardware components for the architecture and derive a mapping of tasks in the application, such that the resulted implementation is both robust and flexible. The architecture also has a high chance to have its unit cost within the cost budget. Robust means that the application has a high chance of being schedulable, considering the WCET uncertainties, whereas a flexible mapping has a high chance to successfully accommodate future functionality changes. We propose a Genetic Algorithm-based approach to solve this optimization problem. The proposed tradeoff analysis methods have been evaluated using several synthetic and real-life benchmarks.

Summary (Danish)

Indlejrede systemer bliver stadig mere komplekse og har stramme, konkurrerende begrænsninger med hensyn til ydelse, pris, energiforbrug, pålidelighed, fleksibilitet, sikkerhed osv. Formålet med denne afhandling er at foreslå metoder og redskaber til at støtte en afvejningsanalyse af konkurrerende design mål i de tidlige design faser, som er karakteriseret af usikkerhed. Vi berører sikkerhedskritiske realtidsapplikationer modelleret som opgave grafer, der skal implementeres på distribuerede heterogene arkitekturer bestående af beregningsselementer (PE'er), sammenkoblet med en delt kommunikationskanal. Opgaver planlægges ved hjælp af fast prioritet afbrydende (preemptive) planlægning, og vi bruger ikke-afbrydende (non-preemptive) planlægning for meddelelser.

Som et første skridt, vi tager fat på problemet med funktion-til-opgave nedbrydning. I denne sammenhæng har vi antaget, at applikationens funktionalitet er beskrevet ved et sæt af funktionelle blokke, med forskellige sikkerhedskrav. Vi foreslår en metaheuristik baseret på en genetisk algoritme til at løse problemet med funktion-til-opgave nedbrydningen. Vores algoritme afgør også fordelingen af opgaver til PE'en i en distribueret arkitektur og pålideligheden af de enkelte PE'er i arkitekturen, således at sikkerhedskravene er opfyldt, skedulerbarheden af realtids-applikationen er garanteret og de overordnede udviklings og produkt omkostninger minimeres.

Dernæst undersøger vi afvejninger mellem ydeevne, energi og pålidelighed. Håndtering af energi og pålidelighed samtidig er særligt udfordrende, fordi at sænke spændingen til at reducere energiforbruget har vist sig at øge hyppigheden af midlertidige fejl. Vi er interesseret i at tolerere forbigående fejl og vi bruger opgave replikering til fejlhåndtering. Vi foreslår metoden "Tabu Search", som beslutter fordeling af opgaver til PE'er, samt processor spænding og frekvens niveauer for udførelse af hver enkelt

opgave således at: forbigående fejl tolereres, realtids kriterier i applikationen er opfyldt og energiforbruget minimeres.

I denne afhandling fokuserer vi på de tidlige design faser, hvor beslutninger har en stor indvirkning på de efterfølgende implementeringsvalg. Dog er de tidlige design faser karakteriseret ved et højt niveau af usikkerhed på grund af manglende information. F.eks. i de værst tænkelige eksekveringstider (WCET'er), i de funktionelle krav, eller i de hardware komponenten omkostninger. I denne sammenhæng, vi vælge de hardware komponenter til arkitektur og udleder en fordeling af opgaver i applikationen, således at den endelige implementeringen er både robust og fleksibel. Arkitekturen også har en høj chance for at få sin enhedsomkostninger inden omkostningerne budget. Robust betyder, at programmet har en høj chance for at være skedulerbar, taget WCET usikkerheder i betragtning, mens en fleksibel fordeling har en høj chance for succesfuldt at rumme fremtidige funktionelle ændringer. Vi foreslår en genetisk algoritme metodik til at løse dette optimeringsproblem. De foreslåede afvejnings-analyse metoder er blevet evalueret ved hjælp af flere syntetiske og real-life benchmarks.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science, the Technical University of Denmark in fulfillment of the requirements for acquiring the Ph.D. degree in computer engineering.

In this thesis, we propose methods and tools to support automatic design space exploration for the design of embedded systems in the early design phases. The thesis consists of an introductory chapter and three papers.

The work has been supervised by Associate Professor Paul Pop and co-supervised by Professor Jan Madsen.

甘君荷

Junhe Gan

Acknowledgements

I would like to express my sincere respect and heartfelt thanks towards Paul Pop and Jan Madsen, for their professional supervision. I feel it is my great honor to be their PhD student. During these years of my PhD, they were always there, providing me all the help I need.

My greatest appreciation to Paul Pop, for two things. Firstly, I appreciated he selected me to be his PhD student, which gave me a chance to improve my skills and knowledge, which has proved to be a very rewarding experience. Secondly, I appreciated his patience towards me during these years. I learned a lot from his feedback, and improved, not only the ability to get things done, but also the manner and enthusiasm of my work.

I would also like to express a big thank you to Jan Madsen and Flavius Gruian. They gave me a lot of inspiring advice and patient reviews on our joint papers.

I have to mention that, close to half of my PhD time has been done at the Royal Institute of Technology (KTH), Sweden. I would especially like to thank Axel Jantsch and Ingo Sander for providing me an office and all the other help, which enabled me to finish my PhD. I enjoyed so much the various conversations with them and the time at KTH.

During my Master study in Lund University and Carnegie Mellon University, I had been working under the guidance of Claus Führer and Fernando De la Torre. They encouraged me and provided inspiration, from where the idea of pursuing a PhD grew in me.

I would also like to thank all my friends and all the fellow students for their support and help in different aspects through the years when I was working in Sweden, USA and Denmark.

In particular, I want to give my special thanks to my husband, Hao Wang, for his understanding and support over the years. He always encourages me not to give up, when I feel discouraged.

Finally, I take this opportunity to express my deep gratitude to my parents, Yang Gan and Yiqiu Liao, for their endless love and support.

Junhe Gan

February 2014, Stockholm

Contents

Summary	i
Summary (Danish)	iii
Preface	v
Acknowledgements	vii
1 Introduction	1
1.1 Design Metrics	2
1.1.1 Energy Consumption	3
1.1.2 Cost	3
1.1.3 Performance and Predictability	4
1.1.4 Dependability	5
1.1.5 Robustness and Flexibility	7
1.2 Design of Embedded Systems	7
1.2.1 Systems Engineering Stages	8
1.2.2 Early Decisions	9
1.2.3 System-Level Design	10
1.2.4 Design Challenges	12
1.2.5 Design Space Exploration	14
1.3 Thesis Objective and Contribution	15
2 Paper A: Criticality-Aware Function-to-Task Allocation for Distributed Real-Time Embedded Systems	19
2.1 Introduction	20
2.1.1 Related Work	22
2.2 System Model	24
2.2.1 Application Model	24

2.2.2	Platform Model	29
2.3	Problem Formulation	32
2.3.1	Motivational Example	32
2.4	Optimization Strategy	35
2.4.1	Objective Functions	35
2.4.2	Genetic Algorithm	37
2.5	Evaluation	39
2.6	Conclusion	44
3	Paper B: Reliability-Aware Dynamic Energy Management for Fault-Tolerant Distributed Embedded Systems	45
3.1	Introduction	46
3.2	System Model	48
3.3	Reliability Model	51
3.3.1	Energy/Reliability Trade-off Model	53
3.4	Problem Formulation	55
3.4.1	Motivational Example	55
3.5	Offline Synthesis	59
3.6	Online Scheduling	63
3.7	Experimental Results	67
3.7.1	Offline Synthesis Evaluation	67
3.7.2	Online Scheduling Evaluation	69
3.8	Conclusions	71
4	Paper C: Design for Robustness and Flexibility of Real-time Distributed Applications during the Early Design Phases	73
4.1	Introduction	74
4.2	System Model	76
4.2.1	Early Design Stages	76
4.2.2	Modeling WCET Uncertainties	78
4.2.3	Modeling Functionality Uncertainties	79
4.3	Problem Formulation	80
4.3.1	Robustness and Flexibility	80
4.4	Schedulability Analysis	81
4.5	Motivational Example	84
4.6	Mapping Optimization	85
4.6.1	NSGA-II for Multiobjectives Optimization	86
4.6.2	Determining the Mapping of Future Scenarios	88
4.7	Experimental Results	89
4.8	Architecture Selection under Uncertainties	92
4.8.1	Modeling Cost Uncertainty in Architecture Selection	93
4.8.2	Architecture Selection Problem	94
4.8.3	Architecture Selection: Motivational Example	94
4.8.4	GA-based Approach for Architecture Selection	97

Contents

xi

4.9 Conclusion	99
A List of Abbreviations	101
B List of Notations	103
Bibliography	107

CHAPTER 1

Introduction

We are living in a time of great change, with rapid development of science and technology. High-tech products are greatly improving our quality of life and work efficiency. Most of the products we use today are controlled by digital computer systems. When talking about digital computers, we may immediately think of the general purpose computer systems, such as laptops, desktops or servers. However, the computers inside most of the devices we used are a special type of computer systems, called *embedded systems*. Over 98% of the microprocessors produced are used in embedded systems [EJ09].

Compared to the general purpose computer systems, embedded systems are usually used in a specialized application domain, performing a specific set of functions, repeatedly. Embedded systems are controlled and operated by a predetermined program, and thus the expected operations of the embedded system interacting to the external environment should have been considered and captured in the program.

It is difficult to give a precise definition of an embedded system. Their most important characteristic is that they are built for performing a specific set of functions, and thus are not general purpose computers. Another characteristic is that they are tightly constrained, i.e., embedded systems have requirements such as high performance, low energy consumption, low cost, small size and short time-to-market. Most of embedded systems should be also continually reactive and perform the functions in real-time, with high predictability.

A mobile phone is an example of an embedded system, which is designed and used for making telephone calls, sending messages, and some entertainment activities for the leisure time. With smartphones, the distinction between general purpose computers and embedded systems becomes less obvious.

Other examples of embedded systems are the automotive electronic functions inside a vehicle, such as Anti-lock Braking Systems (ABS), engine control, airbags. A modern high-end vehicle can have more than 100 microprocessors implementing various functionalities [Cha09].

Embedded systems are everywhere. People in their daily life and work use more than 30 embedded systems [EJ09], such as, cars, cameras, televisions, printers, traffic light systems, etc. Embedded systems can be large, for example, airplanes and ships, or can be very small, such as hearing aids or smart sensors. The growth rate in the number of embedded devices is more than 10% per year and it is estimated the number of embedded devices may reach 40 billion by 2020 [Res12]. According to statistics [Res12], the market of embedded systems was valued at 121 billion dollars in 2011, and this number is expected to grow by 6.8%, i.e., 194.27 billion dollars, in 2018.

The complexity of embedded systems is increasing very rapidly. For instance, more than 100 million object code instructions (totaling close to 1 Gbyte of software) are embedded in about 70 chips (electronic control units) in a currently new car [HS06, EJ09], in order to perform the required functions.

The size of embedded software is increasing with 10% to 20% per year, depending on the application area [EJ09]. Embedded software is more challenging to develop than traditional desktop software, e.g., Microsoft Word, because embedded systems have very tight constraints on the requirements that they have to fulfill. For example, the embedded software of controlling a car should react the operations from the driver quickly enough and provide a dependable service even in the worst-case scenarios.

Developing embedded software is expensive. According to [BCC⁺05], non safety-critical embedded software normally costs to develop between 15-30 dollars per line of code, while for highly critical applications, such as the space shuttle, the cost per line of code increases to 1,000 dollars.

1.1 Design Metrics

As mentioned, embedded systems have very tight constraints on the requirements that they have to fulfill. Those requirements are divided into functional requirements and non-functional requirements. The desired functionality depends on the particular ap-

plication that is implemented by the embedded system. Typical non-functional requirements can be expressed in terms of design metrics, such as, time-to-market, size, energy consumption, cost, performance, predictability, robustness, flexibility, and dependability attributes of reliability, safety, security, maintainability and availability. Time-to-market is the time needed to build a working version of the system. Size means the physical space required by the system. Other design metrics are considered in our thesis and introduced in this section. The formal definition of these design metrics is given in the included papers: Paper A, Paper B and Paper C.

1.1.1 Energy Consumption

Energy-efficiency is one of the most important metrics for embedded systems, since many are battery-powered mobile devices. For example, mobile phones have stringent low power requirements, e.g., about two Watts [Mar11], limited by the available battery technology, so the battery lasts for a couple of days. The development of battery technology is far from satisfactory, and does not support the need of rapidly increasing computational requirements and the complexity of embedded applications [Roa09].

Process technology is targeted towards increased density, to support higher operating frequency, resulting not only in higher power density but also in thermal problems. Energy-efficiency aims to keep the embedded systems meeting other requirements such as size, weight, and performance, at the same time saving the power dissipation as much as possible, which can keep the temperature of embedded devices under desirable levels, such that the processors are not burned and the battery-life is extended.

In our thesis, we are interested in reducing the energy consumption without negatively impacting reliability and schedulability. These two design metrics of reliability and schedulability and their tradeoffs will be introduced later. We measure the *energy consumption* of an embedded system in *Processing Elements (PEs)*, which are a major component of the system-level energy consumption. We use the power model from [BBL09], as it is able to capture a set of realistic assumptions, such as discrete operating modes of PEs, the energy and delay due to mode switches, and takes I/O operations into account. The detailed definition of energy consumption for a specific embedded system is presented in Section 3.2, Paper B.

1.1.2 Cost

Cost is always a key decision factor in most industries. The cost of a system could be measured in many ways, for example, as the sum of all component costs integrated in the system, or capturing the design, development and/or manufacturing costs. We

define the *Non-Recurring Engineering cost (NRE)* as the one-time monetary cost of designing the system, and we define the *unit cost* as the monetary cost of manufacturing each copy of the system, excluding NRE cost. The NRE cost is fixed once the design of an system has been done, regardless of how many units are going to be manufactured.

Jokob Axelsson [Axe06] proposed a cost model to estimate the *per-product cost*, i.e., the cost of each copy of the system up to delivery, for a distributed real-time embedded system. In our thesis, we have adapted the cost model from [Axe06], considering the cost of an architecture solution in Section 4.8, Paper C, and considering the total cost of a design alternative that includes the unit cost of hardware components and the development and certification costs of software tasks in Section 2.4.1, Paper A.

1.1.3 Performance and Predictability

Performance typically is captured by the execution time of computations and communications. Taking two mobile phones as an example, iPhone 5 has better performance than iPhone 4, since iPhone 5 can finish a same task much faster with its dual core running at 1,300 MHz, than the single core at 1,000 MHz of iPhone 4.

There are two main measures of performance, depending on what is of concern. One is the *latency* or *response time*, which is the duration of a task's execution. Another is the *throughput*, that is the number of tasks a system executes per unit time. For example, let us assume that, a camera takes 0.25 second to process an image. The latency of that camera is 0.25 seconds, while the throughput of that camera is 4 images per second [Vah06].

These measures of performance are not useful for real-time systems. A *real-time system* is an embedded system where the correctness of the result depends also on the time-instant when it has been produced [Kop11b]. In this context, other performance measures, related to predictability are more appropriate. *Predictability* is a key property of any real-time system that the timing requirements must be met. Before we discuss the timing requirements, some basic terms used in real-time systems are introduced.

There are many models of computation used to capture the functionality of an embedded system [ELLSV97]. In this thesis we have used the *task graph* model of computation, see the details in the attached papers. A *task* is defined as a sequence of instructions, and it is ready to be executed at any time after its *release time*. The *execution time* of the task may be different between task invocations, and this variability of task execution time is typically due to, for example, the variations in the input data of tasks or the speculative features of modern processors.

In real-time systems, to be able to provide guarantees, engineers use the *Worst-Case Execution Time* (WCET). The WCET is an analytical bound on the execution time of a task (the execution time is surely not larger than this WCET bound) and can be determined using tools such as *aiT* from AbsInt [FH04]. Note that a WCET is a bound, i.e., a task may never execute for such a long period of time, and we are interested having as low as possible values for WCET (i.e., less pessimistic). Similarly, we can define the *Best-Case Execution time* (BCET) of a task. The length of time between the release time of a task and the time it finished executing is named as *response time*. A task is required to complete before its *deadline*.

Real-time embedded systems can have two types of timing requirements: hard and soft. In *hard* real-time embedded systems, each task must be completed before its deadline. Otherwise it may lead to catastrophic results. For example, the airbag in a car must be inflated within 10-20 milliseconds (ms), after a collision is detected. Any delay might be too late to save the life of the passenger.

In contrast, *soft* real-time embedded systems accept occasional timing failures, which will not result in disastrous consequences, but can lead to a certain performance degradation. For example, it is likely you are not aware that the DVD player failed to decode some frames of the video source, when you watch a movie.

In our thesis work, we target on hard real-time embedded systems. The metrics used for predictability are defined in Section 2.4.1 of Papers A, Section 3.5 of Paper B and Section 4.4 of Paper C.

1.1.4 Dependability

Some of embedded systems, such as cars, airplanes, or medical equipment, nuclear power plants, are *safety-critical*, since any deviation from the specified functionality can have catastrophic consequences to the people or environment. *Fault-tolerance* means the safety device is still under the control and provides alternative functionality, although an internal fault occurred and was detected. Such faults might be permanent (e.g., damaged microcontrollers or communication links), transient (e.g., caused by electromagnetic interference), or intermittent (appear and disappear repeatedly). The transient faults are the most common [Con03], and their number is increasing due to the rising level of integration in semiconductors.

The *failure rate* of a system is the number of failures within a given period of time, which depends on the technology and age of the system, the voltage or physical shocks that the system suffers, and the ambient temperature that the system works on. The typical permanent fault rate has been reported [ZMM04] in the range of 10^{-8} to 10^{-6} faults per hour for a chip. However, the fault rate increases dramatically while the sys-

tem runs in some harsh environments. For example, for an orbiting satellite is reported that up to 35 errors are found in a 15-minutes interval [CMR92], which equals to 140 faults/hour.

Software faults (bugs) are permanent, i.e., they are due to specification, design or implementation mistakes. Software does not experience transient faults similar to hardware, since it is not aging, for example. A software bug disappears only if the software is updated with a new version, where the bug has been removed. We have considered to tolerate hardware transient faults which manifest themselves at the task-level.

Dependability is the system property that integrates such attributes as reliability, availability, safety, security, maintainability and availability [ALR⁺01]. *Reliability* is defined as the probability of successful execution over a period of time under a given set of operating conditions. We modeled the reliability for fault-tolerant distributed embedded systems in Section 3.3, Paper B.

Safety describes the property that a system will not endanger human life or the environment. A *Hazard* is a situation in which there is active or potential danger to people or the environment. *Risk* is a combination of the probability of a hazardous event and its consequence. If, after performing an initial hazard and risk analysis, a system is considered safety-critical, it has to be certified [KK10]. Certification is a “conformity of assessment” performed by a third party, e.g. an independent organization or a national authority, namely a “certification authority”.

The current certification practice is “standards-based” [Rus07], and requires that the product and the development processes fulfill the requirements and satisfy the objectives of a certain certification standard, depending on the application area. For example, [IEC10] is used in industrial applications, [ISO09] is for the automotive area, whereas [RTC92] refers to software for airborne systems.

During the engineering of a safety-critical system, the hazards are identified and their severity is analyzed, the risks are assessed and the appropriate risk control measures are introduced to reduce the risk to an acceptable level. A *Safety-Integrity Level* (SIL) is allocated to each safety function and captures the required level of risk reduction. SIL allocation is typically a manual process, which is done after performing hazard and risk analysis [Sto96], although a few researchers have proposed automatic approaches for SIL allocation [PWR⁺10b]. SILs differ slightly among areas. For example, the avionics area uses five “Design Assurance Levels” (DALs), from DAL E (lest critical) to DAL A (most critical), while ISO 26262 specifies for the automotive area four “Automotive Safety Integrity Levels” (ASILs), from ASIL A (least critical) to ASIL D (most critical). However, the approach presented in this thesis is applicable to all safety-critical areas, regardless of the standard. SILs are assigned to functional blocks, from SIL 4 (most critical) to SIL 0 (non-critical).

Security concerns the property that a system is able to protect the confidentiality, integrity, and availability of data and guarantee their authenticated communication. Certification standards require that tasks of different SILs are separated. In addition, they also impose constraints on the communication to ensure data integrity. These constraints are similar to the Bell-LaPadula [BL73] and Biba [Bib77] data integrity models from the security domain. In our thesis, we have modeled the safety and integrity requirements for criticality-aware functionality allocation and information communication in Section 2.2, Paper A.

1.1.5 Robustness and Flexibility

Design metrics are often difficult to quantify exactly due to, for example, uncertainties, as discussed about variability of task execution time in Section 1.1.3, hence there can be variations in their values. Also, the requirements of a system can change, especially in the early design phases. In addition, the environment where an embedded system operates will undergo changes. In this context, the issues of robustness and flexibility are of utmost importance.

Robustness is generally defined as the ability of a system to resist change without altering its implementation. In our thesis, robustness means that the application has a high chance of being schedulable, considering uncertainties (variations) in WCETs. For a formal definition, see Section 4.3.1, Paper C.

Flexibility is generally defined as the ability to adapt to change. Many things can change during the engineering of an embedded system, e.g., the initial requirements, the functionality. Also, new functions are always added in subsequent product versions. Performing changes is often very costly and requires extensive validation. For example, the time-to-market of a power-train unit in automotive industry is 24-months. Five months out of the time is used for the validation, which the percentage of validation time is more than 20% of its time-to-market [PEPP04].

In our thesis work, flexibility is defined as the likelihood of successfully implementing the future functionality changes, which have been modeled as “future scenarios” [BE06]. The detailed definition of flexibility is presented in Section 4.3.1, Paper C.

1.2 Design of Embedded Systems

The design of embedded systems is facing ever-increasing demands from the rapidly growing complexity of embedded systems and the competing constraints. The con-

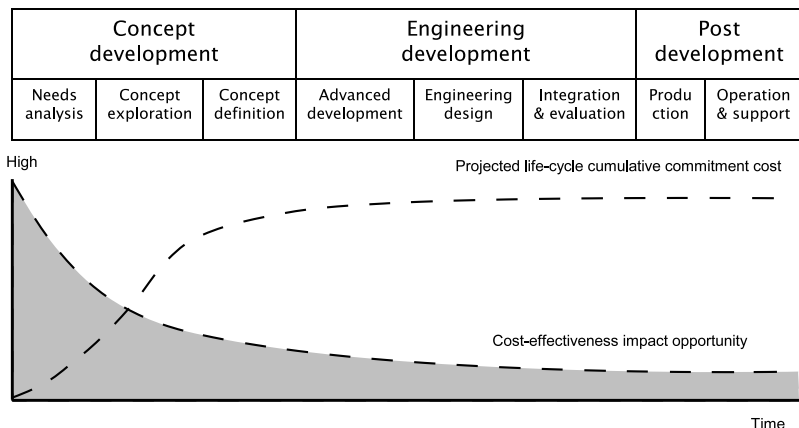


Figure 1.1: Life Cycle of Systems Engineering

straints that embedded systems should meet are measured by design metrics such as, energy consumption, cost, predictability, robustness, flexibility, reliability, which are described in the previous section,. The design methodology used in many organizations follows some versions of the “waterfall” model [Est07]. Such a methodology is inadequate for complex systems, and many other methodologies have been proposed, such as the V-model of system development [Est07].

In the following subsections, firstly we introduce the different stages of systems engineering life cycle, and then present the system-level design methodology. We continue to talk about the main challenges in designing embedded systems, which is to determine an implementation that simultaneously optimizes the competing design metrics. In the end, methods and tool for automatic design space exploration are also discussed.

1.2.1 Systems Engineering Stages

There are many life cycle models used in the industry [Est07], such as waterfall models [Boe88] and V-models [FHKS09], but in principle they all have the following main stages: concept development, engineering development and post-development, see the top part of Fig. 1.1 [KSSB11]. During concept development, system engineers perform a needs analysis, do concept exploration and definition. Engineering development consists of advanced development, engineering design, integration and evaluation. Then, in post-development, the mass-production from the prototype starts, followed by the operation and maintenance of the system. In this thesis, we focus on the early design stages, see the following subsection.

1.2.2 Early Decisions

Experience from completed system design and development projects has indicated that, the cost spent for the concept development stage accounts for about 20% of the total cost. However, 80% of the cumulative cost of the system is committed already in this stage [Bue11]. Decisions made in the early design stages not only have a high impact on the subsequent implementation choices, but also have substantially negative impacts on the total cost of the system, since it is costly and time-consuming to modify and correct a decided design to other alternative designs in the engineering development stage. i.e., early decisions have a high cost-effectiveness impact opportunity.

For example [Tas02], in the context of testing the software of a system, reports that, 70% of faults are introduced in the concept development stage, while 30% are found in the engineering development stage. However, the cost for removing faults in the engineering development stage is much more expensive (5-10 times) than the case we fix them in the early design stages of concept development.

The bottom part Fig. 1.1 [SR11] shows the projected committed cumulative cost, and the impact opportunity of a design decision over time. The conclusion is that we should spend more effort on the early decisions, in order to increase the chance of completing the projects on time and successfully.

There is a lot of research on embedded systems design [Mar11, GAGS09], but very few researchers have addressed the early design stages. The challenge is that early design stages are characterized by many uncertainties.

Uncertainty in the context of systems engineering refers to the inability to determine precisely the state or attributes of a system. It can be caused by incomplete knowledge or by stochastic variability. A detailed discussion on the taxonomy of uncertainty is available in [Hai11].

Uncertainties in the early design stages need to be quantified, such that *risks*, defined here as the probability of not reaching design targets, of different design alternatives can be estimated for making early design decisions. In Paper C, we have modeled the uncertainties of task WCETs (Section 4.2.2), functionality requirements (Section 4.2.3), and hardware component costs (Section 4.8.1) in the early design stages, and considered them in evaluating the design metrics of robustness, flexibility and cost.

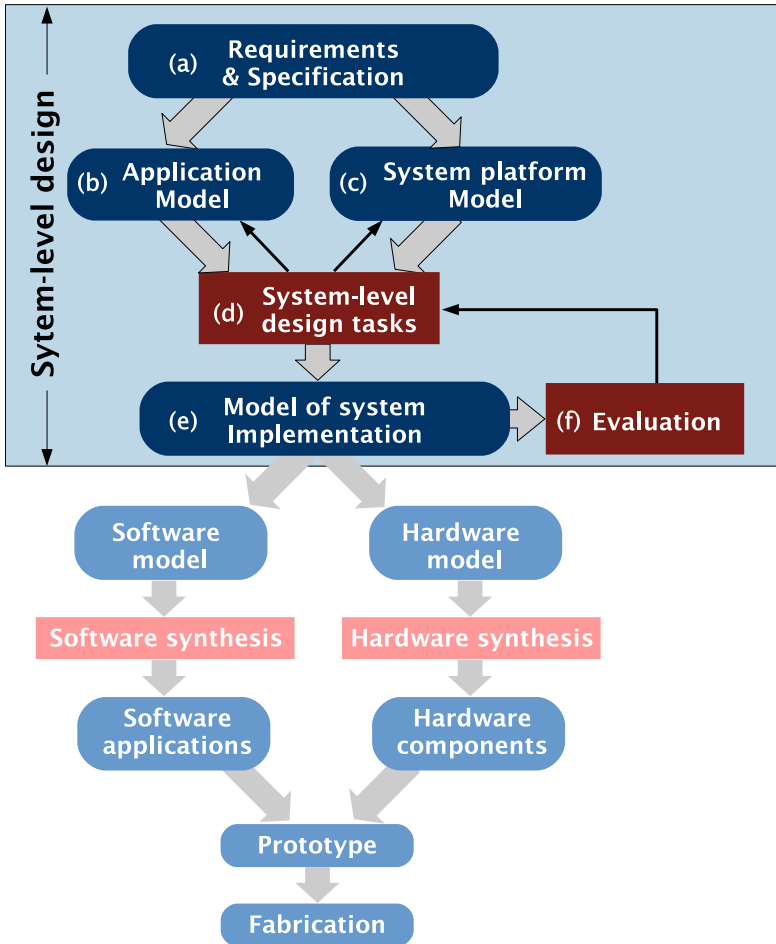


Figure 1.2: System-level Design

1.2.3 System-Level Design

The aim of any design methodology is to minimize the time-to-market, and coordinate the design tasks such that design metrics are simultaneously optimized. In the embedded systems area, the design methodology is typically organized as in the Fig. 1.2. We use boxes with rounded corners for input and output information, and rectangles for design tasks. Our thesis work includes the steps which are inside the blue box. The low-level development, i.e., a full synthesis of the design solution, whose steps are under the blue box of Fig. 1.2, is outside the scope of our thesis work. In the following, we explain the system-level design process (a) - (f).

(a) Besides demanding requirements on functionality, embedded systems have to meet also non-functional requirements, which are captured by the “design metrics” presented in Section 1.1.

(b) The applications are typically modeled with formalism from the particular domain. For example, the functionality of a vehicle can be described in terms of control algorithms using differential equations to model the behavior of the vehicle and its environment. Researchers have used many Models of Computation (MoC) [ELLSV97], such as *task graphs* or *synchronous data flow networks* for the modeling of embedded systems. In our thesis work, the detailed application models are presented in Sections 2.2.1 of Paper A, Sections 3.2 of Paper B and Sections 4.2 of Paper C.

(c) In our thesis work, the system platform is modeled at system level, which is viewed as a set of heterogeneous processing elements interconnected by a shared communication channel. We consider two cases for determining the system platform. One is given a fixed system platform that no modification is allowed, such a case is considered in Section 3.2, Paper B. Another is considering a system platform that can be parametrized, e.g., the number of components, the type of components, and the performance of components. We consider parametrized architectures in Section 2.2.2, Paper A and Section 4.8, Paper C.

(d) Once the application functionality and system platform have been modeled, several system-level design tasks are performed. The design tasks are typically performed automatically with the help of tools.

In our thesis work, we have considered the following system-level design tasks.

- *Function-to-task allocation* determines the decomposition options for implementing functions to tasks. In Fig. 1.2(b), an application is composed of several functionalities with safety requirements. In the early design stages, such functionalities are captured using functional blocks of different SILs. At the implementation level, the functional blocks from the design level have to be transformed into software or hardware tasks, or a combination of both. The certification standards allow several decomposition options. The function-to-task allocation problem is addressed in Paper A.
- *Architecture selection* determines the system platform. In Fig. 1.2(c), when the given system platform can be parametrized (in Paper A and Paper C), we may decide the number of components interconnected in the system platform, and choose the components from different types, performance and other features to build the system platform.
- *Mapping* decides the assignment of tasks to PEs and of messages to buses. The mapping problem has been tackled in all our papers included in the thesis.

- *Voltage scaling* refers to the assignment of processor operating modes (consisting of voltage/frequency pairs) to each mapped task. We consider each PE is characterized by a set of operating modes. Voltage scaling is the focus of Paper B, together with mapping.
- *Scheduling* decides the execution order of the mapped tasks on the PE or messages on the bus. In all our papers included in the thesis, tasks are scheduled using a fixed-priority preemptive scheduling, and messages are scheduled using fixed-priority non-preemptive scheduling.

(e) the outcome of Fig. 1.2(d) box is a “model of system implementation”. A huge number of alternative implementation solutions will be visited during the design space explorations. Each such alternative implementation is evaluated in terms of the design metrics, captured in (f) box of Fig. 1.2.

(f) The evaluation can be done analytically or using simulation. In our thesis work, we used analytical models for evaluating a wide range of design metrics, and those models are presented in each included paper in details.

We can perform the modeling, design and evaluation of embedded systems at several abstraction levels [GK83]. From the lowest level to the highest level, there are circuit level, logic level, register-transfer level and system level. System-level design, by means of highly abstraction of modeling, is able to investigate the many alternative implementations of the system, and allows fast evaluation of each alternative, such that the exploration of a huge design space is possible.

1.2.4 Design Challenges

It is challenging to design embedded systems with complex functionality and simultaneously optimize multiple design metrics, especially when the design metrics compete with one another. Approaches used for improving a certain design metric can worsen another one. Tradeoffs must be analyzed in order to determine the best solutions that meet such an optimization challenge. In our thesis work, we have measured six design metrics, i.e., energy consumption, cost, schedulability, robustness, flexibility and reliability. Their tradeoffs are listed as follows.

- Tradeoffs between schedulability and cost, within the safety requirements.

The system-wide safety requirement is maintained if each component has been developed and operated under the given safety target. It is a common practice to decompose the system-wide safety requirements and allocate per-component

safety requirement to hardware and software components, separately. However, the decomposition of function-to-task with safety requirements are not trivial. For each function with a given criticality level, there are several decomposition options, with varying impact on total cost (including the development cost of software tasks and the unit cost of hardware components) and schedulability of the implementation.

In Paper A, we have considered mixed-criticality applications to be implemented on distributed architectures. We have proposed a GA-based approach to decide the function-to-task allocation, the mapping of tasks to PEs and the reliability of each PE, such that the safety requirements are preserved, the development and unit costs are minimized, and the schedulability is maximized.

- Tradeoffs between energy consumption and schedulability, within the reliability requirements.

The most common approach for energy minimization that allows energy and performance trade-offs during run-time of the application is *Dynamic Voltage Scaling (DVS)* [SAHE03]. DVS aims at reducing the dynamic power consumption by scaling down operational frequency and circuit supply voltage, while adapting the component (PE or communication link) performance to the actual requirement of the system. A considerable amount of work has been done on DVS, see [SAHE03] for a survey.

However, lowering the operating voltage and frequency not only decreases the performance, but also increases the number of transient faults exponentially. The main reason for such an increase is that, with lower voltages, even very low energy particles are likely to create a critical charge that leads to a transient fault. Moreover, the redundancy-based fault-tolerance techniques (such as replication) and DVS-based low-power techniques compete for the available slack.

In Paper B, we addressed the mapping, voltage and frequency scaling for fault-tolerant hard real-time applications mapped on distributed embedded systems. We captured the effect of voltage and frequency scaling on system reliability, and we showed that if the supply voltage and the operating frequency are lowered to reduce energy consumption, reliability is significantly reduced. We have prepared an offline synthesis approach, based on a Tabu Search metaheuristic, which decides the mapping and operating mode for each task such that the energy is reduced and the schedulability and reliability constraints are satisfied. We have also proposed an online scheduling approach, which considers the mapping determined by offline synthesis approach and decides at runtime the operating mode for each job such that the energy is further reduced, while guaranteeing the timing and reliability constraints.

- Tradeoffs among robustness, flexibility, and cost.

In the early design stages of building a new system version, the choice of reusing legacy architecture components, using or upgrading to new components has a

significant impact on the robustness, flexibility and architecture cost of the new system version. In the case we choose to use new hardware components, with improved metrics such as better performance, or lower power dissipation, we may need to redevelop and validate the system platform which results in high costs for this new architecture solution and high uncertainties in evaluating the WCET of tasks. In case we migrate the legacy hardware components from the previous products, the cost of such an architecture solution should be much less than that in the former case, and the task WCET are more certain as well, but we may not benefit from the improved metrics of the new architecture.

In Paper C, we have addressed the architecture selection and the mapping of hard real-time applications on distributed heterogeneous architectures, during the early design phases. We have proposed a GA-based optimization for architecture selection and task mapping, targeting robustness, flexibility, and cost, which takes into account the uncertainties in WCETs, functionality requirements, and hardware component costs respectively. The proposed model allows the system designer to make the early design decisions after considering the tradeoff among robustness, flexibility, and cost.

1.2.5 Design Space Exploration

Design space exploration (DSE) is the process of visiting and analyzing the design alternatives to identify good quality solutions. The search in design space is presented in the three arrows which repeatedly visits new design alternatives in (b), (c) and (d) in Fig. 1.2. The design alternative are created by performing design tasks presented in Section 1.2.3.

DSE can be done manually, which is often the case in practice, but this is very inefficient for designing complex systems. In our thesis, we do automatic DSE supported by tools, and the search in the automatic DSE tool is implemented using optimization algorithms, in order to determine the good quality solutions in a reasonable time.

We consider m -dimensional space X of possible design alternatives, and F is the n -dimensional space of values for each objectives, i.e., design metrics.

$$F(x) = opt(f_1(x), \dots, f_n(x)) \quad (1.1)$$

where $x \in X$, opt could be minimize or maximize the objectives, e.g., minimizing $f_1(x)$ and maximize $f_2(x), \dots, f_n(x)$.

A design solution $x \in X$ is called *Pareto-optimal* with respect to X iff there is no design $y \in X$ such that $u = F(x)$ is dominated by $v = F(y)$.

In our thesis work, we consider simultaneously optimizing multiple competing design metrics. In Paper B, we merge all design metrics into a single objective function by using a weighted average, and determine one optimized solution, while in Paper A and Paper C, we perform multiobjective optimization, which aims to determine Pareto-front of solutions.

In general, there are exact methods, such as integer/mixed linear program formulation, and heuristics, such as Tabu Search (TS) or Genetic Algorithm (GA), for solving the multiobjective optimization problems. In our thesis work, the optimization problems we addressed (in Paper A, Paper B and Paper C) are NP-hard. In order to find high quality solutions in a reasonable time, we proposed a TS-based heuristics in Paper B, and used a GA-based heuristics for multiobjective optimization in Paper A and Paper C.

1.3 Thesis Objective and Contribution

The objective of this thesis is to propose design methods and tools for the design of embedded systems. We are interested in addressing competing design metrics such as energy consumption, cost, schedulability, reliability, robustness and flexibility, and to support the designer making early design decisions during the early design phases, which are characterized by uncertainties. The proposed methods have been implemented in optimization tools which use advanced optimization techniques to determine good quality solutions in a reasonable time.

The following papers and contributions have been included in the thesis:

- **Paper A:** Gan, Junhe, Paul Pop, Domitian Tamas-Selicean and Jan Madsen. **“Criticality-Aware Functionality Allocation for Distributed Real-Time Embedded Systems.”** To be submitted to International Journal of Reliability, Quality and Safety Engineering (IJRQSE). A preliminary version of this paper has been accepted to the Design, Automation and Test in Europe (DATE) conference workshop: Performance, Power and Predictability of Many-Core Embedded Systems (3PMCES), 2014.

Gan, Junhe, Paul Pop and Jan Madsen. **“Criticality-Aware Functionality Allocation for Distributed Multicore Real-Time Systems.”** In the Design, Automation and Test in Europe (DATE) conference workshop: Performance, Power and Predictability of Many-Core Embedded Systems (3PMCES), 2014.

In this paper, we are interested in implementing mixed-criticality hard real-time applications on a distributed heterogeneous hardware architecture, consisting of a set of processing elements (PEs) interconnected by a shared bus. We assume that the architecture provides the required separation mechanisms for mixed-criticality applications. An application is modeled as a set of functional blocks, with different Safety-Integrity levels (SILs), which dictate the development processes and certification procedures that have to be followed. Before the applications are implemented, the functional blocks have to be decomposed into software tasks. There are several decomposition options, which use redundancy and diversity to achieve the desired SIL. We are interested to determine (1) the function-to-task allocation, (2) the mapping of tasks to PEs, and (3) the reliability of each PE, such that the total cost is minimized, the application is schedulable and the safety and integrity constraints are satisfied. The proposed algorithm has been evaluated using a synthetic benchmark and a real-life benchmark.

- **Paper B:** Gan, Junhe, Paul Pop, Flavius Gruian and Jan Madsen. **“Reliability-Aware Dynamic Energy Management for Fault-Tolerant Distributed Embedded Systems.”** To be submitted to ACM Transactions on Design Automation of Electronic Systems (TODAES). A preliminary version of this paper has been published at the Asia and South Pacific Design Automation Conference (ASP-DAC), 2011.

Junhe Gan, Flavius Gruian, Paul Pop and Jan Madsen. **“Energy/Reliability Trade-Offs in Fault-Tolerant Event-Triggered Distributed Embedded Systems”** In Proceedings of the 16th Asia and South Pacific Design Automation Conference, pp. 731 – 736. IEEE Press, 2011.

This paper presents an approach to the synthesis of low-power fault-tolerant hard real-time applications mapped on distributed heterogeneous embedded systems. We first propose a design-time (offline) synthesis approach which decides the mapping of tasks to processing elements, as well as the voltage and frequency levels for executing each task, such that transient faults are tolerated, the timing constraints of the application are satisfied, and the energy consumed is minimized. Tasks are scheduled using fixed-priority preemptive scheduling, while replication is used for recovery from multiple transient faults. Addressing energy and reliability simultaneously is especially challenging, since lowering the voltage to reduce the energy consumption has been shown to increase the transient fault rate. We present a Tabu Search-based approach which uses an energy/reliability trade-off model to find reliable and schedulable implementations which minimizes the energy consumption. We also propose a runtime (online) synthesis algorithm, which changes dynamically the voltage and frequency levels of running tasks to reduce further the energy consumption, while guaranteeing the schedulability of the application and its fault-tolerance to transient faults. To provide such guarantees, the offline synthesis has to assume the worst-case, i.e., that tasks will execute up to their worst-case execution times, and the maximum faults will occur. The online scheduling will know at runtime the actual

execution times of tasks and the fault occurrences. We evaluated the proposed synthesis approaches using several synthetic and real-life benchmarks.

- **Paper C:** Gan, Junhe, Paul Pop and Jan Madsen. “**Design for Robustness and Flexibility of Real-time Distributed Applications during the Early Design Phases.**” To be submitted to Journal of Systems and Software (JSS). A preliminary version of this paper has been published at the Design, Automation and Test in Europe (DATE) conference, 2012.

Gan, Junhe, Paul Pop, Flavius Gruian and Jan Madsen. **Robust and Flexible Mapping for Real-Time Distributed Applications during the Early Design Phases.** In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 935 – 940. EDA Consortium, 2012.

We are interested in mapping hard real-time applications on distributed heterogeneous architectures. An application is modeled as a set of tasks, and we consider a fixed-priority preemptive scheduling policy. We target the early design phases, when decisions have a high impact on the subsequent implementation choices. However, due to a lack of information, the early design phases are characterized by uncertainties, e.g., in the Worst-Case Execution Times (WCETs) or in the functionality requirements. We model uncertainties in the WCETs using the “percentile method”. The uncertainties in the functionality requirements are captured using “future scenarios”, which are task sets that model functionality likely to be added in the future. In this context, we derive a mapping of tasks in the application, such that the resulted implementation is both robust and flexible. Robust means that the application has a high chance of being schedulable, considering the WCET uncertainties, whereas a flexible mapping has a high chance to successfully accommodate the future scenarios. We propose a Genetic Algorithm-based approach to solve this optimization problem. We also show how this problem can be extended to consider the architecture selection: deciding what hardware components to use in the architecture. In this context, we consider the uncertainties related to hardware component costs. Extensive experiments show the importance of taking into account the uncertainties during the early design phases.

CHAPTER 2

Paper A: Criticality-Aware Function-to-Task Allocation for Distributed Real-Time Embedded Systems

In this paper, we are interested in implementing mixed-criticality hard real-time applications on a distributed heterogeneous hardware architecture, consisting of a set of processing elements (PEs) interconnected by a shared bus. We assume that the architecture provides the required separation mechanisms for mixed-criticality applications. An application is modeled as a set of functional blocks, with different Safety-Integrity levels (SILs), which dictate the development processes and certification procedures that have to be followed. Before the applications are implemented, the functional blocks have to be decomposed into software tasks. There are several decomposition options, which use redundancy and diversity to achieve the desired SIL. We are interested to determine (1) the function-to-task allocation, (2) the mapping of tasks to PEs, and (3) the reliability of each PE, such that the total cost is minimized, the application is schedulable and the safety and integrity constraints are satisfied. The proposed algorithm has been evaluated using a synthetic benchmark and a real-life benchmark.

2.1 Introduction

Safety is a property of a system that will not endanger human life or the environment. *Safety-Integrity Levels* (SILs) are assigned to safety-related functions to capture the required level of risk reduction, and will dictate the development processes and certification procedures that have to be followed [IEC10], [ISO09], [RTC92]. There are four SIL levels, ranging from SIL 4 (most critical) to SIL 1 (least critical). Certification standards require that safety functions of different criticality levels are *protected* (or, *isolated*), so they cannot influence each other. For example, without protection, a lower-criticality task could corrupt the memory of a higher-criticality task.

The “Research Agenda for Mixed-Criticality Systems” [BBB⁺09] defines a *mixed-criticality* system as “an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure computing platform”. Many such applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of several different types of hardware components (called *nodes*), interconnected in a network. Initially, each function was implemented in a separate node, which has led to a large increase in the number of nodes.

The current trends are towards “integrated architectures”, where several functions are integrated onto the same node. In this context, designers are relying on partitioning mechanisms at the platform level. For example, in the avionics area, the partitioned architecture is called “Integrated Modular Avionics” (IMA) [Ari97], and the platform-level separation mechanisms are provided by implementations of the ARINC 653 standard [Ari13]. ARINC 653 consists of hardware-mediated operating system-level *spatial* and *temporal* partitioning mechanisms [Rus99]. Similar platform-level separation mechanisms are available in other industries [Ern10, LSOH07, PTV⁺13].

In this paper we are interested in the implementation of mixed-criticality hard real-time applications on integrated distributed architectures. The functionality of such applications is captured in the early design stages using functional blocks of different SILs. Before the applications are implemented, the functional blocks have to be decomposed into software tasks. For example, in the automotive area [CGL⁺07], the functionality is captured at the “Vehicle” and “Analysis” levels using functional blocks. During the “Design” level these functions are decomposed into tasks, which are implemented on the target architecture in the “implementation” level.

SIL allocation of functional blocks is typically a manual process, which is done after performing hazard and risk analysis, but researchers have proposed automatic approaches for SIL allocation [PWR⁺10a]. However, no automatic approaches have been proposed for the function-to-task allocation.

The function-to-task allocation problem is not trivial, since, for each function with a given SIL k , there are several decomposition options. For example, for a SIL 3 function, the automotive certification standard ISO 26262 [ISO11] allows several possible decompositions: a SIL 3 task; a SIL 2 task and a SIL 1 task; further, the SIL 2 task from the previous decomposition can be further decomposed into two SIL 1 tasks. Decomposing a SIL 3 function into two or even more tasks of lower SILs may reduce the development and certification costs. Such decompositions rely on redundancy and software diversity to achieve the desired SIL level using lower SIL tasks.

The decomposition will have an impact on the total cost and the schedulability of the implementation. The development and certification costs of a software task grow exponentially with each SIL [IBM10]. We assume that for each SIL k we have a corresponding required reliability R_k of processing element (PE). That is, high-criticality tasks are only allowed to be mapped on a correspondingly high reliability PE. Our optimization approach decides the type of PEs to be used in the architecture, and high-reliability PEs are more expensive, which impacts the total cost. In addition, the function-to-task decomposition will affect schedulability, because it may introduce additional tasks to be scheduled and because it may require message validation for communication integrity, which introduces additional overheads.

We consider heterogeneous distributed platforms, consisting of several PEs interconnected using a broadcast bus. We assume that the platform provides both spatial and temporal partitioning, thus enforcing enough separation for the mixed-criticality applications. The separation among mixed-criticality tasks also affects the communication. For example, integrity models require that a task can only receive an input from a task of the same criticality level or higher than its own. In this paper we consider that ACROSS integrity model [Was14], which allows lower-criticality tasks to send messages to higher-criticality tasks if they pass first through a validation middleware. We assume that the platform provides such a validation middleware to be used for communication integrity.

Each partition can have its own scheduling policy. However, to simplify the discussion, in this paper, we assume that all applications are scheduled using Fixed-Priority Preemptive Scheduling (FPPS). Although we address hard real-time applications, (non-critical) soft real-time applications can also be handled using a technique such as the Constant Bandwidth Server [AB98], where the server is seen as a hard task providing a desired level of service to soft tasks.

We assume that the communication protocol has mechanisms to enforce partitioning at the bus level. For example, space partitioning is attained in SAFEbus [HD93] by mapping the messages to unique locations in the inter-module memory, protected by a memory-mapping hardware in the host, and temporal partitioning is achieved in TTP [Kop11a] by enforcing a Time-Division Multiple Access scheme. TTEthernet [AS11] offers spatial separation for mixed-criticality messages through the concept of vir-

tual links, and temporal separation, enforced through schedule tables for time-triggered messages and bandwidth allocation for rate constrained messages. Researchers have shown how realistic bus protocols such as TTP [PEP04], FlexRay [PPE⁺08] and TTEthernet [TSPS12b] can be taken into account during the design. However, in this paper we consider a simple bus where messages are transmitted using a fixed-priority non-preemptive policy.

Given a mixed-criticality application modeled as a set of safety functions to be implemented on such a distributed architecture, we are interested to determine the function-to-task decomposition, the type of PEs in the architecture and the mapping of tasks to the PEs, such that the total cost is minimized, the application is schedulable and the safety and integrity constraints are satisfied.

2.1.1 Related Work

There is a large amount of research on hard real-time systems [Kop11a, But97], including task mapping to heterogeneous architectures [BSB⁺01]. Researchers have also started to address the mapping problem in the context of mixed-criticality systems [TSP11a], where the problem is to decide the assignment of tasks to partitions and the mapping to PEs such that the applications are schedulable. The work in [TSP11a] can also decide to “elevate” a task, i.e., raise its SIL, if this is needed to improve the schedulability, and uses a SIL-related development cost model to support the reduction of the overall cost.

The problem of deciding the SIL of a safety function is typically a manual process, which is done after performing hazard and risk analysis. In this context, researchers have started to propose automatic approaches to this problem, which is called “SIL allocation” [PWR⁺10b]. A more broader view is taken by [SBK], which propose a method for the propagation, transformation and refinement of safety requirements in general.

Some of the work on SIL allocation addresses also SIL decomposition (which they call “SIL algebra”), but in the context of safety functions implemented using hardware architectures [PWA⁺13, APW⁺13, BDS11]. [PWA⁺13] have proposed a Genetic Algorithm for SIL decomposition and [APW⁺13] have proposed a Tabu Search metaheuristic. Both works aim at a SIL decomposition which reduces the development costs, and are interested in deriving a fault-tolerant architecture. The safety of the resulted architecture is evaluated using Fault-Tree Analysis.

Researchers have proposed [BDS11] a tool called DALculus for the automatic allocation of SILs (which in the avionics area are called “Design Assurance Levels” or DAL) such that the smallest DAL possible is allocated to a function in order to mini-

mize costs, while following the recommended practices in the avionics area. The rules from [ARP10] and the architecture constraints are formulated as a constraint satisfaction problem and solved with an existing solver.

However, when performing SIL allocation and decomposition at the level of safety function, special case has to be taken not to abuse the SIL concept, which is not a measure of safety, but a way to dictate the development process rigor, and which cannot be decomposed unless “sufficient independence” can be shown between the requested functionalities [WC12].

Other related works are in the context of the automatic transformation of structural models to runtime models. For example, [KWS03] have proposed an automatic transformation of structural models (components and their interaction) to runtime models (tasks that communicate via message), taking into account real-time constraints. Their method also does priority assignment and thread mapping. The transformation of component models to tasks is also addressed in [FÅS04]. Thus, real-time components which model the desired functionality at “design time” are transformed into tasks at the “runtime” level. The authors propose a Genetic Algorithm-based approach to solve this transformation problem. Their work is limited to single-processor systems.

[FDT05] have looked at transforming functional blocks at the level of the “functional architecture” to Timed Petri Nets (TPNs) at the “runtime platform” level, deciding at the same time the links between the functions and resources, which from the “operational architecture”. None of these works address safety-critical systems and the SIL decomposition.

We have proposed a Genetic Algorithm-based optimization approach, which decides the decomposition of functions to tasks, the mapping of tasks to PEs and the type of PEs to be used in the architecture, such that the costs are minimized, the safety and integrity constraints are satisfied and the schedulability of the applications is guaranteed.

The paper is organized as follows. The next section present the system models considered. We formulate the problem and illustrate a motivational example in Section 2.3. The proposed a Genetic Algorithm-based approach is presented in Section 2.4, and is evaluated using a synthetic and a real-life benchmarks in Section 2.5. We draw the conclusion in the last section.

2.2 System Model

2.2.1 Application Model

The set of all applications in the system is denoted with Γ . At the design level, we model an application as *functional blocks*. Functional blocks have input and output ports and communicate with each other with directional links, which connect the output port of a functional block to the input port of another functional block. We assume that there are no loops in this communication (if loops are present, they can be unrolled).

All the functional blocks in the system are modeled as a graph $\mathcal{G}(\mathcal{F}, \mathcal{E})$, where each node $F_i \in \mathcal{F}$ is a functional block, and an edge $e_{ij} \in \mathcal{E}$ from F_i to F_j denotes a data dependency between an output port of F_i and an input port of F_j . We use fixed-priority preemptive scheduling at the PE-level, hence we assume that we know the period t_i and deadline d_i of each functional block F_i . If dependent functional blocks have different periods, they are combined into a merged graph capturing all activations for the hyper-period (least common multiple of all periods).

As mentioned, a *safety-critical* system should not endanger human life or the environment. A *hazard* is a situation in which there is actual or potential danger to people or to the environment. *Risk* is a combination of the frequency or probability of a specified hazardous event, and its consequence. If, after performing an initial hazard and risk analysis, a system is deemed safety-related, it has to be certified [Sto96]. Certification is a “conformity of assessment” performed by a third party, e.g., an independent organization or a national authority, namely a “certification authority”.

The current certification practice is “standards-based” [Rus07], and requires that the product and the development processes fulfill the requirements and satisfy the objectives of a certain certification standard, depending on the application area. For example, [IEC10] is used in industrial applications, [ISO09] is for the automotive area, whereas [RTC92] refers to software for airborne systems.

During the engineering of a safety-critical system, the hazards are identified and their severity is analyzed, the risks are assessed and the appropriate risk control measures are introduced to reduce the risk to an acceptable level. A Safety-Integrity Level (SIL) is allocated to each safety function and captures the required level of risk reduction. SIL allocation is typically a manual process, which is done after performing hazard and risk analysis [Sto96], although a few researchers have proposed automatic approaches for SIL allocation [PWR⁺10b]. SILs differ slightly among areas. For example, the avionics area uses five “Design Assurance Levels” (DAL), from DAL E (lest critical) to DAL A (most critical), while ISO 26262 specifies for the automotive area four “Automotive Safety Integrity Levels” (ASIL), from ASIL A (least critical) to ASIL D (most

Table 2.1: ISO/DIS 26262 SIL decomposition schemes

SIL	Can be decomposed as
SIL 4	SIL 3 + SIL 1 or SIL 2 + SIL 2 or SIL 4
SIL 3	SIL 2 + SIL 1 or SIL 3
SIL 2	SIL 1 + SIL 1 or SIL 2
SIL 1	SIL 1

critical). However, the approach presented in this paper is applicable to all safety-critical areas, regardless of the standard. SILs are assigned to functional blocks, from SIL 4 (most critical) to SIL 0 (non-critical).

At the implementation level, applications are modeled as a set of interacting tasks. A *task* is a set of instructions which execute on a PE. Hence, the functional blocks from the design level have to be transformed into software or hardware tasks, or a combination of both. Let us consider a safety function of SIL k , to be implemented as software tasks. The certification standards allow several options. For example, the safety-function could be implemented as one task of SIL k or, using redundancy to increase dependability, as several redundant tasks of a lower SIL, e.g., SIL $k-1$.

Decomposing a safety function of a higher SIL into several redundant tasks of lower SILs can reduce the development and certification costs, and could be the right choice in a particular context. For software redundancy, the standards recommend the use diversity, i.e., different implementations of the same functionality. This is because a fault (bug) in a software task will lead to a correlated failure in all of the tasks sharing the same implementation, unless software diversity is used. Often, one of the redundant tasks will implement a simpler (and maybe less accurate) algorithm as alternative diverse implementation.

Certification standards refer to this process as “SIL decomposition” and provide recommendations on the possible decompositions. For example, ISO/DIS 26262¹, Part 9, Section 5, provides the guide shown in Table 2.1 for SIL decomposition. Such a decomposition guide amounts to a “SIL algebra” [PWA⁺13], i.e., the SIL of the safety function is the sum of the SILs of the redundant tasks.

In this paper we assume that the safety functions are implemented as software tasks running on a distributed architecture. Let us consider a tasks τ_A which has to fulfill a safety requirement of SIL 3. According to Table 2.1, we can decompose task τ_A into two redundant tasks, e.g., τ_B with SIL 2 and τ_C of SIL 1. Task τ_B can be further decomposed into two SIL 1 tasks.

¹As mentioned, ISO 26262 uses the concept of Automotive SIL, or ASIL. To simplify the discussion, we consider ASIL D to be SIL 4 and ASIL A to be SIL 1.

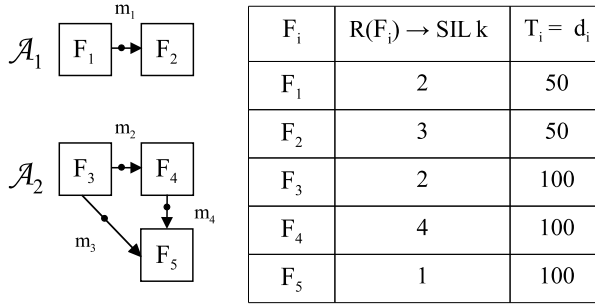


Figure 2.1: Example applications modeled as functional blocks

We assume that, for those tasks which are considered for decomposition, the designer will specify a library \mathcal{L}_A of possible decompositions based on the standard considered, similar to the library in Table 2.1. The library \mathcal{L}_A will specify for each function $F_i \in \mathcal{F}$ the possible decompositions into tasks. Each decomposition of a function F_i is modeled as a task graph $G_i(\mathcal{V}_i, \mathcal{E}_i)$ in \mathcal{L}_A .

Each node $\tau_j \in \mathcal{V}_i$ represents one task. We introduce the function R , which applies to a functional block or a task, to capture the safety requirements in terms of SIL 0 to SIL 4. The SILs of the functional blocks from Fig.2.1 and of the tasks in the decomposition library \mathcal{L}_A , resulted after decomposition, are presented in Fig.2.2.

The mapping of task to PE is denoted by the function $M : \Gamma \rightarrow \mathcal{N}$, where \mathcal{N} is the set of PEs in the architecture. This mapping is not yet known and will be decided by our approach. For each task τ_i we know the Worst-Case Execution Time (WCET) $C_i^{N_j}$ on each processing element N_j where τ_i is considered for mapping. The WCETs of the tasks are presented in Table 2.2. Note that in this small example, we assume PEs N_j have the same performance (speed) but with different SILs, so in Table 2.2 there is only one WCET for each task.

An edge $\varepsilon_{ij} \in \mathcal{E}_i$ from τ_i to τ_j indicates that the output of τ_i is the input of τ_j . A task becomes ready after all its inputs have arrived, and it issues its outputs when it terminates. Communication between tasks mapped to different PEs is performed by message passing over the bus. We assume that the message sizes s_{m_i} of each message m_i are known.

We define the decomposition function $D(F_i), D(F_i) : \mathcal{V}_i \rightarrow \mathcal{D}_i$, where \mathcal{D}_i is a set of decomposition options, specified in the decomposition library \mathcal{L}_A . There are three decomposition options for $D(F_4)$ in Fig. 2.2: D_4^1 in one task τ_4 (SIL 4), D_4^2 in two tasks, namely τ_8 (SIL 3) and τ_9 (SIL 1), and D_4^3 in three tasks τ_{10} of SIL 2, τ_{11} of SIL 1, and τ_9 of SIL 1.



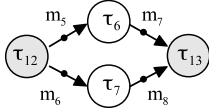


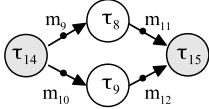
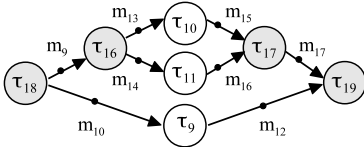

F_i	SIL Decomposition		Task graphs
F_1	D_1^1	SIL 2	
F_2	D_2^1	SIL 3	
F_2	D_2^2	SIL 2 + SIL 1	
F_3	D_3^1	SIL 2	
F_4	D_4^1	SIL 4	
F_4	D_4^2	SIL 3 + SIL 1	
F_4	D_4^3	SIL 2 + SIL 1 + SIL 1	
F_5	D_5^1	SIL 1	

Figure 2.2: Decomposition library \mathcal{L}_A for the applications in Fig.2.1

In the bottom task graph of Fig. 2.6, we show how F_4 , once decomposed as D_4^2 , is connected to the graph of application \mathcal{A}_2 from Fig. 2.1. We assume that a decomposed task will be connected to the original application graph via two “connecting” tasks (see the gray tasks in Fig. 2.2); one task which is distributing the input to the redundant decomposed tasks (e.g., τ_{14}) and one task which is collecting the outputs (e.g., τ_{15}). The SIL of the connecting tasks are given by the designer based on the communication constraints as discussed in Section 2.2.2 and on the requirements from the standards.

The WCETs for our example tasks are shown in Table 2.2. Note that in the decomposition option D_4^2 , the sum of the WCETs of the decomposed tasks, i.e., $\tau_8, \tau_9, \tau_{14}, \tau_{15}$, has a value of 44, compared to the value of 20 for the single task τ_4 in D_4^1 .

Table 2.2: Tasks in \mathcal{L}_A

Tasks	C_i	$t_i = d_i$	SIL	W_{τ_i}
τ_1	25	50	2	6
τ_2	25	50	3	16
τ_3	40	100	2	8
τ_4	20	100	4	28
τ_5	30	100	1	3
τ_6	15	50	2	5
τ_7	15	50	1	2
τ_8	20	100	3	15
τ_9	20	100	1	3
τ_{10}	20	100	2	6
τ_{11}	20	100	1	3
τ_{12}	2	50	2	1
τ_{13}	2	50	2	1
τ_{14}	2	100	3	2
τ_{15}	2	100	3	2
τ_{16}	2	100	2	1
τ_{17}	2	100	2	1
τ_{18}	2	100	2	1
τ_{19}	2	100	2	1

The SIL assigned to a task will dictate the development processes and certification procedures that have to be followed. Standards provide checklists of objectives required to be fulfilled for each SIL. Depending on the SIL, the standard may also impose that some objectives to be satisfied with independence, to ensure an unbiased evaluation and to avoid misinterpretation of the requirements [RTC92]. For example, for the verification process, independence is achieved by using tools and personnel other than those used throughout the development process.

SIL 0 functions are non-critical and do not impact the safety of the systems, thus are not covered by the standards. In the case of SIL 1, the processes are similar to those covered by quality management standards such as ISO 9001 [ISO08]. SIL 2 involves more reviewing and testing. SIL 3 is significantly more difficult, and requires “semi-formal” methods. SIL 4 often mandates formal methods, increasing further the development costs.

The assessment of conformity to the checklist of objectives has to be performed by independent assessors. For SIL 1 is enough to have an independent person, whereas for SIL 2 an independent department is required. In the case of SIL 3 and SIL 4, an independent organization has to be used. Moreover, the number of objectives that have

to be satisfied with independence is also growing. For example, in the case of DO-178B, the main difference between DAL A and DAL B is the number of objectives to be satisfied with independence: 25 out of 66 objectives are required for DAL A to be satisfied with independence, while for DAL B it is only 14 out of 66.

Software development cost estimation is a widely researched topic, and is beyond the scope of this paper. The reader is directed to [JS07, BAC00] for reviews on this topic. One of the most influential software cost models is the Constructive Cost Model (CO-COMO) [BMS00]. Researchers have shown how to take into account the development costs during the design process of embedded systems [DMG97].

The development of safety-critical systems is a highly structured and systematic process dictated by standards. These standards increase the development costs due to additional processes for software development and testing, qualification activities involved in compliance and increased process complexity, shown also by an IBM Rational study [IBM10]. Because of the systematic nature of the development processes dictated by the standards, we assume that the designer will be able to estimate the development effort required for a task. Hence, we define the development cost function W_{τ_i} to capture the cost to develop and certify a task τ_i to its required SIL. Table 2.2 shows an example of the development costs for each of the tasks in Fig. 2.2. Similarly, we define the development costs for the set of all the applications, $W(\Gamma)$, as the sum of the costs for each application. An example certification cost estimation in person-days for an Air Traffic Control radio platform is presented in [Roc09].

2.2.2 Platform Model

We consider hardware architectures consisting of a set \mathcal{N} of processing elements, interconnected by a broadcast communication channel, see Fig.2.3. In this paper, our focus is on the decomposition of safety function, so we consider a simple shared bus where the communication is performed according to a non-preemptive fixed-priority scheduling policy. We have shown in [TSPS12a] how a realistic communication protocol such as TTEthernet can be taken into account. We consider that the tasks are scheduled using Fixed-Priority Preemptive Scheduling (FPPS), but our work can be extended to consider also other scheduling policies, such as Static Cyclic Scheduling (SCS) [TSP11b].

Tasks of different SILs can share the same PE. To be able to host high-criticality tasks, a PE has to have a high reliability. We consider that for each PE in the architecture we have implementations with different reliabilities. The reliability of a PE can be increased through “hardening” or by using a more rigorous design [IPP⁺09]. We denote with \mathcal{L}_H the library of varying reliability PEs available, and with w_j^k the unit cost of PE N_j with a reliability corresponding to a SIL k . Note that we will always use in an

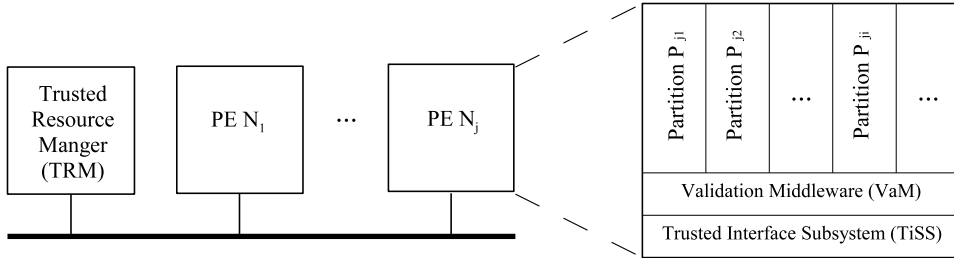


Figure 2.3: Example Platform

architecture the lowest cost PEs available, which have the required reliability to host the highest-criticality task mapped on that PE, and that the unit cost increases with increased reliability.

When several tasks of different SILs share the same processing element, the certification standards require that they are developed at the highest SIL among the SILs of the tasks, which is very expensive. Unless, the standards state, it can be shown that the implementation of the tasks is “sufficiently independent”, i.e., there is both spatial and temporal separation among the tasks. Hence, tasks of different SILs have to be protected from each other. Otherwise, for example, a lower-criticality task could corrupt the code or data area of a higher-criticality task [CAS01], or block the higher-criticality task from accessing the CPU, leading thus to a failure.

In this paper, we consider that the separation between tasks of different SILs, and from different applications, is achieved through a temporal- and space-partitioning scheme similar to Integrated Modular Avionics (IMA) [Rus99]. Note that partitioning schemes similar to IMA are available in several application areas [PTV⁺13], not only in the avionics area. Space partitioning uses mechanisms such as a Memory Management Unit (MMU) to ensure that, for example, applications running on different partitions cannot corrupt the memory for the other applications. Temporal partitioning ensures the access of each application to the CPU, according to the scheduling policy used at the PE-level. A detailed discussion about partitioning is available in [Rus99].

Fig. 2.3 shows the partitions $P_{j1}, P_{j2}, \dots, P_{ji}, \dots$ on the PE N_j . We consider that we have one partition set for each application on each PE where the tasks are mapped for execution, and we have one partition P_{ji} for each SIL, such that only tasks from the same application and the same SIL will share a partition. During scheduling, when performing context switching between tasks of different SILs we will also perform partition switching. Recent architecture [WESK10, AFOTH06] and partitioned operating systems [Pik, POK] have a very small overhead associated with partition switching. We take into account in the WCET of tasks the partition switching overhead.

The PEs of the architecture are connected to the network through a *Trusted Interface Subsystem* (TiSS). The communication mechanisms are implemented by a *Trusted Resource Manager* (TRM), see Fig. 2.3. The TiSSes and TRM offer all the services required for dependable communications in mixed-criticality systems. Such an infrastructure has been proposed in several architectures [WESK10, Kop11b], and offers spatial and temporal partitioning for messages. The reliability of the communication infrastructure is high enough to accommodate the highest-criticality.

As mentioned, certification standards require that tasks of different SILs are separated. In addition, they also impose constraints on the communication to ensure data integrity. These constraints are similar to the Bell-LaPadula [BL73] and Biba [Bib77] data integrity models from the security domain. In this paper, we consider that the platform supports the ACROSS Integrity Model [Was14], which is an adaptation of Totel's model intended to offer support for multiple levels of criticality [TBDP98]. Such mechanisms are available in many platforms, and are similar to multiple independent levels of security/safety architectures [AFOTH06]. Thus, the integrity model we used has the following rules:

1. One task is assigned to exactly one partition (in our case, a partition is considered be similar to the “component” concept used in [Was14]). Note that each partition is assigned an integrity level (SIL).
2. Communication is allowed only between tasks of the same criticality level, or from a higher-criticality tasks to a lower-criticality task.

These rules are very restrictive for communication and prevent higher-criticality tasks from cooperating with lower criticality tasks. However, in practice it is often necessary to have such a cooperation in order to implement cost-efficiently mixed-criticality applications (otherwise all lower-criticality tasks would have to be developed and certified at the highest integrity levels). In this paper we consider (as in [Was14] and [TBDP98]) that if a communication is required from a lower-criticality task to a higher-criticality task, we use a “Validation Middleware” (VaM) to mediate this communication.

A VaM (see Fig. 2.3) receives lower-integrity inputs and produces high-integrity outputs. This is achieved through using application-dependent fault-tolerance mechanisms, see [Was14], which proposes a VaM, for details. The VaM will introduce a non-negligible overhead in the communication. We take into account this overhead in our application model by adding it to the WCET of a high-criticality task receiving a message from a lower-criticality task. Thus, we have the third rule:

3. Communication from a lower-critical task to a higher-criticality task must pass through a VaM.

2.3 Problem Formulation

The problem we are addressing in this paper can be formulated as follows. As an input to our problem we have (i) the set of applications Γ , which consist of functional blocks and their SIL requirements, (ii) the library \mathcal{L}_A of function-to-task decompositions, which includes the WCET of tasks and their SIL-dependent development and certification costs, (iii) an architecture consisting of a set \mathcal{N} of PEs interconnected by a shared bus, and (iv) the library \mathcal{L}_H of architecture implementations for the PEs, with varying reliability and unit costs.

We are interested to determine an implementation \mathcal{S} , such that the total costs are minimized, the schedulability of the applications maximized, and the requirements on safety (the SILs of the safety functions) and integrity (the three rules used to preserve the integrity of communication) are satisfied.

Synthesizing an implementation \mathcal{S} means deciding on (1) the function-to-task decomposition \mathcal{D} , (2) the mapping of tasks to PEs M , and (3) the types of PEs H to use in the architecture.

2.3.1 Motivational Example

Let us consider the example applications from Fig. 2.1, where we have five functional blocks, F_1 to F_5 , with different SIL requirements, periods and deadlines. We use the decomposition library \mathcal{L}_A from Fig. 2.2, with the task properties from Table 2.2. We consider an architecture of three PEs, and the library \mathcal{L}_H of alternatives is presented in Table 2.3. We are interested to determine an implementation \mathcal{S} such that the total costs are minimized, the schedulability is maximized, and the safety and integrity requirements are satisfied.

In this example, for simplicity, we have not considered all possibilities of decomposition for \mathcal{L}_A . For instance, F_1 in Fig. 2.2 could be decomposed into two tasks with SIL 1,

Table 2.3: Alternatives for N_j in Library \mathcal{L}_H

PEs	Maximum SIL k	$H(N_j) \rightarrow N_j^k$	$w(N_j^k)$
N_1	1	N_1^1	10
N_2	2	N_2^2	20
N_3	3	N_3^3	40
N_4	4	N_4^4	60

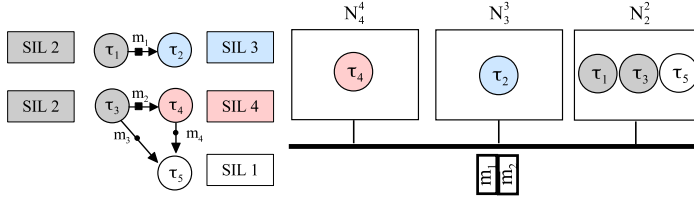


Figure 2.4: Straightforward Solution

based on the SIL decomposition schemes in Table 2.1. We assume that the engineer has decided only to perform decompositions in F_2 of SIL 3 and F_4 of SIL 4.

One possible solution to this problem is presented in Fig.2.4. We call this solution the “Straightforward Solution”, or “SFS”. This is a solution where: (1) we do not decompose the functional blocks into tasks with lower SILs, i.e., we use the decomposition options D_1^1 for F_1 , D_2^1 for F_2 , D_3^1 for F_3 , D_4^1 for F_4 , and D_5^1 for F_5 ; (2) we consider a simple mapping where we cluster all tasks based on SILs. Note that three PEs are considered in the current architecture, so tasks of SIL 4 are mapped on one PE with maximum SIL 4, tasks of SIL 3 are mapped on another PE with maximum SIL 3, and the left tasks of SIL 2 and SIL 1 are mapped on the third PE with maximum SIL 2.

Fig.2.4 shows the resulted tasks graphs after decomposition, the mapping of tasks on PEs, and which PE version from Table 2.3 we have used. For example, F_4 is decomposed into a single task τ_4 and is mapped on N_4^4 . As mentioned in Section 2.2.2, according to Rule 3, when a lower criticality task sends data to a higher criticality task, we have to use a VaM. In Fig.2.4, messages m_1 and m_2 , depicted by squares in the task graphs and bold boxes in the bus, are sent from τ_1 (SIL 2) to τ_2 (SIL 3), τ_3 (SIL 2) to τ_4 (SIL 4), respectively, have to use VaMs in the receiving components.

The SFS is also shown in Fig.2.5, using a “x”, where we display a graph with the resulting values of two objective functions, total cost (y-axis) and schedulability (x-axis). The exact definition of these two objective functions is given in Section 2.4.1. For the total cost (which includes both the development and certification costs of tasks and the unit costs of PEs), the smaller values means lower costs. For the schedulability, values above zero means “not schedulable”, whereas values below zero means that the solution is schedulable, and the smaller the value, the higher the schedulability. As we can see from Fig.2.5. SFS has a high cost and it is not schedulable. This is not surprising, considering that it is known that mapping decisions have a strong impact on solution quality, and SFS does not optimize the mapping.

Fig.2.5 also shows solutions that optimize the mapping. “Criticality-Aware Mapping Optimization” (CMO) uses the same simple approach for decomposition as SFS, but optimize the mapping aiming at improving the objective functions. CMO solutions,

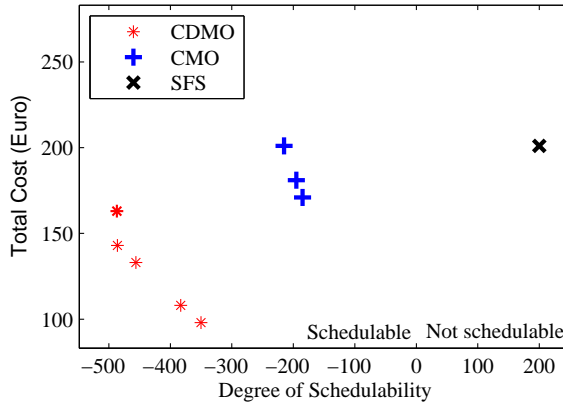


Figure 2.5: SFS and Optimized Results

determined after exhaustive search, are Pareto-optimal solutions. As we can see from Fig.2.5, where each CMO solution is depicted with a blue “+”, by optimizing the mapping, we are able find schedulable solutions. However, the cost remains high.

The focus of this paper is on the function-to-task decomposition. we have proposed an optimization approach called “Criticality-Aware Functional Decomposition and Mapping Optimization” (CDMO), which decides the functional decomposition, the mapping of tasks to PEs and the types of PEs such that our design objectives are optimized. CDMO solutions are found after exhaustive search as well. They are shown in Fig.2.5 using red “*”. As we can see from the figure, by carefully deciding on the function-to-task decomposition, we are able to significantly reduce the costs, and further improve the schedulability, compared with CMO which does not optimize the function-to-task decomposition.

Due to space reasons, we are not able to depict all Pareto-optimal solutions as we have done with SFS in Fig.2.4. However, we choose two extreme solutions found by CDMO: the best solution in terms of schedulability is shown in Fig.2.6, and the best solution in terms of costs is shown in Fig.2.7. In these figures, tasks with the same criticality (SIL) are marked by using the same color. Messages sent from a lower-SIL task to a higher-SIL task that have to go through VaMs have been highlighted by squares in the task graphs and bold boxes on the bus.

The problem presented in this section is NP-hard (determining task mapping itself is proven to be an NP-hard [TBW92]). To solve this multiobjective optimization problem, we have proposed an approach based on Genetic Algorithm, see the next section.

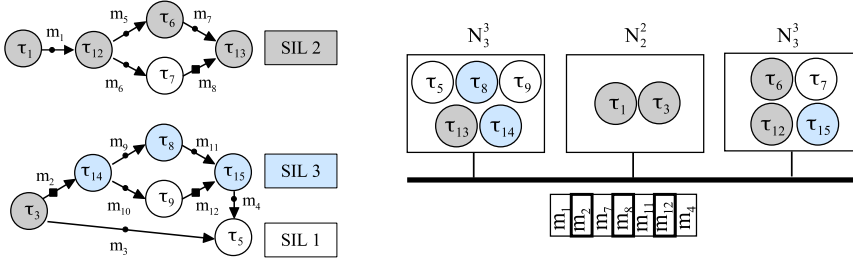


Figure 2.6: Best Solution in terms of Schedulability

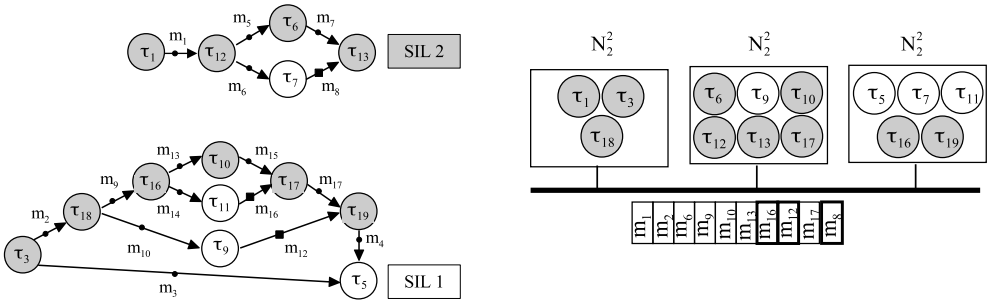


Figure 2.7: Best Solution in terms of Costs

2.4 Optimization Strategy

We propose a Genetic Algorithm (GA)-based approach, called CDMO (Criticality-aware functional Decomposition and task Mapping Optimization), to solve the optimization problem presented in Section 2.3. There are several off-the-shelf multi-objective GA implementations, such as NSGA-II [DAPM00] and search frameworks for multiobjective optimization such as PISA [BLTZ03]. We decided to use the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [DAPM00], due to its good performance and its simple implementation.

2.4.1 Objective Functions

We are interested to minimize the total cost and maximize schedulability. To determine the system schedulability we use response-time analysis to calculate the worst-case response time r_i of every task τ_i , which is compared to the deadline d_i . The basic analysis presented in [But1] has been extended over the years. For example, the state-of-the-art analysis in [PGH98] considers arbitrary arrival times and deadlines, offsets

and synchronous inter-task communication (where a receiving task has to wait for the input of the sender task).

Our focus in this paper is optimizing the design of a mixed-critical hard real-time system between schedulability and cost. Hence, we decide to use the basic analysis from [But11]. We do, however, take communication into account to make sure that the mapping solutions do not create too much bus traffic. We consider a non-preemptive fixed-priority bus and during the design space exploration we mark as infeasible solutions which result in a bus utilization over 100%. The utilization is calculated from the bus speed, the size of the messages exchanged over the bus and period of the sender tasks. The overhead C_o required by the validation middleware to increase the integrity of a message is added to the receiving task's WCET.

The schedulability of an implementation \mathcal{S} is captured using the "degree of schedulability", $r_{\mathcal{S}}$ over all tasks:

$$r_{\mathcal{S}} = \begin{cases} \ell_1 = \sum_i \max(0, r_i - d_i) & \text{if } \ell_1 > 0 \\ \ell_2 = \sum_i (r_i - d_i) & \text{if } \ell_1 = 0 \end{cases} \quad (2.1)$$

If the application is not schedulable, there exists at least one r_i greater than the deadline d_i , therefore ℓ_1 of the function will be positive. In this case $r_{\mathcal{S}}$ is equal to ℓ_1 . However, if the application is schedulable, then each r_i is smaller than the corresponding deadline d_i . In the case $\ell_1 = 0$ and we use ℓ_2 as the $r_{\mathcal{S}}$, as it is able to differentiate between two design alternatives, both leading to a schedulable implementation.

The second objective is the total cost that includes the unit cost of the PEs and the development and certification costs of software tasks. We define the total cost as:

$$w_{\mathcal{S}} = \sum_{N_j^k \in \mathcal{S}} w_{N_j^k} + \frac{\sum_{\tau_i \in \mathcal{S}} W_{\tau_i}}{\text{Units}} \quad (2.2)$$

where \mathcal{S} is the implementation solution currently evaluated. The first term is a summation of the unit cost of PEs, $w_{N_j^k}$, considering the selection N_j^k from the library \mathcal{L}_H , and the second term is the sum of the development costs W_{τ_i} the tasks τ_i included in \mathcal{S} according to the chosen decomposition function. Note that the software development costs W are divided by the number of estimated *Units* that are going to be produced.

2.4.2 Genetic Algorithm

GA is a metaheuristic optimization approach, which belongs to the class of Evolutionary Algorithms, inspired from the process of natural evolution. The set of candidate solutions is called a “population”, and each solution is (i) *encoded* using a string called a “chromosome”. The population is (ii) *initialized* to n candidate solutions, where n is the *population size*. The population is evolved by (iii) *selecting* a set of solutions and performing (iv) *recombination* and (v) *mutation* to generate offsprings. Finally, the parent population is (vi) *replaced* with an offspring population with better “fitness”. The fitness of a solution is evaluated using our two objectives. Steps (iii) to (vi) are repeated until a *termination condition* is reached.

Steps (i) to (vi) are explained in the reminder of this section. There are several choices for their implementation. Through experiments, we decided to choose the following approaches, which can find good solutions in a reasonable time. The parameters were also determined experimentally.

(i) *Encoding*: We encode a design alternative of function-to-task allocation and task mapping as a chromosome. A chromosome is composed of genes. Fig. 2.8 shows several chromosomes for the motivational example presented in Section 2.3.1. Each chromosome has two parts.

The first part contains a gene for each functional block in the application. In the motivational example, we have five genes for F_1 to F_5 . The value of the gene, j , represents the decomposition option D_i^j selected from the library \mathcal{L}_A for the respective F_i . Taking the 4th gene of chromosome (a) in Fig. 2.8 as an example, we have used the decomposition option 3 for F_4 , which corresponds to D_4^3 .

The second part of the chromosomes is related to mapping information. Thus, we have one gene for each task that could be used in an implementation \mathcal{S} , i.e., all the tasks are included in the library \mathcal{L}_A . In our current example (see Table 2.2), we have a total of 19 tasks. The value of the gene for each task encodes the index of the PE where the task will be mapped on. For example, in Fig. 2.8, gray genes are related to mapping. The 1st gray gene “1” in the chromosome (a) means task τ_1 will be mapped on PE N_1 .

(ii) *Initialization*: The initial population is randomly generated and has a population size n .

(iii) *Selection*: We use “tournament selection” to select parents for performing recombination and mutation. In a tournament, four chromosomes are chosen at random, and the fittest one wins. In total, $2(p_c \times n)$ parents are chosen for performing recombination, while $n - (p_c \times n)$ parents are chosen for performing mutation, where p_c is the probability of recombination.

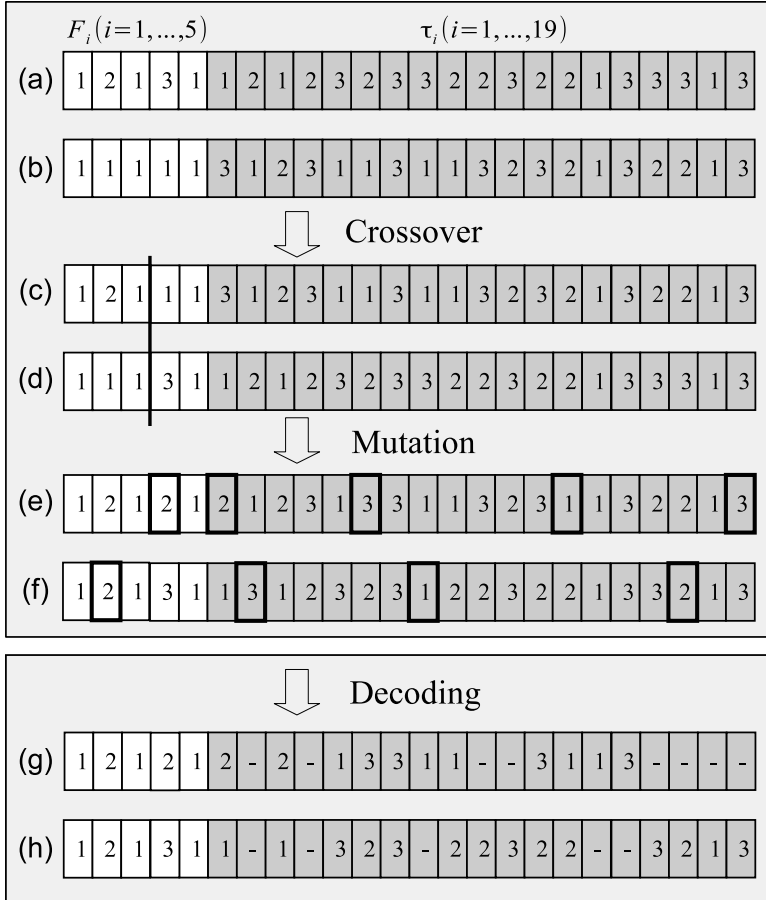


Figure 2.8: Design Transformations and Chromosome Decoding

(iv) *Recombination* (also called *crossover*): We employ a standard single point crossover. For each two parents, we compare a randomly generated number with p_c , if this number $\leq p_c$, the two parents are cut at a random point and the sections after the cut point are swapped to generate the offsprings. Otherwise, the offsprings are just copies of their parents. A crossover example is presented in Fig. 2.8: The chromosome (a) and (b) are recombined along the cut point depicted with a vertical line, obtaining the chromosomes (c) and (d).

(v) *Mutation* is used to add diversity to a population obtained from recombination. For each gene in the chromosome, we compare a randomly generated number with p_m (probability of mutation) and if this number $\leq p_m$, this position is mutated. For the function-to-task allocation genes, we will randomly select another available de-

composition option. In the genes encoding the task mapping, we will randomly select another PE index from the architecture library \mathcal{L}_H . We present two mutation examples in Fig. 2.8. Chromosome (e) is the mutation result of (c) and chromosome (f) is the mutation result of (d). The affected genes are highlighted using bold boxes.

(vi) *Replacement*: Recombination and mutation generate n offsprings out of the n parents in the current population. Before we can evaluate the quality of a implementation \mathcal{S} encoded in a chromosome using the objective functions presented in Section 2.4.1, we need to *decode* the chromosome. We use a fixed size chromosome, which makes crossover and mutation easier, by including a gene for all the tasks in \mathcal{L}_A into the chromosome. However, not all tasks from \mathcal{L}_A will be used in the implementation \mathcal{S} . The tasks used will depend on the decomposition options used for the function blocks. Thus, during the decoding, we go through each gene corresponding to a task and check if that task is used in \mathcal{S} , based on the decomposition options recorded in the first part of the chromosome, i.e., the functional block genes. Fig.2.8(g) and (h) show the decoding of chromosomes (e) and (f), respectively. The tasks not part of \mathcal{S} are marked with “-”, and will not be used in the objective functions calculation.

Replacement decides which n solutions are kept out of the $2n$ solutions available. The key advantage of NSGA-II lies in how it performs selection and replacement, with the goal of preserving diverse non-dominated solutions, in the hope of finding the Pareto-optimal front. See [DAPM00] for the details on the selection and replacement procedures used in NSGA-II.

Steps (iii) to (vi) are repeated until there is no improvement for a given number of consecutive generations, e.g., 30. In the end, we obtain a Pareto-front of solutions, which, however, is not guaranteed to contain the Pareto-optimal, since NSGA-II is a search metaheuristic which does not guarantee optimality.

2.5 Evaluation

To evaluate our proposed “Criticality-Aware Functional Decomposition and Mapping Optimization” (CDMO) approach, we used a synthetic benchmark and a real-life benchmark. The synthetic benchmark has two applications with a total of eight functional blocks. We have assigned SILs from 1 to 3 to the functional blocks. The periods and deadlines have been assigned by 50 or 100 ms. The application model of this synthetic benchmark is presented in Fig. 2.9(a).

We have used a decomposition library \mathcal{L}_A based on Table 2.1. Functional block of SIL 3 has two decomposition options: D_i^1 will be one task of SIL 3, D_i^2 will be two tasks of SIL 2 and SIL 1. Functional block of SIL 2 also has two decomposition options: D_i^1

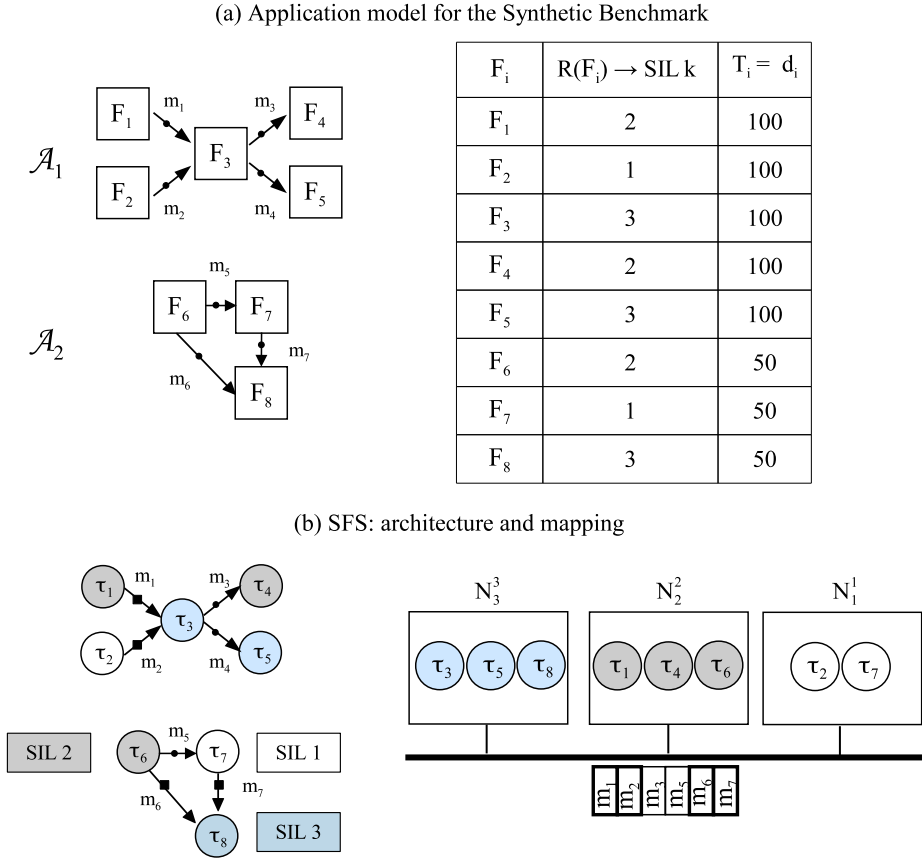


Figure 2.9: Application Model of the Synthetic Benchmark

will be one task of SIL 2, D_i^2 will be two tasks of SIL 1. Functional block with SIL 1 is decomposed into one task of SIL 1. We do not consider the further decomposition, i.e., the decomposed tasks of SIL 2 from functional block of SIL 3 will not be decomposed into two tasks of SIL 1.

The WCET for the decomposed tasks have been randomly decided between 4 – 40 ms. The WCET of each “connecting” task generated by decomposition is assumed to be 2 ms, and the overhead C_o required by the VaM is assumed to be 1 ms. The size of the messages are assumed between 10 – 30 bytes. We were interested to implement these applications on an architecture consisting of three PEs.

The real-life benchmark is an automotive case study adapted from [TCN00], and consists of six applications: Engine Controller (SIL 3, seven functions), Automatic Gear

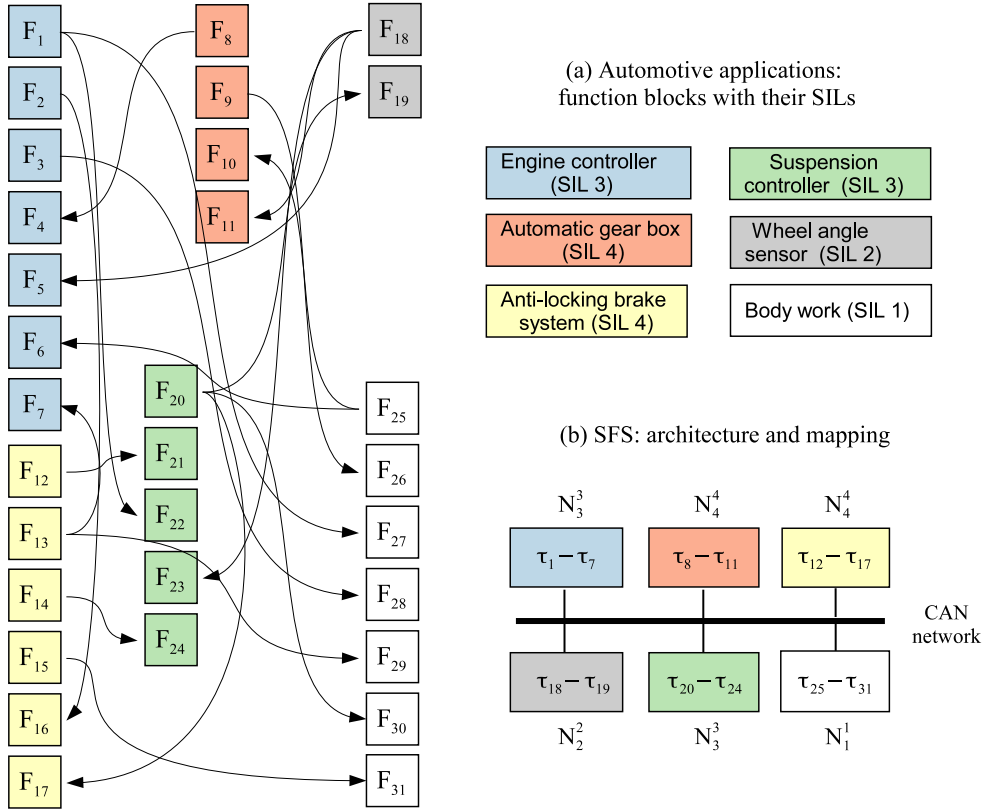


Figure 2.10: Real-life Automotive Applications

Box (SIL 4, four functions), Anti-locking Brake System (SIL 4, six functions), Suspension controller (SIL 3, five functions), Wheel Angle Sensor (SIL 2, two functions), and Body Work related to the passengers (SIL 1, seven functions). See Fig. 2.10(a) for this real-life automotive applications.

We have used the periods and deadlines from [TCN00], and a decomposition library based on the automotive certification standard ISO 26262 [ISO11], see Table 2.1. Each function in Automatic Gear Box (SIL 4) and Anti-locking Brake System (SIL 4) has three decomposition options: D_i^1 will be one task of SIL 4, D_i^2 will be two tasks of SIL 3 and SIL 1, and D_i^3 will be two tasks of SIL 2 and SIL 2. Each function in Engine Controller (SIL 3) and Suspension Controller (SIL 3) has two decomposition options: D_i^1 will be one task of SIL 3 and D_i^2 will be two tasks of SIL 2 and SIL 1. Each function in Wheel Angle Sensor (SIL 2) has two decomposition options: D_i^1 will be one task of SIL 2 and D_i^2 will be two tasks of SIL 1. Functions in Body Work (SIL 1) are only

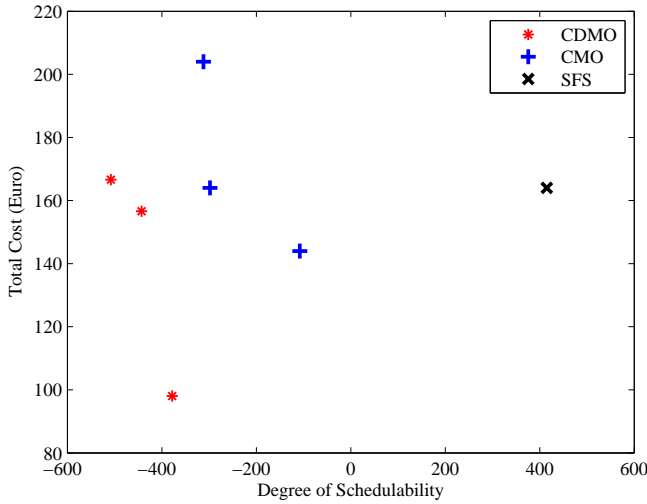


Figure 2.11: Synthetic Benchmark

decomposed into one task of SIL 1. Further decompositions are not considered. The automotive benchmark has to be implemented on an architecture consisting of six PEs.

We have run our CDMO approach to these two benchmarks on their respective architectures. The algorithms were implemented in Matlab 2013a and the experiments run by 5 and 20 minutes on an Intel Core i7 CPU 920 (2.67 GHz) computer. We have tuned the NSGA-II parameters such that the results are as close as possible to the optimal (i.e., no improvements were seen after a very long runtime). We set $n = 100$, $P_c = 0.25$, $P_m = 0.5$ and the search terminates if no improvement is seen after 10 generations.

The Pareto-front of solutions obtained by CDMO are depicted with a red “*” in Fig.2.11 (synthetic benchmark) and Fig.2.12 (real-life automotive benchmark). Together with CDMO, we also present the two other approaches: “Criticality-Aware Mapping Optimization” (CMO) and “Straightforward Solution” (SFS). CMO does not optimize the function-to-task decomposition (it assumes that each function is implemented as a task), but it optimizes the mapping of tasks. SFS does not perform any optimizations: each function is implemented as a single tasks and tasks are cluster based on SILs, hence no mapping optimization is performed. The SFS of the synthetic benchmark is displayed in Fig. 2.9(b), while the SFS of the real-life automotive applications is displayed in Fig. 2.10(b).

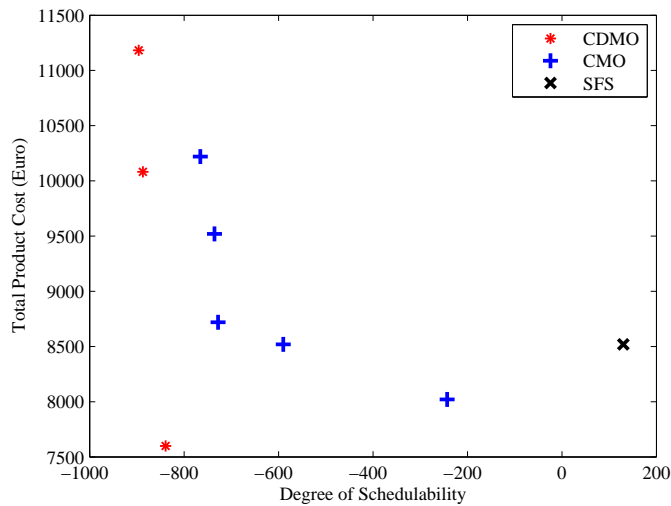


Figure 2.12: Real-life Automotive Applications

Fig.2.11 and Fig.2.12 have the degree of schedulability on the X-axis and the total cost on the y-axis, see Section 2.4.1 for the definition of the cost functions. As we can see from the figures, SFS depicted with a “x”, is not able to obtain schedulable results.

The mapping optimization CMO is performed using the same GA approach as in CDMO, but using a same function-to-task decomposition which functions will not be decomposed into tasks with lower SILs, i.e., no design transformations in the first part of the chromosomes. The Pareto-front of solutions obtained by CMO is shown using blue “+” signs. As we can see CMO is able to obtain schedulable solutions, but they have a high cost.

Only by using CDMO, which performs also the optimization of function-to-task decomposition, we are able to obtain schedulable solutions which also have a lowest cost. This shows that providing automatic tools to support the engineer in performing safety function decomposition can lead to cheaper implementations. The engineer can choose from the Pareto-front solutions that a desired trade-off between cost and schedulability. An improved degree of schedulability means that the implementation has more space for future upgrades.

2.6 Conclusion

In this paper we have considered mixed-criticality applications to be implemented on distributed architectures, and we were interested in decomposing the function-to-task allocation and the task mapping such that the development and unit costs are minimized, and the schedulability is maximized.

We have taken into account safety and integrity requirements. The safety functions are assigned a SIL and we have used SIL decompositions from the standards to ensure that the safety requirements are preserved after decomposition. We also ensure that we use high-reliability PEs for the high-SIL functions. To ensure the communication integrity, we perform a validation in case a lower-criticality task sends information to a higher-criticality task. The separation required by the certification standards for mixed-criticality functions is ensured by partitioning, whereas for integrity preservation we use a validation middleware.

We have proposed a GA-based approach, called CDMO, for our two-objective optimization problem. The algorithm decides the function-to-task allocation, the mapping of tasks to PEs and the reliability of each PE. The proposed CDMO approach has been evaluated on a synthetic benchmark and on a large automotive real-life case study. The results show that, by optimizing the function-to-task allocation, we are able to obtain low-cost schedulable implementations.

CHAPTER 3

Paper B: Reliability-Aware Dynamic Energy Management for Fault-Tolerant Distributed Embedded Systems

This paper presents an approach to the synthesis of low-power fault-tolerant hard real-time applications mapped on distributed heterogeneous embedded systems. We first propose a design-time (offline) synthesis approach which decides the mapping of tasks to processing elements, as well as the voltage and frequency levels for executing each task, such that transient faults are tolerated, the timing constraints of the application are satisfied, and the energy consumed is minimized. Tasks are scheduled using fixed-priority preemptive scheduling, while replication is used for recovery from multiple transient faults. Addressing energy and reliability simultaneously is especially challenging, since lowering the voltage to reduce the energy consumption has been shown to increase the transient fault rate. We present a Tabu Search-based approach which uses an energy/reliability trade-off model to find reliable and schedulable implementations which minimizes the energy consumption. We also propose a runtime (online) synthesis algorithm, which

changes dynamically the voltage and frequency levels of running tasks to reduce further the energy consumption, while guaranteeing the schedulability of the application and its fault-tolerance to transient faults. To provide such guarantees, the offline synthesis has to assume the worst-case, i.e., that tasks will execute up to their worst-case execution times, and the maximum faults will occur. The on-line scheduling will know at runtime the actual execution times of tasks and the fault occurrences. We evaluated the proposed synthesis approaches using several synthetic and real-life benchmarks.

3.1 Introduction

Safety-critical applications have to function correctly, satisfy their timing constraints and be energy-efficient even in the presence of faults. Such faults might be permanent (e.g., damaged microcontrollers or communication links), transient (e.g., caused by electromagnetic interference), or intermittent (appear and disappear repeatedly). The transient faults are the most common [Con03], and their number is increasing due to the rising level of integration in semiconductors.

Researchers have proposed several hardware architecture solutions, such as the Time-Triggered architecture [KB03], that rely on hardware replication to tolerate a single permanent fault in any of the components of a fault-tolerant unit. Such approaches can be used for tolerating transient faults as well, but they incur great hardware costs. Alternatives to such purely hardware-based solutions are approaches such as re-execution, replication, and checkpointing. Several researchers have shown how the schedulability of an application can also be guaranteed with appropriate levels of fault-tolerance [BM94, BDP96, ZC03].

With regard to energy minimization, the most common approach that allows trade-offs between energy and performance during run-time of the application is Dynamic Voltage and Frequency Scaling (DVFS) [SAHE03]. DVFS aims at reducing the dynamic power consumption by scaling down operational frequency and circuit supply voltage. A considerable amount of work has been done on DVFS, see [SAHE03] for a survey. The effectiveness of DVFS for the current processors is debatable [LSH10] because of their reduced dynamic range of power consumption. However, the safety-critical embedded systems we address in this paper typically use mature microprocessors, for which the failure rates are well-documented, and where DVFS is relevant.

Incipient research has analyzed the interplay of energy/performance trade-off and the fault-tolerance techniques [EAHS⁺06, MME04, ZC04]. Redundancy-based fault-tolerance techniques (such as re-execution and replication) and DVFS-based low-power techniques compete for the available slack. The interplay of power management and fault

recovery has been addressed in [MME04], where checkpointing policies were evaluated with respect to energy. In [EAHS⁺06], time redundancy was used in conjunction with information redundancy, which does not compete with DVFS for slack, to tolerate transient faults. In [ZC04], fault tolerance and dynamic power management were studied, and rollback recovery with checkpointing was used to tolerate multiple transient faults in distributed systems.

Addressing energy and reliability simultaneously is especially challenging because lowering the voltage to reduce energy consumption has been shown to increase the number of transient faults exponentially [ZMM04]. The main reason for such an increase is that, with lower voltages, even very low energy particles are likely to create a critical charge that leads to a transient fault.

Several researchers have started to address this aspect. A single-task reliability-aware checkpointing scheme was evaluated in [ZMM04]. Applications distributed on multiprocessors scheduled using non-preemptive static cyclic scheduling and modeled with Directed Acyclic Graphs (DAG) have been addressed in [AGK12, PPIE07]. Pop et al. [2007] have proposed a constraint logic programming based approach, which, considering a given mapping of tasks to processors, decides the voltage and frequency levels, and the start time of tasks in the static schedule such that transient faults are tolerated, the timing constraints of the application are satisfied and the energy is minimized. The authors have used task re-execution in a shared recover block to tolerate the transient faults.

Aydin and Zhu [2009] have proposed a reliability-aware DVFS heuristic for independent task sets on uni-processor systems, which schedules a separate recovery task for each task executed at a reduced voltage and frequency level, in order to preserve the desired reliability. This approach has been improved in [ZAZ13] to consider a shard-recovery scheme, i.e., several tasks will use a shard recovery block, in the context of dependent tasks modeled using DAG. Researchers have also addressed reliability in the context of temperature-aware design. Their focus has been on limiting the operating temperature in order to increase the life-time of the system [CRM⁺06]. Such a technique could be used in conjunction with our approach.

Pop et al. [2007], as mentioned, consider energy/reliability trade-offs in the context of distributed time-triggered systems, where tasks and messages are scheduled based on a static-cyclic scheduling policy, and transient faults are tolerated using task re-execution. In [PIEP09], researchers show how re-execution and active replication can be combined in an optimized implementation that leads to a schedulable fault-tolerant application without increasing the resources required.

In this paper, we consider heterogeneous distributed event-triggered systems, where tasks are scheduled using fixed-priority preemptive scheduling, and messages are scheduled using fixed-priority non-preemptive scheduling. Transient faults are tolerated

through task replication. In this context, we propose an optimization approach that decides at design time (offline) the mapping and the operating voltage and frequency level of tasks, such that transient faults are tolerated, the timing constraints of the application are satisfied, and the energy consumed is minimized.

We have developed a Tabu Search-based approach for the synthesis of fault-tolerant event-driven systems that takes into account the influence of voltage and frequency scaling on reliability. The offline synthesis, assumes that tasks will execute up to their worst-case execution time (WCETs) and that all the task replicas will have to execute in order to tolerate transient fault occurrences. However, during runtime, tasks typically will execute only a fraction of their WCETs [EY97]. In addition, the knowledge about fault occurrences, i.e., which replicas do not have to execute because faults have not occurred in the original task, could potentially be used for further energy savings. Our interest is to obtain such energy savings online without impacting negatively the reliability. Therefore, in this paper, we also propose an online DVFS algorithm which, based on the knowledge of actual task execution times and fault occurrences, decides the operating voltage and frequency levels of tasks such that the energy is further reduced (compared to the offline synthesis), the transient faults are tolerated and the timing constraints (deadlines of tasks) are satisfied.

There is limited work on reliability-aware online DVFS. The approaches in [ZC06, WMWZ12] consider uni-processor systems and have an online component that decides the voltage and frequency levels to minimize energy, such that the task deadlines are satisfied. Fault-tolerance is achieved by using checkpointing with rollback recovery. However, these approaches ignore the negative impact of lower voltages on the reliability [ZMM04]. This aspect is taken into account in the online approach proposed in [ZAZ13] which decides the frequency for tasks under schedulability and fault-tolerance guarantees, but Zhao et al. [2013] also consider only uni-processor systems.

The next two sections present system models and reliability model. Section 3.4 introduces the problem and discusses a motivational example. Our proposed offline reliability-aware mapping, voltage and frequency scaling approach is presented in Section 3.5, and the online reliability-aware voltage and frequency scaling algorithm is introduced in Section 3.6. Section 3.7 presents the evaluation of the proposed synthesis approaches. We draw our conclusions in the last section.

3.2 System Model

We consider hardware architectures consisting of a set \mathcal{N} of heterogeneous processing elements (PEs) and interconnected by a shared communication channel. Tasks

are scheduled using fixed-priority preemptive scheduling. Researchers have shown [TSPS12a] how realistic protocols such as TTEthernet can be taken into account during the analysis and synthesis. However, for simplicity, in this paper we consider that the PEs are interconnected by a shared bus, which uses a fixed-priority dynamic scheduling scheme and assumes that the messages are non-preemptible. This is similar to how a widespread bus protocol such as the controller area network [Bos91] works.

In this paper we are interested in reducing the energy consumption without negatively impacting reliability and schedulability. We use DVFS [SAHE03] for performing power management in PEs of the architecture. For new processors and memory architectures, DVFS is limited in its potential for energy savings, in some situations, paradoxically, leading to an increased power consumption [LSH10]. In addition, other system components, such as memories, and I/Os have an increased energy consumption. However, as we mentioned in the introduction, in this paper we are interested in safety-critical embedded systems, which use mature platforms due to dependability reasons, where DVFS is still effective [LSH10]. We use similar assumptions and energy model as all the previous research [AZ09, ZAZ13, AGK12], which focuses on the energy management of PEs.

We use the power model from [BBL09], as it is able to capture a set of realistic assumptions, such as discrete operating modes of PEs, the energy and delay due to mode switches, and takes I/O operations into account. Thus, a PE N_i is characterized by a set of operating modes, where each operating mode, $\Lambda_j^{N_i} = (f_j^{N_i}, v_j^{N_i}, p_j^{N_i})$, is described by three parameters: $f_j^{N_i}$ is the operating clock frequency of N_i running in mode j , which is measured in Hz; $v_j^{N_i}$ is the supply voltage measured in Volts; $p_j^{N_i}$ is the power spent measured in Watts. We also introduce the normalized frequency F and normalized voltage V where $F_j^{N_i} = f_j^{N_i} / f_{max}^{N_i}$ and $V_j^{N_i} = v_j^{N_i} / v_{max}^{N_i}$, respectively.

In this paper we are interested in tolerating transient faults, which are the most common faults in today's embedded systems [Con03]. There are many transient errors detection techniques [CMS82] with varying error detection coverage and overhead. Similar to the previous research [AZ09, ZAZ13, AGK12], we assume that faults are detected at the completion of a task's execution, and the overhead of fault detection is included in the task's WCET. Researchers have shown [PIEP09] how re-execution, which provides time-redundancy, and active replication, which provides spatial-redundancy, can be combined in an optimized implementation that reduces the fault-tolerance overheads.

In this paper we are not interested in the optimization of the redundancy mechanisms needed for fault tolerance. Hence, for simplicity, we have decided to use active replication to tolerate transient faults. Thus critical tasks will have replicas that will execute on the same or a different PE, delivering the desired functionality in case the original task fails due to a transient fault. Our work can be extended to consider any other fault-tolerance mechanisms such as re-execution or check-pointing with rollback recovery.

We model an application as a set Γ of periodic real-time tasks. The mapping of a task τ_i to a PE N_j is captured by the mapping function $\mathcal{M}: \Gamma \rightarrow \mathcal{N}$, i.e., $\mathcal{M}(\tau_i) = N_j$. This mapping is not yet known and will be decided by our approach presented in Section 3.5. For each task τ_i we know its WCET $C_i^{N_j}$ (in the maximum speed operating mode) with respect to each N_j which it is considered for mapping, its period T_i and deadline D_i . An application Γ is repeated periodically with a period T_{cycle} . Since we use fixed-priority preemptive scheduling for tasks, each task τ_i is assigned a unique priority.

Tasks are divided into two categories: critical and non-critical. To prevent the application failure, critical tasks have to tolerate transient faults. We assume that for each critical task τ_i , the designer will specify a desired *redundancy level* k_i , i.e., how many replications of τ_i have to be introduced, see Section 3.3 for details. The desired redundancy level is captured by the function $\mathcal{F}: \mathcal{F}(\tau_i) = k_i$. If $k_i = 0$, the task is non-critical. A critical task and its replicas could be mapped on the same PE, or mapped on different PEs, and can run in different operating modes.

Each execution of a periodic task is called a job. The j th job of task τ_i is referred to as J_{ij} . Our offline synthesis will assign a same operating mode to all jobs J_{ij} of a task τ_i . The assigned mode of task τ_i is captured by the function $\mathcal{L}: \Gamma \rightarrow \{\Lambda_\ell^{\mathcal{M}(\tau_i)}\}$. However, the online scheduling may decide at runtime to run jobs of the same task in different operating modes. In that case, the function $\mathcal{L}(J_{ij})$ will refer to the mode of the job J_{ij} .

For simplicity, we do not consider the situation when an intermediate mode can be obtained by using two modes applied to different execution segments of the same task, as in [BBL09]. However, our approach can be extended to consider this aspect.

Tasks communicate using messages. We know the size s_{m_i} and the priority of each message m_i . Our model captures two types of communication: Sampling and queuing. A *sampling* communication has a buffer storage for a single message; arriving messages overwrite the buffer, and reading does not remove the message, i.e., it can be read repeatedly. A *queuing* communication uses a buffer that can store several messages, and works as a FIFO queue. A reader task will block if the buffer is empty and a writer task will block if the buffer is full. We assume that the buffer size have been determined such that there is no overflow or underflow Manolache et al. [2006].

The bus is used also to communicate fault-occurrence information. Thus, if a task does not experience a fault, and one of its replicas is on a different PE, we broadcast a message informing the scheduler on the other PEs not to start the replicas (or terminate them, if they are executing). Our schedulability analysis takes all these messages into account.

3.3 Reliability Model

Safety is a property of a system that will not endanger human life or the environment. A *Hazard* is a situation in which there is active or potential danger to people or the environment. *Risk* is a combination of the probability of a hazardous event and its consequence. If, after performing an initial hazard and risk analysis, a system is considered safety-critical, it has to be certified [KK10]. *Certification* is a “conformity of assessment” performed by a third party, according to area-specific certification standards. For example, the IEC 61508 standard is used for industrial applications, ISO 26262 is for the automotive area, and D0-178B is used for avionics software.

During the engineering of a safety-critical system, the risks are assessed and the appropriate risk control measures mandated by the standards are introduced to reduce the risk to an “acceptable level”. Thus, safety functions are assigned a *Safety-Integrity Level* (SIL), which captures the required level of risk reduction. The SIL assigned to a software task or hardware component will dictate the development processes and amount of redundancy that have to be used.

In this paper we are interested to tolerate hardware transient faults which manifest themselves at the task-level. Note that all software faults (bugs) are permanent, i.e., they are due to specification, design or implementation mistakes. Software does not experience transient faults similar to hardware, since it is not aging, for example. A software bug disappears only if the software is updated with a new version, where the bug has been removed. If permanent faults have to be tolerated, we assume that the designer has introduced the appropriate measures. For example, permanent faults in hardware are tolerated through hardware redundancy: using several (identical) hardware modules, such that a healthy module can take over if the original module is faulty. Fault masking configurations (such as Triple Modular Redundancy) are used when no erroneous output can be allowed even for a short period of time, and reconfiguration, which switches in a spare module after the fault is detected, is used otherwise. Permanent faults in software can only be tolerated using some form of functional diversity, i.e.,: developing separate software modules using different specifications, teams, methods, in the hope that the permanent faults are not correlated across the modules [KK10].

Transient faults are typically tolerated by re-executing again the failed task. By its nature, the transient fault has most likely disappeared, and the re-execution will happen fault-free. If time constraints are not strict, the re-execution will be scheduled after the transient fault has been detected. To reduce the recovery delay, a replica of a task can be scheduled in parallel on a different PE. Such an approach is called *active replication*. In this paper we allow replicas to be executed also on the same PE (amounting, practically, to re-execution) if our synthesis decides that this is a good option. Finally, checkpointing with rollback recovery is used for tasks with very long computation times, where

saving the current status of a task into periodic checkpoints is better than waiting until the end to recover [KK10].

The amount of redundancy used for fault tolerance is determined by the reliability of the components and the criticality of the function (i.e., the SIL, which corresponds to a certain reliability objective). The more reliable a component (hardware module or software task), the less redundancy has to be used. We define the *reliability* as the probability of successful execution. Since we are interested in tolerating transient faults, we have to determine the probability that a task will not fail due to a transient fault.

The previous related research [AZ09, ZAZ13, AGK12] has used the exponential failure law to capture the reliability R_i of a task τ_i :

$$R_i = e^{-\lambda c_i} \quad (3.1)$$

where λ is the *permanent* fault rate of the PE running τ_i and c_i is the execution time of the task. However, the exponential failure law refers to permanent hardware faults, [KK10] and does not accurately capture the probability of correct operation in case of transient faults. In this case, researchers have proposed to use a Weibull distribution for hardware transient modeling [CMS82]:

$$R_i = e^{-(\lambda_j^\ell c_i)^{\mu_j^\ell}} \quad (3.2)$$

where λ_j^ℓ now captures the hardware transient fault rate of the PE N_j running task τ_i in the operating mode ℓ , and μ_j^ℓ is a shape parameter. A value of $\mu_j^\ell < 1$ indicates that the failure rate decreases over time, for $\mu_j^\ell = 1$ the failure rate is constant, and a value of $\mu_j^\ell > 1$ means that the failure rate increases with time. More complex models, which can capture the dependence of transient errors on patterns of usage, have also been proposed, but they are beyond the scope of this paper [CMS82].

Note that not all hardware transient faults will manifest themselves as software task failures. This depends on the length of the transient faults, the fault type and its source in terms of the architectural component affected [WRPG11]. The percentage of hardware transient faults affecting the software can be determined by fault injection, and our model can use this information, if available. However, since quite a large percentage of hardware transients will lead to a task failure [WRPG11], we will use Eq.3.2 as it is a safe (pessimistic) estimation of task failures.

Researchers have so far assumed that the software itself is perfect and hence will only fail because of transient hardware failures [AZ09,ZAZ13,AGK12]. However, software bugs are very common, and for this reason the certification standards make the opposite assumption, i.e., that software always fails [IEC98] and require the use of appropriate mitigation. The reason the standards make this assumption is that software reliability modeling is difficult and there are no industrial-strength well-established software reliability models, and using software redundancy without using diversity does not work, as it does for hardware. Under this assumption, we consider that the permanent software faults have already been addressed as discussed, through software diversity, and we do not explicitly take them into account in our model. Therefore, we capture the reliability R_i of executing a task τ_i on a PE N_j using Eq.3.2, where, for simplicity, we consider that the fault rate is constant over time, i.e., $\mu_j^\ell = 1$.

The reliability of a critical task, R_i^{rep} , is increased through introducing k_i replicas. R_i^{rep} is expressed as the probability of *not* having all these tasks fail,

$$R_i^{rep} = 1 - \prod_{i=1}^{k_i+1} (1 - R_i) \quad (3.3)$$

Note that when $k_i = 0$ (i.e., the task is non-critical), $R_i^{rep} = R_i$. Considering independent faults, the reliability R_S of the system for a period of time T_{cycle} is determined as

$$R_S = \prod_{J_{ij} \in \Gamma} R_i^{rep} \quad (3.4)$$

where $J_{ij} \in \Gamma$ are all the jobs of the application Γ released during T_{cycle} . Note that the reliability is a function of time, i.e.,: the probability of continuous successful execution over a given time period. For safety-critical embedded systems, this time is typically defined as the ‘‘mission duration’’. We denote the mission duration with T_S .

3.3.1 Energy/Reliability Trade-off Model

Let us denote with $c_{ij}^{N_j}$ the execution time of a job J_{ij} in the fastest operating mode. Then, considering the well-known energy/performance tradeoff with DVFS, the execution time c_{ij} of a job J_{ij} mapped on PE N_j and executed in mode ℓ is $c_{ij} = c_{ij}^{N_j} / F_\ell^{N_j}$. The energy consumption E_S of finishing all jobs during T_{cycle} is calculated by:

$$E_S = \sum_{J_{ij} \in \Gamma} p_\ell^{N_j} c_{ij} \quad (3.5)$$

where Γ' is the set of all tasks in the application Γ to which we add all the replicas, J_{ij} are all the jobs of the tasks in Γ' released during T_{cycle} , and $p_\ell^{N_q}$ is the power consumed by N_q in the operating mode ℓ and c_{ij} is the execution time of job J_{ij} considering the mode ℓ on N_q .

Note that the assigned redundancy levels k_i are only valid under the assumption that the system reliability R_S does not fall below a specified reliability goal R_g . If R_S is below R_g , the fault rate might be higher and thus more redundancy might be necessary to tolerate the increased number of faults.

According to the energy/reliability trade-off model from [ZMM04], which is used in all of the related work [AZ09, ZAZ13, AGK12], the fault rate depends on the operating mode, i.e., lowering the voltage to reduce energy consumption has been shown to increase the number of transient faults exponentially [ZMM04]. More formally, [ZMM04] shows that when a PE runs in the minimum voltage and frequency level, the fault rate increases 10^d times compared to that of PE running in the maximum voltage and frequency level, where d ($d > 0$) is a PE-specific constant.

Let us define λ_j^0 as the minimum fault rate, which corresponds to the maximum voltage and frequency mode $\Lambda_{max}^{N_j}$. Thus, the maximum fault rate λ_j^{max} , which corresponds to the minimum voltage and frequency mode $\Lambda_{min}^{N_j}$ is:

$$\lambda_j^{max} = \lambda_j^0 \cdot 10^d \quad (3.6)$$

where d (> 0) is an PE-specific constant.

Based on [ZMM04], and using the normalized frequency F and normalized voltage V of PE N_j running on operating mode ℓ , we get the fault rate for a specific operating mode:

$$\lambda_j^\ell = \lambda_j^0 \cdot F^\alpha \cdot 10^{-\beta V} \quad (3.7)$$

where α and β are calculated by the given λ_j^0 and λ_{max}^0 . Further, we have considered $d = 2$, i.e., the fault rate increases 100 times between the maximum and minimum operating modes [ZMM04]. Note that this mode-dependent fault rate is used in the task reliability calculation from Eq.3.2. Since the fault rate increases exponentially when supply voltage and operating frequency decrease, switching the operating mode has an impact on the system reliability.

3.4 Problem Formulation

The problem we are addressing in this paper can be formulated as follows. Given (1) an application modeled as a set Γ of tasks, with associated redundancy levels captured by function \mathcal{F} , (2) an architecture consisting of a set \mathcal{N} of heterogeneous PEs that work under different operating modes Λ , and (3) a reliability goal R_g which bounds the system reliability R_s , we are interested in synthesizing an implementation \mathcal{S} , such that the deadlines of all tasks are satisfied, the system reliability is within the imposed reliability goal, i.e., $R_s \geq R_g$, and the energy consumption is minimized.

Synthesizing an implementation $\mathcal{S} = (\mathcal{M}, \mathcal{L})$ means deciding offline (1) the mapping \mathcal{M} of tasks and replicas to the PEs and (2) the operating mode \mathcal{L} for executing each task.

The offline synthesis has to assume that the tasks will execute up to their WCET and that we need to execute all replicas. At runtime we will know the actual task execution times and fault occurrences. Hence, at runtime we are interested to dynamically set the operating mode for executing each job in order to exploit the resulted slack, such that the energy is further reduced while the timing and reliability constraints are guaranteed.

3.4.1 Motivational Example

Let us consider the example in Fig.3.1 where we have a task set of six tasks (four tasks and two replicas) mapped on an architecture of two PEs. For simplicity, in this example we ignore the communication. The WCETs, periods, deadlines and priorities of all tasks are presented in the left table. τ_1 and τ_2 are critical tasks and they are replicated once. We denote the replicated task of τ_i as τ'_i and its jobs as J'_{ij} . Recall that tasks are periodic and they are executed in accordance to fixed-priority preemptive scheduling. The operating modes of the two PEs are given in the right table, which also contains λ^0 , α , and β (we assume the same values for both PEs in this example). We set the shape parameter μ_j^ℓ to 1 (see Eq.3.2) for both PEs and consider the application cycle $T_{cycle} = 100$ ms. We denote with E_0 and R_0 the energy and reliability, respectively, of \mathcal{S}_0 , see Fig. 3.1. In the motivational example we have set R_g to 0.99960, which means that we accept a probability of failure that is ten times greater than R_0 , i.e., $R_g = 1 - 10(1 - R_0)$.

Our problem has two components: an offline synthesis component and an online component. We will use the example to discuss first the offline synthesis. Fig.3.2 shows two alternatives to the offline synthesis problem. The figure shows a timeline for each PE. As mentioned, the offline scheduling has to assume that the tasks will execute up to their WCET (the length of the rectangles in the timeline) and that all replicas are

Γ'	$C_i^{N_1}$	$C_i^{N_2}$	$T_i = D_i$	Priority
τ_1	7	14	50	1
τ_2	6	12	100	4
τ_3	5	10	50	2
τ_4	8	16	100	5
τ'_1	7	14	50	3
τ'_2	6	12	100	6

$T_{cycle} = 100$ ms

ℓ	N_1			N_2		
	Freq. [MHz]	Volt. [V]	Power [W]	Freq. [MHz]	Volt. [V]	Power [W]
1	333	1.2	4	166	1.1	2
2	666	1.4	12	333	1.25	4.5
3	1000	1.6	25	500	1.5	11

$\alpha = -4, \beta = -0.04, \lambda^0 = 10^{-7}$
 $E_0 = 1312$ mJ. $R_0 = 0.9999969$. $R_g = 0.9999969$

Figure 3.1: System Model Example

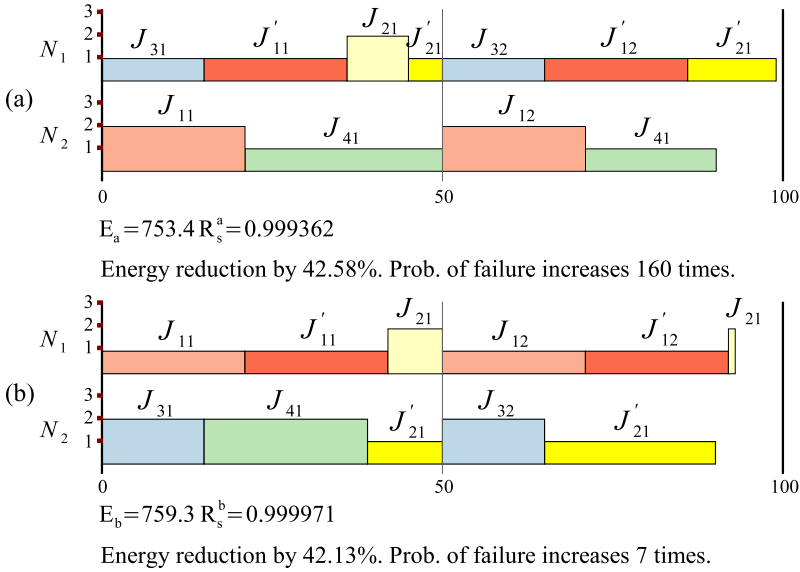


Figure 3.2: Offline Synthesis

executed. In addition, the timeline on each PE considers the scenario when all tasks are released at $t = 0$. However, tasks might be released at different times. In Section 3.5, we discuss the schedulability analysis and the assumptions considered.

The offline synthesis has to decide the mapping and the operating mode for each task. In the figures, the mapping is indicated by placing the task on the timeline of the PE where it is mapped and the height of the rectangle indicates the operating mode. For the discussion, we use a reference implementation S_0 which does not perform voltage and frequency scaling, and thus runs all the tasks in the maximum speed operating mode and S_0 attempts to reduce the energy consumption by mapping the tasks on the PEs which consume the least energy in the maximum operating mode, without missing their deadlines.

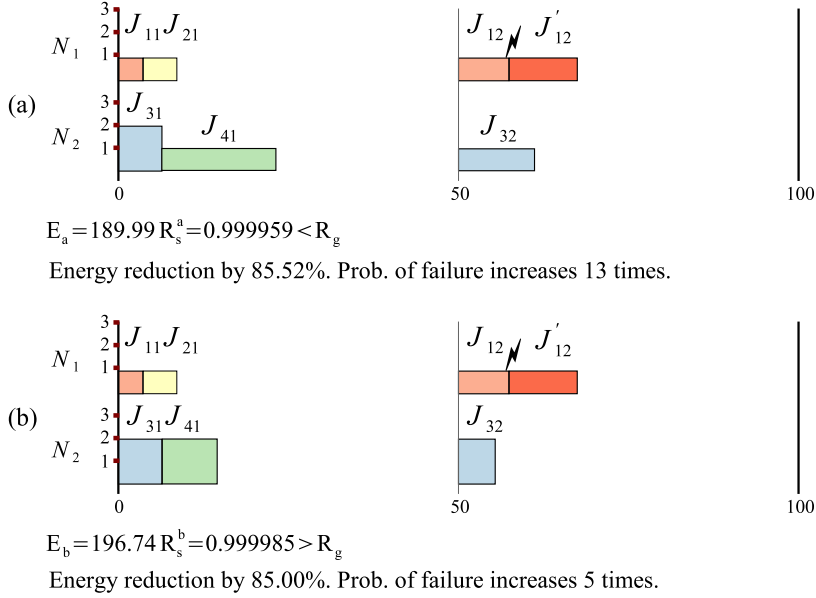


Figure 3.3: Online Scheduling

Fig.3.2(a) shows the solution \mathcal{S}_a which minimizes the energy consumption without concerning for reliability. \mathcal{S}_a meets all tasks' deadlines and reduces the energy by 42.58%, compared to E_0 . However, \mathcal{S}_a does not meet the system reliability goal, i.e., $R_{\mathcal{S}_a} = 0.999362 < R_g = 0.999969$. Since $1 - R_S$ captures the probability of failure, we use the following equation to measure the degeneration of the system reliability, in case of \mathcal{S}_a :

$$\theta = \frac{1 - R_S^a}{1 - R_0} \quad (3.8)$$

where R_0 is the system reliability of \mathcal{S}_0 . According to Eq. 3.8, the probability of failure for \mathcal{S}_a is 160 times greater than that of \mathcal{S}_0 .

By carefully deciding the mapping \mathcal{M} and operating modes \mathcal{L} , it is possible to reduce the negative impact on reliability without a significant loss of energy savings. Another implementation, \mathcal{S}_b , shown in Fig.3.2(b) fulfills all timing and reliability requirements, with only 0.45% loss in energy savings, compared to \mathcal{S}_a .

Let us now discuss the online component of our synthesis approach. At runtime, tasks will finish before their WCETs and in the case that no errors are occurring in the critical tasks, the corresponding replicas are terminated. This will increase the available slack in the schedule. Hence, at runtime, additional energy savings can be obtained by exploiting this additional slack. Our online scheduling uses the mapping decided by

the offline component, but will decide at runtime a different (if needed) operating mode for executing each *job* such that the energy is further minimized, under reliability and timing constraints.

We consider as the starting point the offline synthesis result from Fig.3.2(b) and we assume that, during runtime, a single fault is going to happen in the second job of task τ_1 , namely J_{12} . Fig.3.3 shows two online scheduling alternatives. We assume that the jobs will finish before their WCETs, and the actual execution times during runtime are shown by the length of the rectangles in Fig.3.3, under the respective operating mode (which is the height of the rectangle).

Fig.3.3(a) shows an online scheduling solution which uses the additional slack at runtime to reduce further the energy, but without enforcing the reliability constraint R_g . Job J_{11} finishes before its WCET and does not experience a fault. This means that we do not have to execute the replica J'_{11} . The online scheduling will use the slack resulted the earlier finish of job J_{11} and the removal of J'_{11} to run J_{21} in a lower operating mode compared on Fig.3.2(b), in order to save energy.

In the runtime scenario depicted in Fig.3.3(a), J_{21} will finish without faults before its replica J'_{21} is released on N_2 . Our online scheduling approach uses the shared bus to broadcast the error information to all PEs (we take these extra messages into account during schedulability analysis). Thus, J'_{21} is not activated on N_2 , creating extra slack. However, because a fault has occurred in J_{12} , its replica, J'_{12} , is executed.

We use the slack created on N_2 due to the earlier termination of J_{31} and due to not executing J'_{21} to scale down the operating modes of J_{41} and J_{32} compared to the offline solution in Fig.3.2(b). This leads to an energy reduction of 85.52% compared to S_0 ¹. However, this energy reduction comes at the cost of not meeting the reliability goal. The system reliability of Fig.3.3(a) is only 0.999959, which is lower than R_g (0.999969), and increases 13-times in the probability of failure.

In our online scheduling we are interested to exploit the extra slack created at runtime to further reduce the energy, but without violating the reliability goal. Such a solution is shown in Fig.3.3(b). Compared to the solution in Fig.3.3(a), we decide at runtime not to scale down further J_{41} and J_{32} , in order to preserve the reliability goal. Thus, the resulted system reliability is 0.999985, which is within the reliability goal. Compared to Fig.3.3(a), in Fig.3.3(b) we are still able to reduce the energy consumption to a similar level, but this time without negatively impacting the reliability.

These examples show that it is important, both offline and online, to carefully optimize the mapping of tasks and the operating modes of jobs such that the reduction of energy

¹The reference energy E_0 , which considers running all tasks in the maximum speed operating mode and up to their WCETs, is a very pessimistic metric.

does not lead to impaired reliability. The next two sections present our proposed offline and online scheduling algorithms, respectively.

3.5 Offline Synthesis

Determining an optimal task mapping is a NP-hard problem [TBW92]. To solve the offline synthesis problem presented in the previous section, we propose a Tabu Search-based approach, which decides offline the mapping \mathcal{M} and operating mode \mathcal{L} for executing each task.

Tabu Search (TS) [Glo89] is an optimization metaheuristic which iteratively explores the solutions in the vicinity (neighborhood) of the current solution, selecting the ones which optimize the *cost* function. Our proposed cost function in Eq. 3.9 captures the energy minimization under timing and reliability constraints:

$$\text{cost}(\mathcal{S}) = E_{\mathcal{S}} + W_R \cdot \max(0, R_g - R_{\mathcal{S}}) + W_{\mathcal{S}} \cdot r_{\mathcal{S}} \quad (3.9)$$

where the first term is the energy consumption of running the implementation \mathcal{S} , the second term is the reliability constraint and the third term represents the timing constraint. Instead of considering solutions which do not meet timing and reliability constraints as infeasible, and to ignore them during the design space exploration, we decided penalize such solutions in the *cost* function by giving large values, the weights of the corresponding terms, namely W_R and $W_{\mathcal{S}}$, which allows us to explore infeasible regions of the search space and drive the search towards feasible regions.

The reliability constraint is enforced by calculating difference between R_g and $R_{\mathcal{S}}$. If the reliability constraint is satisfied ($R_{\mathcal{S}} \geq R_g$) then the second term is zero. Otherwise, it is a large positive value, depending on the penalty weight W_R . The timing constraints are enforced through the “degree of schedulability” $r_{\mathcal{S}}$, which is calculated as:

$$r_{\mathcal{S}} = \sum_{\tau_i \in \Gamma'} \max(0, r_i - D_i) \quad (3.10)$$

We use a response-time analysis [But11] to calculate the worst-case response time (WCRT) r_i of every task τ_i , which is compared to the deadline D_i . The basic analysis presented in [But11] has been extended over the years. For example, the state-of-the-art analysis in [PGH98] considers arbitrary arrival times and deadlines, offsets and inter-task communication. The schedulability analysis takes into account the delays of the “queuing” messages when determining the WCRT of receiving tasks, and checks the schedulability of the bus considering all type of messages.

Algorithm 1 MVFS

```

1:  $\mathcal{S}^0 \leftarrow \text{InitialSolution}(\Gamma', \mathcal{N})$ ;
2:  $\mathcal{S} \leftarrow \mathcal{S}^{now} \leftarrow \mathcal{S}^0$ 
3: while  $\text{max\_iterations}$  is not exhausted do
4:    $\mathcal{C} \leftarrow \mathcal{M}\text{-moves}(\mathcal{S}^{now}, m)$ 
5:   for each solution  $\mathcal{S}_i^{new} \in \mathcal{C}$  do
6:      $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{L}\text{-moves}(\mathcal{S}_i^{new}, P_L, n)$ 
7:   end for
8:    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{L}\text{-moves}(\mathcal{S}^{now}, P_L, n)$ 
9:   for each solution  $\mathcal{S}_j^{new} \in \mathcal{C}$  do
10:    Calculate  $\Delta_j = \text{cost}(\mathcal{S}^{now}) - \text{cost}(\mathcal{S}_j^{new})$ 
11:   end for
12:    $\mathcal{S}^{now} \leftarrow \text{SelectNewSolution}(\Delta_j, \text{length\_tl})$ 
13:    $\mathcal{S} \leftarrow \text{SaveBestSolution}(\mathcal{S}^{now})$ 
14: end while
15: return  $\mathcal{S}$ 

```

If the application is schedulable, each r_i is smaller than the corresponding deadline D_i . Then $r_S = 0$ which means that the schedulability constraint is satisfied and the third term in Eq. 3.9 is zero. However, if the application is not schedulable, there exists at least one r_i greater than the deadline D_i , therefore r_S is positive. This means that the third term in Eq. 3.9 is a large positive value depending on the penalty weight W_S .

Algorithm 1 presents our reliability-aware Mapping, Voltage and Frequency Scaling optimization (MVFS), based on Tabu Search algorithm. MVFS takes the complete task set Γ' , i.e., the set of original task Γ and the replicas, and the architecture \mathcal{N} as inputs, and returns the implementation $\mathcal{S} = (\mathcal{M}, \mathcal{L})$ which minimizes the cost function. The search starts from an initial solution \mathcal{S}^0 (line 1) which is a mapping such that the utilization of the PEs is balanced and the communication is minimized. In \mathcal{S}_0 , all tasks are assigned the maximum speed operating mode.

Tabu Search explores the design space by using design transformations (called also “moves”), applied to the current solution \mathcal{S}^{now} in order to generate neighboring solutions. The search iteratively moves from the current solution \mathcal{S}^{now} to a new solution \mathcal{S}^{new} in the neighborhood of \mathcal{S}^{now} . We first introduce the moves we have used in our implementation and then we explain the search. We will use an example to discuss how Tabu Search works. We perform two types of design moves: (1) mapping moves (\mathcal{M} -moves) and (2) voltage and frequency scaling moves (\mathcal{L} -moves). The neighborhood might be very large, and evaluating the cost function for all neighbors is computationally very expensive, thus we consider a limited number of neighboring solutions called a *candidate set*, denoted with \mathcal{C} .

We introduce the following neighbors (lines 4-8) into the candidate set \mathcal{C} . In line 4, we add m neighbors obtained using \mathcal{M} -moves into \mathcal{C} . For each \mathcal{M} -move (line 4), we move a randomly chosen task from one PE to another PE, or swap two randomly chosen tasks between two PEs. The tasks involved in an \mathcal{M} -move will be assigned the highest operating mode on the new PE. In lines 5-7, for each new solution \mathcal{S}_i^{new} generated in line 4, we add n neighbors into the candidate set \mathcal{C} obtained by modifying them using \mathcal{L} -moves. For each task in Γ' , if a randomly generated number is larger than the given P_L , the task will be affected by an \mathcal{L} -move, i.e., the task is assigned with another operating mode. In line 8, we also consider adding a number of n neighbors of \mathcal{S}^{now} by only performed \mathcal{L} -moves into \mathcal{C} . The \mathcal{L} -moves in line 8 is performed as the same as in line 6.

In each iteration of the loop in lines 3-14, TS selects a neighbor from \mathcal{C} as the current solution \mathcal{S}^{now} , from which the search will continue. To avoid being stuck in a local optimum, and to prevent cycling TS filters the neighborhood using a memory structure called a *tabu list*. In our implementation, we mark as *tabu* the move that generated an improved solution \mathcal{S}^{now} . We record a number of *length_tl* such moves in the tabu list. The size *length_tl* of the tabu list is also called the tabu tenure.

We evaluate all candidates in \mathcal{C} (lines 9-11). For selecting a new solution, we first attempt to select an improved solution with the largest improvement compared to current solution, as long as it is not on the tabu list or its tabu status can be *aspirated*. A tabu status can be *aspirated* when the solution has a lower cost than that of the current best-known solution. If no such improved solution exists, we randomly select a non-improved solution to be the new solution as long as it has a non-tabu status, i.e., it is not on the tabu list. Randomly accepting non-improving moves introduces the *diversification*, which forces the search into previously unexplored areas of the design space.

The selection of new solution and the maintenance of tabu list are done inside the `SelectNewSolution` function (line 12 of Algorithm 1). Then, `SaveBestSolution` function (line 13) saves the new solution if it is the best solution so far. After *max_iterations* without an improvement, the algorithm stops and the best-found \mathcal{S} is reported. Our algorithm can also be stopped if a given time limit has been exceeded.

Let us illustrate how TS works using the example in Fig. 3.4, where we have the application and an architecture from Fig.3.1. In Fig. 3.4, the tasks are represented using rectangles labeled with the task name. The length of the rectangle indicates the WCET of the respective task while the height reflects the operating mode. We assume that during the search, TS has reached to the current solution \mathcal{S}^{now} depicted in Fig. 3.4(a). This solution meets both the timing and reliability constraints. The cost function value (Eq.3.9) is equal to the energy value, 1126.51, i.e., the last two terms in Eq.3.9 are zero. However, \mathcal{S}^{now} is not the best solution so far. The cost of the best-so-far solution,

938.37, shown in the table of Fig. 3.4(a), is lower than the cost of S^{now} . In Fig. 3.4(a), we also show the tabu list, where $length_tl = 3$ for this small example.

In S^{now} , tasks $\tau_1, \tau_2,$ and τ_3 are mapped on N_1 , while tasks τ'_1, τ'_2 and τ_4 are mapped on N_2 . Tasks τ_1, τ_2 and τ_4 are assigned to the maximum operating mode, L_3 , while tasks τ'_1, τ'_2 and τ_3 are assign to L_2 . Fig. 3.4(b)–(e) present four neighbors of S^{now} obtained by applying \mathcal{M} -moves and \mathcal{L} -moves to S^{now} . These neighbors together with other neighboring solutions, are placed in the candidate set C .

The solution S_b^{new} has been obtained from S^{now} by remapping τ_4 from N_2 to N_1 , changing its mode in N_1 to L_2 , and changing the mode of τ'_2 in N_2 to L_1 . We denote with a thick border a task that has been involved in an \mathcal{M} -move, and with a shadow a task that has been involved in an \mathcal{L} -move. Under each of the neighbor in Fig. 3.4(b)–(e), we also present the cost of that solution and its energy consumption. S_b^{new} is schedulable and

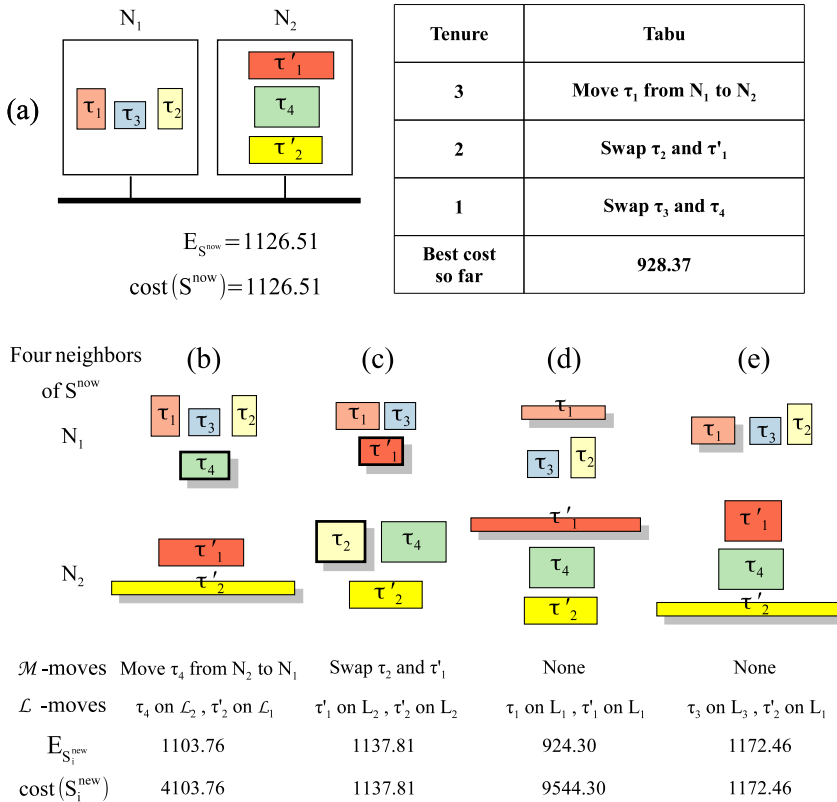


Figure 3.4: Moves in Tabu Search

reduces the energy compared to \mathcal{S}^{now} . However, \mathcal{S}_b^{new} does not meet the reliability goal, and hence the second term in Eq.3.9 is penalized (we have used $W_R = 10^7$). Hence, the cost \mathcal{S}_b^{new} is 4103.76, which is much larger than the cost of \mathcal{S}^{now} .

The solution \mathcal{S}_c^{new} in Fig. 3.4(c) has been obtained from \mathcal{S}^{now} by swapping the PEs of τ_2 and τ'_1 and by changing their operating modes of τ_2 and τ'_1 to L_2 . In Fig. 3.4(e), we change the mode of τ_3 to L_3 and τ'_2 to L_1 , compared to \mathcal{S}^{now} . The solution \mathcal{S}_d^{new} in Fig. 3.4(d) is not schedulable, hence we penalized the third term in Eq.9 (with $W_S = 200$), resulting in a cost of 9544.30.

We have to select a new solution \mathcal{S}^{new} from which to continue the search. The solution \mathcal{S}_c^{new} in Fig. 3.4(c) has the best cost function among the neighbors considered, and also improves on \mathcal{S}^{now} . However, the move "swap τ_2 and τ'_1 " which has involved in creating this solution is tabu, i.e., it is on the tabu list in Fig. 3.4(a). The tabu status of \mathcal{S}_c^{new} can not be aspirated since the cost of \mathcal{S}_c^{new} is not better than the best-so-far cost of 938.37. Solutions \mathcal{S}_b^{new} and \mathcal{S}_d^{new} do not improve the cost function because they are penalized. Solution \mathcal{S}_e^{new} from Fig. 3.4(e) improves on the cost function of \mathcal{S}^{now} and it is not tabu. Hence, \mathcal{S}_e^{new} is selected as the new solution \mathcal{S}^{now} and the exploration continues from this solution.

3.6 Online Scheduling

As mentioned, to guarantee the schedulability and reliability constraints, the offline synthesis algorithm presented in the previous section has to assume that the tasks will execute up to their WCET and all the replicas have to be executed for fault tolerance. However, during runtime, tasks will execute for a small fraction of their WCET [EY97] and in the case the critical tasks do not experience faults, we can avoid starting (or we could terminate) the replicas. In this section, we propose an online scheduling approach which uses the reclaimed slack at runtime to further reduce the energy consumption by scaling down the operating voltage and frequency.

The area of real-time scheduling for DVFS multiprocessors has received a lot of attention, and there are many online DVFS algorithms proposed in the literature (see [CK07] for a survey). These algorithms have been proposed for both uniprocessor and multiprocessor systems. Several scheduling policies have been targeted, such as aperiodic scheduling, and periodic preemptive scheduling such as Rate Monotonic and Earliest Deadline First, and periodic non-preemptive scheduling, such as Static cyclic Scheduling. Energy efficiency can be achieved through strategies such as slack reclamation [Gru01, AMMA01, PS01, CYK06, WMWZ12], DVS with power-down [ISG03, QNHM04], by using stochastic information on the expected workload [GK03, LS04], or by taking into account the leakage current [CHK06].

In general, there are two kinds of slack reclamation strategies. One is collecting all the possible slack, and planing for all the jobs that have not yet started to execute. We denote it as "global" slack reclamation. Another, is collecting only the current available slack, and planning for the current job that is going to execute. We can call it "greedy" slack reclamation.

The multiprocessor strategies decide at runtime not only the operating mode for each task, but also the task mapping to PEs. In our online scheduling, we consider the task mapping fixed, as determined at design time by our offline synthesis strategy, and we are interested online in deciding only the operating mode for executing the jobs of tasks. Our intention is not to propose a novel runtime algorithm. Instead, we are interested to determine if we can obtain further energy savings at runtime without negatively impacting the imposed reliability requirement, by further lowering the operating voltage and frequency. Hence, we have decided to use a greedy slack reclamation scheme as the starting point of our online energy/reliability trade-off algorithm.

Our proposed Online Voltage and Frequency Scaling (OVFS) algorithm is presented in Algorithm 2. As mentioned, we use a fixed-priority preemptive scheduling policy. Thus the scheduler in each PE will select the job J_{ij} with the highest priority on that PE to be executed. OVFS is called before a job J_{ij} starts to execute and returns the operating mode ℓ for executing the job J_{ij} .

The slack management strategy used in OVFS is similar at the core with the greedy slack distribution from [Gru01] and the D-TDVS presented in [WMWZ12]. Both of these employ slack levels, such that a slack of certain priority generates slack at that level (if it finishes earlier than its WCET) but consumes slack produced by higher priority tasks (starting with the highest priority slack). Furthermore, idle times are also consuming the slack in a similar way, thus unused slack degrades and finally disappears. So far the approach is a standard DVS online method that keeps the response time guarantees computed by the offline analysis. However, to take into account reliability, two improvements are added. First, when a task with replicas finishes successfully, the replicas will be also finished (some even before starting), generating a certain amount of slack at their respective level. Second, the operating speed for the newly scheduled task is limited not only by its deadline, but also by the reliability goal, which means that tasks may actually be required to execute slightly faster than in a classic DVS approach.

Line 2 in Algorithm 2 computes the slack H_i available to a task with priority i , which is about to start executing. This slack may be partially or completely used up by choosing a lower operating speed for this task (lines 6-9), such that the WCET at the lower speed (C_i^ℓ) does not consume more than the available slack on top of the WCET C_i^{MVFS} precomputed offline by MVFS. Furthermore, the reliability with the new mode $\phi_S^{\ell-1}$ should maintain the reliability goal ϕ_g .

Algorithm 2 OVFS

```

1: before  $J_i$  starts to execute
2: calculate the available slack  $H_i$  which can be used for  $J_{ij}$ 
3: retrieve the pre-computed  $C_i^{MVFS}$ 
4: start from  $\ell$ , the mode computed by MVFS
5: calculate  $\phi_S^{\ell-1}$  (the GSFR when  $J_i$  is executed in operating mode  $(\ell - 1)$ )
6: while  $(H_i + C_i^{MVFS}) \geq C_i^{\ell-1}$  and  $\phi_S^{\ell-1} \leq \phi_g$  do
7:    $\ell = \ell - 1$ 
8:   retrieve the pre-computed  $C_i^{\ell-1}$ 
9:   calculate  $\phi_S^{\ell-1}$ 
10: end while
11: return the operating mode  $\ell$  to execute  $J_i$ 

```

When lowering the operating mode for the next job, our DVFS algorithm does not only guarantee the schedulability, but also guarantees that the imposed reliability constraint is met. Recall that according to Eq.3.7, lowering the voltage and frequency will increase the transient fault rate, consequently lowering the reliability R_S of system. The assumption is that if R_S becomes smaller than R_g , the system may need more replicas for the critical tasks due to the increased transient faults. Thus, unless $R_S \geq R_g$, the system is not fault-tolerant.

During our proposed offline synthesis approach MVFS, the reliability goal is enforced by checking the reliability R_S of every S (the second term in the cost function from Eq.3.9) over the mission duration T_S . However, at runtime, we cannot use the system reliability R_S as calculated by Eq.3.4. Since reliability is a function of time, it can lead to a “so far so good” situation where the operating mode is lowered aggressively initially for all jobs because we are well-above the goal R_g . A similar situation is shown in [AGK11] where using the reliability as a metric would result in too much replication for the tasks placed forwards the end of a schedule and no replication for the initial tasks. Therefore, for the purpose of our online scheduling approach, similar to [AGK11], we use a “reliability per time unit” metric, Global System Failure Rate, called GSFR, for measuring the reliability [GK09]. The GSFR ϕ_S of a system is defined as:

$$\phi_S = \frac{-\log(R_S)}{U_S} \quad (3.11)$$

where \log is using the natural logarithm. U_S is the sum of the execution times for all the jobs executed so far, including the current job J_{ij} to be scheduled, considering the pre-calculated operating mode Λ_ℓ , and R_S is the reliability at the time when job J_{ij} will finish. Calculation examples of R_S will be given later for the cases shown in Fig. 3.5.

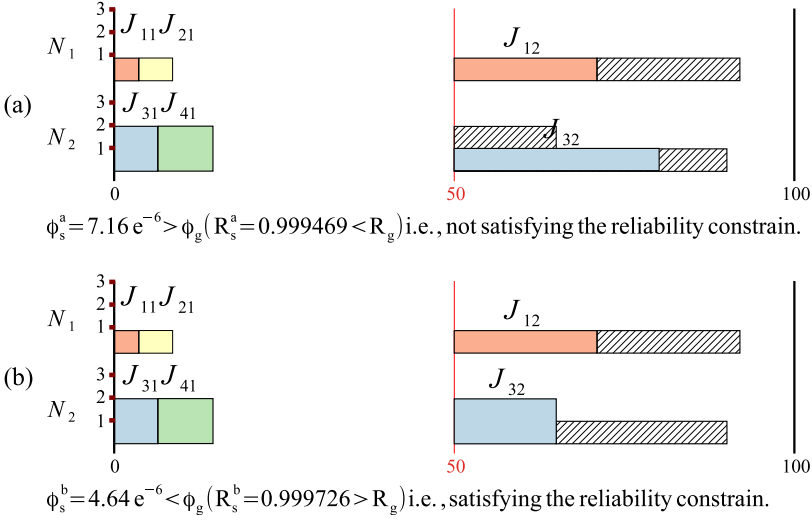


Figure 3.5: OVFS example

To enforce the reliability goal R_g , we convert it to a GSFR goal ϕ_g and compare it to a calculated GSFR ϕ_S . We define ϕ_g as $-\log(R_g)/U$, where R_g is the reliability over the system cycle T_{cycle} and U is the sum of the WCET of all jobs within T_{cycle} . We consider the system satisfies the reliability requirement at runtime if the current $\phi_S \leq \phi_g$. Note that a smaller GSFR means higher reliability. In line 5 of Algorithm 2, this reliability check is expressed as $\phi_S^{\ell-1} \leq \phi_g$. For instance, $\phi_g = 4.65 \times 10^{-6}$ for our motivational example of Fig.3.1.

Let us illustrate how OVFS works in Fig. 3.5 by using the example from Fig. 3.3(b), where we have the application and architecture from Fig. 3.1, and we start with the offline solution produced by MVFS in Fig. 3.2(b). Let us assume that we have reached to the point we have to decide the mode ℓ of executing J_{32} at $t = 50$. The calculation of R_S for J_{32} is expressed as $R_{32}^\ell = \prod_{J_{ij} \in t_{32}} R_{ij}^\ell$, where $J_{ij} \in t_{32}$ means we consider all finished and started jobs when J_{ij} will finish. For those finished jobs, we use their actual execution time while for the started but not yet finished jobs, we use their WCETs. Thus, $R_{32}^1 = R_{11}^1 \cdot R_{21}^1 \cdot R_{12}^1 \cdot R_{31}^2 \cdot R_{41}^2 \cdot R_{32}^1 = 0.999469$ for the case J_{32} will be executed on mode L_1 Fig. 3.5(a), while $R_{32}^2 = R_{11}^1 \cdot R_{21}^1 \cdot R_{12}^1 \cdot R_{31}^2 \cdot R_{41}^2 \cdot R_{32}^2 = 0.999726$ for the case J_{32} will be executed on mode L_2 Fig. 3.5(b).

From the point of view of the schedulability, the slack (hashed rectangles) is enough to scale down J_{32} to L_1 , as we can see in Fig. 3.5(a). However, in this case, $\phi_s^a > \phi_g$ ($R_s^a = 0.999469 < R_g$), i.e., the reliability goal would be missed. Hence, we keep running J_{32} by the offline calculated mode L_2 , as shown in Fig. 3.5(b), to preserve the system reliability.

Table 3.1: Three types of PEs

Fast PE			Medium PE			Slow PE		
Freq. [MHz]	Volt. [V]	Power [W]	Freq. [MHz]	Volt. [V]	Power [W]	Freq. [MHz]	Volt. [V]	Power [W]
500	1.2	9.2	300	1.4	4.3	133	1.1	1.36
600	1.25	12	350	1.5	5.6	166	1.2	1.9
700	1.3	15.1	400	1.6	7.1	200	1.3	2.58
800	1.35	18.6	450	1.7	8.95	233	1.4	3.4
1000	1.4	25	500	1.8	11.4	266	1.5	4.4
$\alpha = -6.5, \beta = -0.039$			$\alpha = -8.9, \beta = -0.038$			$\alpha = -6.5, \beta = -0.039$		

3.7 Experimental Results

We have implemented our proposed algorithms in C++ and run them on an Intel Core i7 CPU 920 (2.67 GHz, 4 GB RAM, and Windows 7) computer. The offline synthesis MVFS is evaluated in Section 3.7.1 and the online scheduling OVFS is evaluated in Section 3.7.2. For evaluation, we used ten synthetic benchmarks and five real-life case studies. In all benchmarks, a quarter of tasks were considered critical and for each critical task we introduced one redundancy. We have used only “sample” type of communication for messages, see Section 3.2. We used three types of PEs, described in Table 3.1. The minimum failure rate (when system runs at the maximum speed mode) of all PEs was set to $\lambda_0 = 10^{-7}$ and the shape parameter μ^h was set to 1.

For comparison purposes, we have derived a baseline offline solution \mathcal{S}_0 . In \mathcal{S}_0 , all jobs execute in the maximum speed operating mode, and the mapping is optimized in terms of energy consumption, under schedulability and reliability goals. This is the same baseline solution used in the motivational example. For the experiments, \mathcal{S}_0 is determined by performing very long runs with our proposed MVFS, where we use only \mathcal{M} -moves and we do not use \mathcal{L} -moves (as mentioned, all tasks are set to the highest operating mode). The resulted energy consumption and reliability in \mathcal{S}_0 are denoted with E_0 and R_0 , respectively. The reliability goal R_g is set such that we accept a probability of failure which is ten times larger than R_0 , i.e., $R_g = 1 - 10(1 - R_0)$.

3.7.1 Offline Synthesis Evaluation

We tuned the Tabu Search parameters of MVFS such that no improvements were seen for longer run times, thus leading to near optimal solutions. The terminating condition for MVFS used in the experiments is a time limit. We have used time limits between 5 minutes to 5 hours, depending on the size of the benchmark.

In the first experiment we wanted to evaluate the quality of MVFS as the systems become larger. Table 3.2 presents the experimental setup details and the results. We used five synthetic benchmarks of 10 to 105 tasks (including replicated tasks) mapped on architectures with 2 to 6 different types of PEs. The details of each test case are presented in columns 1–4. The results obtained with MVFS are presented in the last two columns. The increase in the failure probability θ was calculated using Eq. 3.8 and had to be smaller or equal to 10 in order to meet the reliability goal R_g . Alongside the MVFS results, we also present the results obtained with $MVFS^-$. This is an implementation which minimizes the energy, but without any concern for reliability (the reliability constraint, the second term in the cost function of Eq. 3.9, is removed). As we can see from Table 3.2, columns 5 and 6, minimizing the energy without considering reliability leads to a dramatic increase in the probability of failure, which increases more than 100 times in most cases. However, our MVFS approach is able to keep the system reliability R_S within the specified R_g , without a significant loss in energy savings (last column) compared to $MVFS^-$ saved energy (column 6). The saved energy percentage, denoted by E_Δ , in the column 6 and 8 of the table, are relative to the energy E_0 of the reference solution S_0 , i.e., $E_\Delta = (E_0 - E_S)/E_0$.

In the second experiment we wanted to determine the ability of MVFS to find good quality solutions as the utilization of the system increases. The experimental setup and the results obtained are presented in Table 3.3. We used a synthetic benchmark with 25 tasks (20 tasks and 5 replicated tasks) mapped on a heterogeneous architecture with 3 different types of PEs. We varied the execution times resulting in five cases corresponding to utilization (column 5), from 27.21% to 71.98%. More energy can be saved in the less utilized systems since more slack could be used for lowering operating voltage and frequency without missing the deadlines. As expected, using MVFS is especially important where the energy saving potential is greater, because reliability is significantly impaired without it.

For example, in the test set 1 in Table 3.3, the probability of failure increases 198 times when $MVFS^-$ does not use the reliability constraint. Using MVFS, we keep the reliability within R_g (i.e., $\theta \leq 10$) with a loss of only $29.57 - 25.00 = 4.57\%$ in energy

Table 3.2: Offline synthesis: Synthetic benchmarks with different system sizes

Test Set	Numbers of			$MVFS^-$		MVFS	
	PEs	Orig. Tasks	Repl. Tasks	θ [times]	E_Δ [%]	θ [times]	E_Δ [%]
1	2	8	2	166	28.23	10	24.64
2	4	31	8	112	28.56	10	25.28
3	4	42	11	137	30.47	10	26.04
4	6	63	16	104	25.92	10	21.92
5	6	84	21	57	22.78	10	20.57

Table 3.3: Offline Synthesis: Synthetic benchmarks with varying initial utilization

Test Set	Numbers of			Initial Util. [%]	MVFS ⁻		MVFS	
	PEs	Orig. Tasks	Repl. Tasks		θ [times]	E_{Δ} [%]	θ [times]	E_{Δ} [%]
1	3	20	5	27.21	198	29.57	10	25.00
2	3	20	5	41.00	121	26.55	8	23.02
3	3	20	5	52.73	101	25.26	9	21.05
4	3	20	5	61.56	72	22.94	10	20.09
5	3	20	5	71.98	7	12.76	7	12.76

Table 3.4: Offline synthesis: Real-life benchmarks

Benchmarks	Numbers of			MVFS ⁻		MVFS	
	PEs	Orig. Tasks	Repl. Tasks	θ [times]	E_{Δ} [%]	θ [times]	E_{Δ} [%]
networking-cords	2	13	3	141	28.01	10	20.49
auto-indust-cords	4	24	6	77	22.68	10	17.87
telecom-cords	4	30	8	129	28.16	9	19.57
above three together	6	67	17	64	15.26	10	13.86
Smart-phone	2	61	16	60	18.71	9	15.23

savings. Trading-off a small percentage of energy savings, our offline MVFS approach is able to guarantee the reliability constraints.

Finally, we evaluated MVFS on real-life case studies. Five benchmarks were selected from the Embedded System Synthesis Benchmark Suite (E3S), version 0.9 [Dic08], and Smart-Phone Benchmarks [SAHE04]². The experimental setup details and the results obtained are presented in Table 3.4. As can be seen, this evaluation confirms the results obtained from the synthetic benchmarks. This means that by using MVFS we are able to eliminate the negative impact of energy minimization on reliability with minimal loss in energy savings.

3.7.2 Online Scheduling Evaluation

To evaluate our proposed OVFS, we have built a simulator which simulates the execution of jobs, according to fixed-priority preemptive scheduling, and the fault occurrences. We have used as input to our simulator the test sets of synthetic benchmarks (Table 4.3) and real-life benchmarks (Table 3.4). We have considered a mission time T_S of 30 hours for each test set.

²We only used the applications of MP3 decoder, GSM decoder, JPEG encoder and JPEG decoder.

Table 3.5: Online scheduling: Synthetic Benchmarks

Test Set	Numbers of			OVFS ⁻	OVFS
	PEs	Orig. Tasks	Repl. Tasks	\tilde{E}_Δ [%]	\bar{E}_Δ [%]
1	2	8	2	79.74	77.68
2	4	31	8	83.91	81.91
3	5	42	11	82.09	79.90
4	6	63	16	80.54	78.42
5	8	84	21	81.69	78.51

Table 3.5 and 3.6 present the online simulation results of the synthetic and real-life benchmarks, respectively. The details of each test case are presented in columns 1 – 4. Similar to the evaluation of our offline approach MVFS, we compare two online approaches, OVFS and OVFS⁻ in terms of their energy savings and the impact on reliability. OVFS is our proposed online scheduling approach presented in Section 3.6. Unlike OVFS, OVFS⁻ does not enforce the reliability goal ϕ_g when reducing the operating voltage and frequency at runtime, i.e., it does not enforce $\phi_S \leq \phi_g$.

For each simulation, we use as starting point the offline solution determined by MVFS in the previous section. We report the average percentage of energy savings per application cycle, denoted by \tilde{E}_Δ , for both OVFS⁻ (column 5) and OVFS (column 6), which we compare the average energy consumption in the online scheduling per application cycle, $\tilde{E}_S^{\text{online}}$ (recall that we simulate the program $T_S = 30$ hours) with E_0 that S_0 has been evaluated for T_{cycle} in the offline synthesis.

As we can see from the resulting Table 3.5, by reclaiming slack at runtime, online algorithm for both implementations leads to significant energy savings compared to the offline reference solution S_0 , i.e., 76% to 83%. Compared to OVFS⁻ which do not impose ϕ_g as a reliability constraint, smaller energy savings (within 5%) are reported in the OVFS columns, however the reliability is preserved. The increase in failure probability varies with OVFS in every application cycle, thus it is not reported in the tables.

The results show that, at runtime, it is possible to exploit the available slack for further energy savings compared to the offline synthesis solution, without negatively impairing the reliability. This conclusion is also supported by the results obtained using the real-life benchmarks, which are reported in Table 3.6.

Table 3.6: Online scheduling: Real-life benchmarks

Benchmarks	Numbers of			OVFS ⁻	OVFS
	PEs	Orig. Tasks	Repl. Tasks	\bar{E}_Δ [%]	\bar{E}_Δ [%]
networking-cords	2	13	3	80.48	78.07
auto-indust-cords	4	24	6	77.54	75.45
telecom-cords	4	30	8	83.34	80.80
above three together	6	67	17	78.84	75.12
Smart-phone	2	61	16	76.40	72.72

3.8 Conclusions

In this paper we addressed the mapping, voltage and frequency scaling for fault-tolerant hard real-time applications mapped on distributed embedded systems where tasks and messages are scheduled using an event-driven scheduling policy. We captured the effect of voltage and frequency scaling on system reliability, and we showed that if the supply voltage and the operating frequency are lowered to reduce energy consumption, reliability is significantly reduced. That is why we have proposed both offline and online approaches that can take reliability into account when performing voltage and frequency scaling.

We have prepared an offline synthesis approach, MVFS, based on a Tabu Search meta-heuristic, which decides the mapping and operating mode for each task such that the energy is reduced and the schedulability and reliability constraints are satisfied. We have also proposed an online scheduling approach, OVFS, which considers the mapping determined by MVFS and decides at runtime the operating mode for each job such that the energy is further reduced. OVFS also guarantees the timing and reliability constraints.

As the experimental results show, our synthesis approaches are able to produce energy efficient implementations which are both schedulable and fault-tolerant. By carefully deciding the mapping, operating voltage and frequency of each task, we showed that it is possible to eliminate the negative impact of voltage and frequency scaling on reliability with minimal decrease in energy savings.

CHAPTER 4

Paper C: Design for Robustness and Flexibility of Real-time Distributed Applications during the Early Design Phases

We are interested in mapping hard real-time applications on distributed heterogeneous architectures. An application is modeled as a set of tasks, and we consider a fixed-priority preemptive scheduling policy. We target the early design phases, when decisions have a high impact on the subsequent implementation choices. However, due to a lack of information, the early design phases are characterized by uncertainties, e.g., in the *worst-case execution times (wcets)* or in the functionality requirements. We model uncertainties in the *wcets* using the “percentile method”. The uncertainties in the functionality requirements are captured using “future scenarios”, which are task sets that model functionality likely to be added in the future. In this context, we derive a mapping of tasks in the application, such that the resulted implementation is both robust and flexible. Robust means that the application has a high chance of being schedulable, considering the *wcet* uncertainties, whereas a flexible mapping has a high chance to successfully accom-

modate the future scenarios. We propose a Genetic Algorithm-based approach to solve this optimization problem. We also show how this problem can be extended to consider the architecture selection: deciding what hardware components to use in the architecture. In this context, we consider the uncertainties related to hardware component costs. Extensive experiments show the importance of taking into account the uncertainties during the early design phases.

4.1 Introduction

There is a lot of research on embedded systems design [LP05], but very few researchers have addressed the early design phases. The decisions taken during the early design phases, e.g., architecture selection and task mapping, have a high impact on the subsequent implementation choices [Axe06]. However, early phases are characterized by many uncertainties, e.g., in terms of the hardware components available, functionality that has to be implemented and attributes such as the *worst-case execution time (wcet)*.

We address hard real-time applications, modeled as a set of tasks, where timing constraints are of utmost importance. We are interested in tackling uncertainties in the functionality requirements and the *wcets* of tasks. We model the uncertainties in the *wcets* using the “percentile method” [Axe05], which captures the *wcet* of a task by two values, the 50th and the 90th percentile. These numbers are chosen by the designers based on the best available information and their experience.

Today, most systems are engineered in an evolutionary fashion: introducing a new version of an existing product, introducing new features—possibly as part of a planned evolution of a product line, performing a design-iteration, etc. Functionality requirements often change during these iterations. For example, new tasks may be introduced to update the existing functionality or add a new feature, etc. Several approaches to generating realistic product scenarios and how these product scenarios should be prioritized, are discussed in [BE06]. Thus, we capture the uncertainties in functionality requirements using “future scenarios” [BE06], which are task sets that model functionalities likely to be added in the future.

We initially assume that the hardware architecture is fixed, and we focus on deciding the mapping of tasks to processing elements (PEs), such that the application is schedulable, even considering the uncertainties. A straightforward solution to tackle uncertainties is to over-design the system, e.g., to build a lot of spare capacity, but this is often prohibitively expensive. As an alternative, researchers have proposed *adaptive systems* [SPM09], which change at runtime their configuration (e.g., re-mapping tasks using task migration) in response to changes in the requirements, execution times, environment, etc. However, many hard real-time applications are safety critical, where on-

line task migration is not feasible, and an offline reconfiguration, e.g., task re-mapping, may be very costly. Hence, we want to derive, early on, a mapping of tasks to PEs, which is both *robust* and *flexible*. In our case, robust means that the application has a high chance of being schedulable, considering the *wcet* uncertainties, whereas a flexible mapping has a high chance to successfully accommodate future scenarios.

For time-triggered systems using static-cyclic scheduling, researchers have proposed approaches for the synthesis of flexible schedules, which can accommodate future changes. In [PIEP09], the future applications are captured using a set of possible *wcets* and their probability distributions, and the flexibility is measured as the likelihood of successfully adding new functionality in the future. In [ZCP⁺05], the flexibility is defined as both “extensibility”, i.e., the maximum increase in the *wcet* of a task that can be handled without rescheduling, and “scalability”, i.e., the maximum *wcet* of a new task that a schedule can accommodate without change. The work in [GZDNBH10] uses the info-gap decision theory to synthesize robust FlexRay bus schedules, considering uncertainties in design parameters, such as the size of a message.

In this paper we consider that the tasks are scheduled using *fixed-priority preemptive scheduling* (*fpps*). In this context, robustness has been addressed using “sensitivity analysis” [RHE06], where the attributes of an implementation, e.g., worst-case response times, are evaluated against changes in system properties, e.g., *wcets*. In [Axe05], the mapping is fixed, and the uncertainties in the *wcets* are captured using the “percentile method”. The author proposes a Monte Carlo simulation approach to evaluate the likelihood that tasks will meet their deadlines. Other researchers [LGT10] quantify the robustness in terms of a “revision cost”, and they aim to provide robust designs, which minimize the revision cost.

Regarding flexibility, [PEP02] proposes a mapping technique to increase the chance of successfully accommodating future applications. The future applications are captured similar to [PIEP09], using probabilities for *wcet* values. The approach in [HTRE02] defines flexibility in terms of the amount of functionality that the design is able to implement, and proposes a flexibility vs. cost trade-off model to search for Pareto-optimal solutions. In [BE06], researchers model the uncertainty in functionality requirements using scenarios, i.e., prioritized task sets, and derive an architecture and a mapping of tasks such that the flexibility is maximized. The flexibility is defined as the likelihood of all tasks being schedulable when adding a scenario to the existing mapping.

In this paper we are initially interested to derive a robust and flexible mapping solution during early design phases, for a given architecture and taking into account the uncertainties. The problem is defined in Section 4.3, and the uncertainty models are presented in Section 4.2. Section 4.5 presents a motivational example which shows the importance of considering the uncertainties. In order to address both robustness and flexibility simultaneously, we propose a Genetic Algorithm-based approach to search for a *Pareto-front* of solutions (Section 4.6). The evaluation of the proposed approach

is presented in Section 4.7. In Section 4.8, we extend the problem to consider also the architecture selection. We are interested to determine that hardware architecture and mapping which maximizes the robustness and flexibility as defined earlier. In addition, derived architecture should have a high chance of meeting an imposed cost budget. In the last section, we draw the conclusion of our work.

4.2 System Model

In this paper, a system is composed of software applications, modeled as a set of tasks, and a hardware architecture consisting of a set \mathcal{N} of PEs, interconnected by a communication channel. The mapping of a task τ_i to a PE N_j is captured by a mapping function: $\mathcal{M}(\tau_i) = N_j$. This mapping is not yet known and will be decided by our proposed approach. For each task τ_i , we assume that the period T_i and deadline D_i are known and given such that $D_i \leq T_i$.

Tasks communicate using messages. We know the size s_{m_i} and the priority of each message m_i . Our model captures two types of communication: sampling and queuing. A *sampling* communication has a buffer storage for a single message; arriving messages overwrite the buffer, and reading does not remove the message, i.e., it can be read repeatedly. A *queuing* communication uses a buffer that can store several messages, and works as a FIFO queue. A reader task will block if the buffer is empty and a writer task will block if the buffer is full. We assume that the buffer size have been determined such that there is no overflow or underflow [MEP06].

Tasks are scheduled using fixed-priority preemptive scheduling. Researchers have shown [TSPS12a] how realistic protocols such as TTEthernet can be taken into account during the analysis and synthesis. However, for simplicity, in this paper we consider that the PEs are interconnected by a shared bus, which uses a fixed-priority dynamic scheduling scheme and assumes that the messages are non-preemptible. This is similar to how a widespread bus protocol such as the Controller Area Network [Bos91] works.

4.2.1 Early Design Stages

There are many life cycle models used in the industry [Est07], such as waterfall models [Boe88] and V-models [FHKS09], but in principle they all have the following main stages: concept development, engineering development and post-development, see the top part of Fig. 4.1 [KSSB11]. During concept development, system engineers perform a needs analysis, do concept exploration and definition. Engineering development consists of advanced development, engineering design, integration and evaluation. Then,

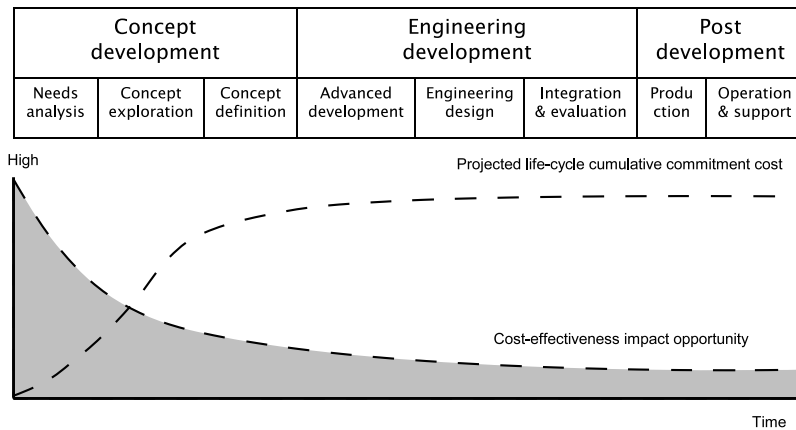


Figure 4.1: Life Cycles of Systems Engineering

in post-development, the mass-production from the prototype starts, followed by the operation and maintenance of the system.

Experience from completed system design and development projects has indicated that, the cost spent for the concept development stage accounts for about 20% of the total cost. However, 80% of the cumulative cost of the system is committed already in this stage [Bue11]. Decisions made in the early design stages not only have a high impact on the subsequent implementation choices, but also have substantially negative impacts on the total cost of the system, since it is costly and time-consuming to modify and correct a decided design to other alternative designs in the engineering development stage. i.e., early decisions have a high cost-effectiveness impact opportunity.

The bottom part Fig. 4.1 [SR11] shows the projected committed cumulative cost, and the impact opportunity of a design decision over time. The conclusion is that we should spent more effort on the early decisions, in order to increase the chance of completing the projects on time and successfully.

There is a lot of research on embedded systems design [Mar11, GAGS09], but very few researchers have addressed the early design stages. The challenge is that early design stages are characterized by many uncertainties.

Uncertainty in the context of systems engineering refers to the inability to determine precisely the state or attributes of a system. It can be caused by incomplete knowledge or by stochastic variability. A detailed discussion on the taxonomy of uncertainty is available in [Hai11].

Uncertainties in the early design stages need to be quantified, such that *risks*, defined as the probability of not reaching design targets, of different design alternatives can be estimated for making early design decisions. The next two subsections present how we model the uncertainties in the *wcet* and the functionality, respectively.

4.2.2 Modeling WCET Uncertainties

In this paper, we are not interested in modeling the variability of the execution time e_i of a task τ_i , but in modeling the uncertainties of the *wcet* c_i . The variability of e_i is typically captured by a probability mass function [KDB⁺05] and is due to, for example, the variations in the input data of tasks or the speculative features of modern processors.

In our case, the uncertainties in a *wcet* c_i come from the lack of information during the early design stages, when, for example, the algorithm used to implement a task τ_i or the parameters of the architecture are not yet known. Similar to [Axe05], we use the “percentile method” to model the *wcet* uncertainties. Thus, we capture the *wcet* of a task by two values, the 50th and 90th percentiles. This means that in 50% of the possible future implementations, a task will have a *wcet* smaller or equal to the 50th percentile value, while in the majority of the cases, i.e., 90%, the *wcet* will not exceed the 90th percentile value.

Table 4.1 presents an example in which four tasks, τ_1 to τ_4 , can be mapped on two PEs, N_1 and N_2 . For each task τ_i and its mapping on each PE N_j , two *wcet* values are given, i.e., the 50th and 90th percentile, respectively. The more information is available about a task and the PEs where it can potentially be mapped, the smaller the difference between the two percentiles. A *wcet* can also be a fixed value, e.g., for legacy tasks mapped on legacy PEs. In our example, τ_4 is an updated task, thus designers believe that its 90th percentile is only 20% larger than its 50th percentile. For τ_3 , this difference is 50%. However, tasks τ_1 and τ_2 are new tasks to be introduced, hence designers have lower confidence and thus their 90th percentiles are twice the 50th percentiles.

Table 4.1: Uncertainties in *wcets*

Tasks	N_1		N_2		$T_i = D_i$
	50 th	90 th	50 th	90 th	
τ_1	10	20	15	30	50
τ_2	25	50	37.5	75	100
τ_3	40	60	60	90	150
τ_4	60	72	90	108	300

We use a Gumbel distribution, with the *cumulative distribution function (cdf)* defined as [KN00]: $P(c_i \leq x) = e^{-e^{-\frac{x-\mu}{\beta}}}$, where P is the probability that the *wcet* c_i will have a value smaller or equal to x . Using the two percentile values, corresponding to 0.5 and 0.9 probability, we can determine the distribution parameters μ and β , i.e., determine the *cdf* of the *wcet*. For example, for τ_1 mapped on N_1 in Table 4.1, we have $\mu = 8.05$ and $\beta = 5.31$. Knowing the *cdf* for a task τ_i , we can also determine its *probability density function (pdf)* σ_i .

4.2.3 Modeling Functionality Uncertainties

Additionally, product requirements often change during later design and development phases. We use the approach from [BE06] to describe an application as one baseline functionality, S_0 , and a set of future scenarios, $S_f = \{S_1, S_2, \dots\}$. Each future scenario is associated a weight w_i , which reflects the probability of this future scenario becoming a reality.

Let us consider the example in Table 4.2, which includes one baseline functionality S_0 (tasks τ_1 to τ_4 from Table 4.1), and four future scenarios S_1 to S_4 . We consider an architecture of two PEs, N_1 and N_2 (the same as in Table 4.1). S_1 replaces τ_1 with τ_5 due to a functionality update. S_2 introduces τ_6 for enhancing the application performance. In S_3 , a new application is added, which is modeled by τ_7 and τ_8 . S_4 is the combination of S_1 and S_2 , which captures the case when both S_1 and S_2 happen at

Table 4.2: Modeling uncertainties in functionality requirements

Tasks	N_1		N_2		$T_i = D_i$
	50 th	90 th	50 th	90 th	
S_0 : Example in Table 4.1					
$S_1 = S_0 \setminus \tau_1 \cup \tau_5$ ($w_1 = 0.8$)					
τ_5	15	30	22.5	45	50
$S_2 = S_0 \cup \tau_6$ ($w_2 = 0.4$)					
τ_6	25	50	37.5	75	100
$S_3 = S_0 \cup \tau_7 \cup \tau_8$ ($w_3 = 0.6$)					
τ_7	30	60	45	90	300
τ_8	50	75	75	102.5	300
$S_4 = S_0 \setminus \tau_1 \cup \tau_5 \cup \tau_6$ ($w_4 = 0.2$)					
τ_5	15	30	22.5	45	50
τ_6	25	50	37.5	75	100

the same time. Next to each scenario S_i , we also specify its weight w_i . These scenarios, and their associated weights, are determined using the methods presented in [BE06].

4.3 Problem Formulation

As an input to our problem, we have the hardware architecture \mathcal{N} , the baseline functionality S_0 and the set of future scenarios S_f . For each task, we know the two *wcrt* percentile values, on every PE where it is considered for mapping.

We are interested to determine the mapping M_0 of the baseline functionality S_0 on the given architecture \mathcal{N} , such that the robustness and flexibility of M_0 is maximized. These two metrics will be formally defined in the next subsection. A mapping M_0 is robust if the tasks in S_0 have a high chance of being schedulable. M_0 is flexible if it has a high chance to successfully accommodate the future scenarios from S_f , such that they are also likely to be schedulable. This is a two-objective optimization problem (robustness and flexibility). Our optimization strategy, presented in Section 4.6, will produce a Pareto-front of solutions.

We target safety-critical hard real-time applications, so we consider that M_0 is fixed when adding a future scenario. Our optimization strategy will produce the mappings M_i of future scenarios S_i , as a byproduct of evaluating the flexibility of M_0 . In the later design stages, when a scenario S_i has become a reality, we use our proposed mapping optimization strategy to decide the mapping M_i of S_i , while keeping the mapping of tasks in S_0 , decided during the early design phases, fixed.

4.3.1 Robustness and Flexibility

We use the “degree of schedulability” to characterize the schedulability of a given mapping alternative M ,

$$r_M = \begin{cases} \ell_1 = \sum_i \max(0, r_i - D_i) & \text{if } \ell_1 > 0 \\ \ell_2 = \sum_i (r_i - D_i) & \text{if } \ell_1 = 0 \end{cases} \quad (4.1)$$

where r_i is the *worst-case response time (wcrt)* of a task τ_i and D_i is its deadline. If a mapping is not schedulable, there exists at least one r_i greater than the deadline D_i , therefore the term ℓ_1 of the function will be positive. In this case r_M is equal to ℓ_1 . However, if a mapping is schedulable, then each r_i is smaller than its corresponding

deadline D_i . In that case $\ell_1 = 0$ and we use ℓ_2 as the r_M , to be able to differentiate between two mapping alternatives, both leading to feasible schedules. $r_M \leq 0$ means the mapping is schedulable. A larger negative value of r_M indicates the mapping is "more schedulable", i.e., the *wcrts* are smaller.

Note that r_M is a stochastic variable, since it is calculated based on *wcrts* r_i , and each r_i is determined by the related *wcets* c_i , see Section 4.4.

The robustness of a mapping M_k ($k = 0, 1, \dots$), for the tasks in a task set S_k , is defined as the probability of all tasks in S_k being schedulable,

$$\mathcal{R}_{M_k} = P(r_{M_k} \leq 0) \quad (4.2)$$

where r_{M_k} is the degree of schedulability from Eq. 4.1.

Let us denote M_i , the mapping of the tasks in a future scenario $S_i \in \mathcal{S}_f$, on top of the mapping M_0 of the baseline functionality S_0 , such that the robustness \mathcal{R}_{M_i} is maximized. Then, the flexibility \mathcal{F}_{M_0} of M_0 is defined as,

$$\mathcal{F}_{M_0} = \frac{\sum_{i=1}^{|\mathcal{S}_f|} w_i \times \mathcal{R}_{M_i}}{\sum_{i=1}^{|\mathcal{S}_f|} w_i} \quad (4.3)$$

where w_i is the weight of scenario S_i , and $|\mathcal{S}_f|$ is the number of future scenarios. To calculate \mathcal{F}_{M_0} , we need first to determine the mapping M_i of each S_i , such that \mathcal{R}_{M_i} is maximized (See Section 4.6.2).

4.4 Schedulability Analysis

We are interested to determine the probability \mathcal{R}_{M_k} of a mapping M_k to be schedulable. This value is used in both metrics, robustness (\mathcal{R}_{M_0}) and flexibility (\mathcal{R}_{M_i} , $i = 1, 2, \dots$). In this paper, we assume that tasks are scheduled using a fixed-priority preemptive scheduling policy, and we use a Response Time Analysis (RTA) [Fid98] to determine the *wcrt* r_i of a task τ_i , according to the recurrence equation:

$$r_i^{n+1} = c_i + \sum_{\forall \tau_j \in hp(\tau_i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil c_j \quad (4.4)$$

where $hp(\tau_i)$ is the set of tasks that have a priority higher than the priority of τ_i .

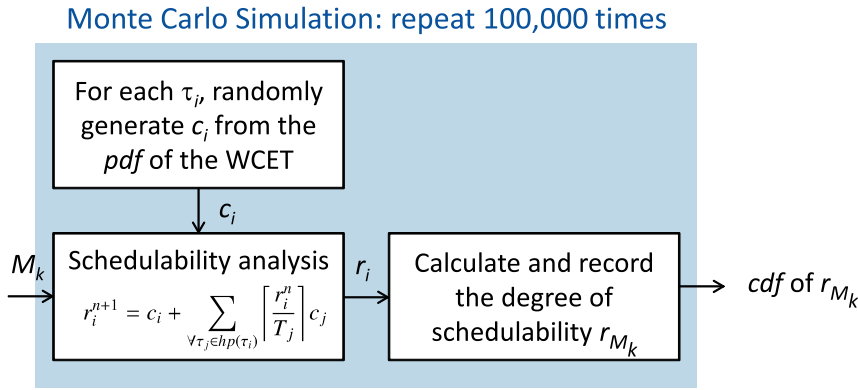


Figure 4.2: Calculating schedulability distribution with MCS

This basic analysis has been extended over the years [Fid98] to take into account blocking times, arbitrary deadlines and release times, jitter, offsets, etc. Our analysis for uncertain *wcets* uses a RTA inside an iterative loop. For simplicity, in this paper, we have decided to consider the case when $D_i \leq T_i$ and ignore the messages. The RTA is orthogonal to our analysis, and can be extended to consider a more general case.

In [KDB⁺05], a stochastic schedulability analysis is used to handle the variability in e_i . Each job $J_{i,j}$ of a task τ_i may have different execution times, depending on the probability distribution function ξ_i of e_i . Thus, for calculating r_i , the updated response time equation (Eq. 4.4) [KDB⁺05] uses stochastic variables of c_i and c_j . In each iteration of the recurrence equation for r_i , c_i and c_j will get different values, based on their *pdfs* of ξ_i and ξ_j , respectively. However, such a solution is not applicable in our case, where each job $J_{i,j}$ of a task τ_i has the same *wcet* c_i in each iteration.

The analysis in [Axe05] uses Monte Carlo Simulation (MSC) to determine the probability distribution of r_i . With MCS, a large number of iterations are run, and the following steps are performed. First, for each task τ_i , a value of c_i is generated based on its *wcet pdf*, σ_i . Second, the generated values of c_i are used to determine the *wcrt* r_i of each task τ_i with Eq. 4.4. Then, the degree of schedulability r_M is calculated using Eq. 4.1. Finally, the r_M values are collected over all iterations, and thus the degree of schedulability *cdf* is obtained. This is illustrated in Fig. 4.2.

MCS requires a large number of iterations (e.g., 100,000) to get an accurate result, which is time-consuming, and thus we cannot use MCS during design space exploration. In this paper, instead of MCS, we propose using the Kernel Smoothing Density Estimate (KSDE) technique [BA97] to quickly approximate the degree of schedulability *cdf*.

Similar to MCS, we start by performing a number of iterations to get the degree of schedulability values. However, we need fewer samples for KSDE (e.g., 1,000 instead of 100,000 in MCS), since a smoothing technique is applied to estimate the *pdf* based on the available samples. Given m random samples X_1, \dots, X_m whose underlying density f is to be estimated, KSDE uses a kernel density estimator,

$$\hat{f}(x, h) = \frac{1}{mh} \sum_{i=1}^m K\left(\frac{x - X_i}{h}\right) \quad (4.5)$$

where $K\left(\frac{x - X_i}{h}\right)$ is the kernel and $h (> 0)$ is the bandwidth. The bandwidth h is a smoothing parameter, which controls how wide the probability mass is spread around a sample.

We evaluated several kernels and bandwidths, and compared the results with those obtained by MCS. We decided to use a *normal* kernel,

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \quad (4.6)$$

and a *normal* optimal smoothing h ,

$$h = \frac{\Phi(X_i - \Phi(X_i))}{0.6745} \cdot \left(\frac{4}{3m}\right)^{\frac{1}{5}} \quad (4.7)$$

where $\frac{\Phi(X_i - \Phi(X_i))}{0.6745}$ is a robust estimate for standard deviation of the distribution, and $\Phi(X_i)$ denotes the median of X_i .

Considering the two mappings, M and M' , from Fig. 4.3, both MCS and KSDE resulted in $\mathcal{R}_M = 93\%$ and $\mathcal{R}_{M'} = 67\%$, i.e., the probability of the tasks in S_0 to be schedulable is 93% (using M) and is 67% (using M'). The difference is that MCS took 25 seconds (using 100,000 samples), whereas KSDE finishes in 0.5 second (using 1,000 samples).

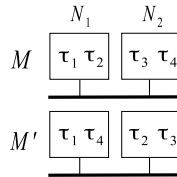


Figure 4.3: Two mapping alternatives

We have used both MCS and KSDE to determine the robustness of 20 task sets mapped on varying number of PEs. The maximum difference between the two techniques is 3%. Thus, we use KSDE as the basis for calculating the two objective functions during the optimization. Note that the analysis presented in this section is only used to guide the search, not to provide schedulability guarantees. We assume that a RTA will be used during the later design and development stages (when maybe more accurate information about *wcets* and the functionality is available) to check the schedulability of an implementation.

4.5 Motivational Example

In the following, we show the importance of modeling and taking into account the uncertainties in the early design phases. For comparison purposes, let us introduce a “straightforward mapping” (SFM) approach, which does not take into account the uncertainties. Thus, with SFM, we consider that each *wcet* is characterized by a single value (the expected value of the *wcet pdf* σ_i , denoted by $E(\sigma_i)$), and the future scenarios are ignored. SFM determines that mapping which minimizes the $E(r_M)$, calculated using the expected *wcet* values in Eq. 4.1.

Let us assume that SFM has to decide between two mappings M and M' from Fig. 4.3, during design space exploration. Using the expected *wcet* values, $r_M = -260$, while, $r_{M'} = -317$, which means that M' would be preferred. However, we reach a different conclusion if we take into account the uncertainties in *wcets* and compare the two mappings in terms of robustness. Fig. 4.4 presents the degree of schedulability *cpfs* for the two mappings. The probability of the mapping being schedulable $P(r_M \leq 0)$ is determined by the intersection of the *cpf* with the vertical line at point “0”. As we can see, M has a better chance of being schedulable (93%) than M' (67%), so actually choosing M instead of M' is more “robust”, i.e., it has a higher chance of being schedulable.

Let us consider the baseline functionality from Table 4.1, and the future scenarios from Table 4.2. We are interested to determine a mapping which maximizes robustness and flexibility. The Pareto-optimal solutions found after an exhaustive search, are depicted by (blue) ‘×’ in Fig 4.5. The rightmost ‘×’ is the most robust mapping, with 95.4% robustness and 88.4% flexibility. The leftmost ‘×’ is the most flexible mapping, with a robustness of 92.7% and a flexibility of 90.1%.

We have also plotted in Fig. 4.5 the optimal mapping obtained by SFM, using a ‘+’ symbol. The robustness and flexibility of this mapping have been calculated using Eq. 4.2 and Eq. 4.3, respectively, taking into account the uncertainty model from

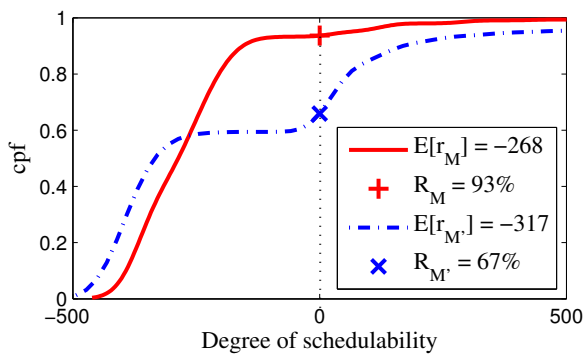


Figure 4.4: Results of M and M'

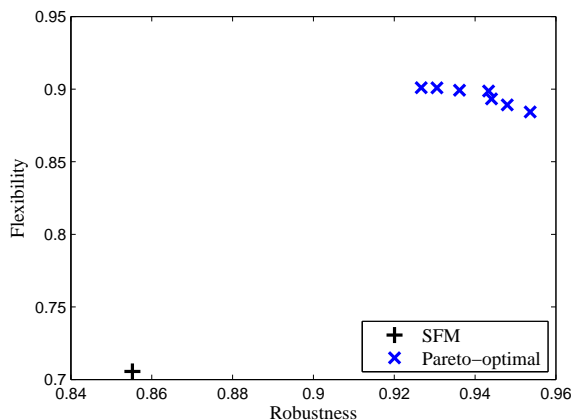


Figure 4.5: SFM and Pareto-optimal

Fig. 4.1 and Fig. 4.2. As we can see, SFM produces a poor quality solution, with only 85.5% robustness and 70.6% flexibility.

4.6 Mapping Optimization

We propose a Genetic Algorithm (GA)-based approach, called Mapping for Robustness and Flexibility (MRF), to solve the optimization problem presented in Section 4.3. In

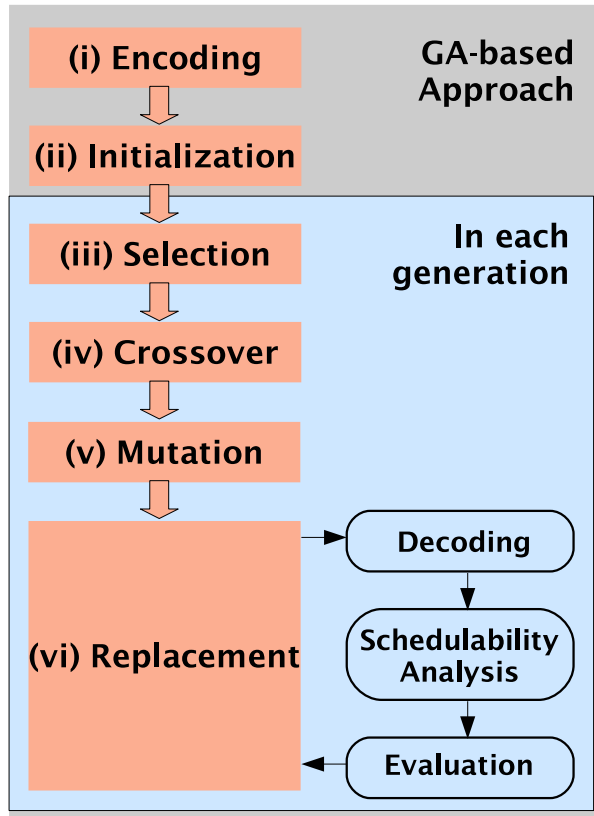


Figure 4.6: Outline of the MRF optimization

Fig.4.6, we show the outline of MRF, and highlight the main steps of GA. Each step in Fig.4.6 will be presented in Section 4.6.1.

4.6.1 NSGA-II for Multiobjectives Optimization

There are several off-the-shelf multiobjective GA implementations, such as NSGA-II [DAPM00] and search frameworks for multiobjective optimization such as PISA [BLTZ03]. In this paper, we focus on determining the importance of modeling the uncertainties in the early design stages, and thus we decided to use the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [DAPM00], due to its good performance and its simple implementation.

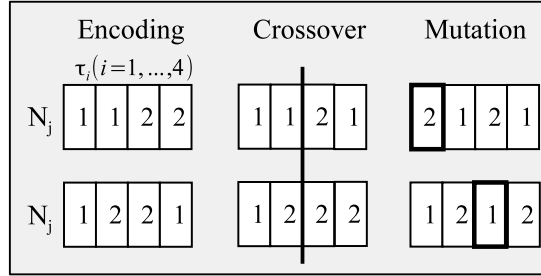


Figure 4.7: Chromosome encoding and design transformations

GA is a metaheuristic optimization approach, which belongs to the class of Evolutionary Algorithms, inspired from the process of natural evolution. The set of candidate solutions is called a “population”, and each solution is (i) *encoded* using a string called a “chromosome”. The population is (ii) *initialized* to n candidate solutions, where n is the *population size*. The population is evolved by (iii) *selecting* a set of solutions and performing (iv) *recombination* and (v) *mutation* to generate offsprings. Finally, the parent population is (vi) *replaced* with an offspring population with better “fitness”. The fitness of a solution is evaluated using our multiobjective function. Steps (iii) to (vi) are repeated until a *termination condition* is reached.

Steps (i) to (vi) are explained in the remainder of this section. There are several choices for their implementation. Through experiments, we decided to choose the following approaches, which can find good solutions in a reasonable time. The parameters were also determined experimentally. One example of parameters is given in Section 4.7.

(i) *Encoding*: We use direct-value encoding, where each chromosome represents a mapping alternative, and each allele (the value of a gene, which is a component of a chromosome) represents a PE. For example, the mapping alternatives M and M' shown in Fig. 4.3 are encoded as chromosomes shown in the first column of Fig. 4.7. The i^{th} position in the chromosome is the index j of the PE N_j which the task τ_i is considered for mapping. Thus, the mapping of τ_1 and τ_2 to N_1 , τ_3 and τ_4 to N_2 is described by the string 1|1|2|2.

(ii) *Initialization*: The initial population is randomly generated and has a population size n .

(iii) *Selection*: We use “tournament selection” to select parents for performing recombination and mutation. In a tournament, four chromosomes are chosen at random, and the fittest one wins. In total, $2(p_c \times n)$ parents are chosen for performing recombination, while $n - (p_c \times n)$ parents are chosen for performing mutation, where p_c is the probability of recombination.

(iv) *Recombination* (also called *crossover*): We employ a standard single point crossover. For each two parents, we compare a randomly generated number with p_c , if this number $\leq p_c$, the two parents are cut at a random point and the sections after the cut point are swapped to generate the offsprings. Otherwise, the offsprings are just copies of their parents. For example, Let us assume that for the two chromosomes in the first column of Fig. 4.7 it is decided to crossover on their second position. In this case, the offsprings are shown in the middle column of Fig. 4.7.

(v) *Mutation* is used to add diversity to a population obtained from recombination. For each position of a parent’s string, we compare a randomly generated number with p_m (probability of mutation) and if this number $\leq p_m$, this position is mutated, i.e., the task is randomly remapped to another PE. In the last column of Fig. 4.7, the gene highlighted by a bold box have been performed mutation.

(vi) *Replacement*: Recombination and mutation generate n offsprings out of the n parents in the current population. Replacement decides which n solutions are kept out of the $2n$ solutions available. The key advantage of NSGA-II lies in how it performs selection and replacement, with the goal of preserving diverse non-dominated solutions, in the hope of finding the Pareto-optimal front. See [DAPM00] for the details on the selection and replacement procedures used in NSGA-II.

The three rounded boxes in the right column of Fig.4.6 describe the steps used for evaluating the fitness of each chromosome (design alternative). *Decoding* the chromosome means extracting the mapping information for each task. Then, we can perform schedulability analysis, used to perform the *evaluation*, when the objective functions are calculated (robustness and flexibility in our case).

Steps (iii) to (vi) are repeated in each generation (highlighted by a blue box in Fig.4.6) until there is no improvement for a given number of consecutive generations, e.g., 10 generations. Our algorithm can also be stopped if a given time limit has been exceeded, e.g., 1 hour. In the end, we obtain a Pareto-front of solutions, which, however, is not guaranteed to contain the Pareto-optimal, since NSGA-II is a search metaheuristic which does not guarantee optimality.

4.6.2 Determining the Mapping of Future Scenarios

When measuring the flexibility \mathcal{F}_{M_0} of a mapping M_0 (performed in the “Evaluation” box of Fig. 4.6), we need to determine the mapping of each future scenario, S_i ($i = 1, 2, \dots$), and calculate its robustness \mathcal{R}_{M_i} using Eq. 4.3. To get an accurate flexibility value for \mathcal{F}_{M_0} , M_i should be as close as possible to the optimal, i.e., it has a maximum robustness value for \mathcal{R}_{M_i} . However, determining such optimal mappings is time-consuming, and the evaluation of flexibility is performed when visiting every

M_0 alternative. Hence, we propose a Greedy algorithm to determine the mapping M_i of each future scenario S_i . Note that we only have to map those tasks from S_i which are not present in the baseline functionality S_0 , i.e., they are new tasks. All the other tasks will keep the some mapping as in M_0 . Thus, the new tasks in S_i are sorted according to their utilization, calculated using the expected *wcets*, i.e., $u_i = E(\sigma_i)/T_i$. Then each task is mapped on the PE with the lowest utilization, and the PE utilizations are updated before moving to the next task.

To determine the quality of the proposed Greedy algorithm, we have also implemented a GA-based mapping. The two algorithms have been evaluated using the synthetic benchmark “*synthetic 1*” (see Section 4.7). The difference is only 4%, but GA is 35-times slower than the greedy algorithm. Therefore, we have decided to use the Greedy algorithm instead of GA for determining the mapping of future scenarios.

4.7 Experimental Results

To evaluate our proposed approach, we used four real-life benchmarks (Table 4.4) from the Embedded System Synthesis Benchmark Suite (E3S), version 0.9 [Dic02], and eight synthetic benchmarks (Table 4.3) generated using Task Graphs For Free (TGFF) [DRW98]. The details of the benchmarks are reported in Columns 1–4 of the tables. For the synthetic benchmarks, *wcets* were generated in the range 30–70 ms. These values are considered as the *wcets* with 50th percentile. *wcets* with 90th percentile are generated to be up to 50% larger than their 50th percentile.

For each benchmark, four future scenarios, S_1 to S_4 , are considered. To create S_1 , we have randomly selected 10% of tasks in S_0 and increased their 50th percentile with 20% (and correspondingly adjusted their 90th percentile). For S_2 , we have randomly

Table 4.3: Synthetic Benchmarks

Test Set	Number of		SFM		MRF				
	PEs	Tasks		\mathcal{R}_{M_0}	\mathcal{F}_{M_0}	Most robust		Most flexible	
		S_0	S_f			\mathcal{R}_{M_0}	\mathcal{F}_{M_0}	\mathcal{R}_{M_0}	\mathcal{F}_{M_0}
<i>synthetic 1</i>	3	22	30	34%	17%	84%	56%	69%	64%
<i>synthetic 2</i>	3	22	30	75%	30%	96%	52%	87%	82%
<i>synthetic 3</i>	6	42	58	26%	22%	70%	23%	61%	63%
<i>synthetic 4</i>	6	42	58	77%	44%	95%	45%	89%	81%
<i>synthetic 5</i>	8	62	86	38%	13%	70%	23%	60%	33%
<i>synthetic 6</i>	8	62	86	81%	23%	97%	64%	93%	73%
<i>synthetic 7</i>	10	84	116	22%	6%	88%	18%	69%	27%
<i>synthetic 8</i>	10	84	116	74%	9%	88%	11%	81%	28%

Table 4.4: Real-life Benchmarks

Test Set	Number of		SFM		MRF				
	PEs	Tasks		\mathcal{R}_{M_0}	\mathcal{F}_{M_0}	Most robust		Most flexible	
		S_0	S_f			\mathcal{R}_{M_0}	\mathcal{F}_{M_0}	\mathcal{R}_{M_0}	\mathcal{F}_{M_0}
<i>consumer-cords</i>	2	12	16	67%	51%	96%	66%	93%	68%
<i>networking-cords</i>	2	13	17	75%	69%	89%	69%	80%	75%
<i>auto-indust-cords</i>	4	24	32	42%	12%	59%	38%	56%	41%
<i>telecom-cords</i>	4	30	42	46%	44%	97%	57%	91%	73%

introduced new functionality, which is about 10% of the size of S_0 . S_3 is similar to S_2 , but larger, 20% of S_0 . Finally, S_4 is a combination of S_1 and S_2 . The weights of four scenarios, S_1 to S_4 , are 0.8, 0.4, 0.6 and 0.2, respectively.

In the first set of experiments (Table 4.3), we were interested to determine the importance of capturing the uncertainties during the early design phases. We have varied the size of the system from 22 tasks (S_0) and 3 PEs to 84 tasks (S_0) and 10 PEs and applied two optimization approaches: MRF, presented in the previous section, which takes into account both uncertainties, and SFM, introduced in Section 4.5, which uses the expected values of *wcets* and ignores the future scenarios.

The robustness and flexibility of SFM are reported in columns 5 and 6 of Table 4.3. MRF produces a Pareto-front of solutions. We show the resulting Pareto-fronts of those synthetic benchmarks in Fig.4.8 – Fig.4.15, using (blue) “x” symbols, and report the extremes in the Pareto-front of all synthetic benchmarks, the most robust solution (columns 7 and 8) and the most flexible solution (columns 9 and 10) in Table 4.3.

Both SFM and MRF are implemented in Matlab 2013a and run on an Intel Core i7 CPU 920 (2.67 GHz) computer. We tuned the NSGA-II parameters such that no improvements were seen for longer run times, thus leading to near optimal solutions. The terminating condition for NSGA-II used in the experiments is a time limit. We have used time limits between 20 minutes to 3 hours, depending on the size of the benchmark. Taking “*synthetic 7*” and “*synthetic 8*” as examples, we set $n = 100$, $p_c = 0.5$, $p_m = 0.25$ and the search terminates in 3 hours.

As we can see from Table 4.3, SFM is not able to find robust and flexible solutions, whereas our MRF approach is able to find good quality solutions, which both robustness and flexibility are significantly increased compared to SFM.

In the second set of experiments, we evaluated the MRF approach using four real-life benchmarks from E3S. The experimental setup details and the results obtained are presented in Table 4.4. We show the resulting Pareto-fronts of those real-life benchmarks in Fig.4.16–Fig.4.19, using (blue) “x” symbols, and report only the extremes in the

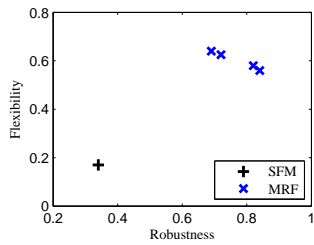


Figure 4.8: synthetic 1

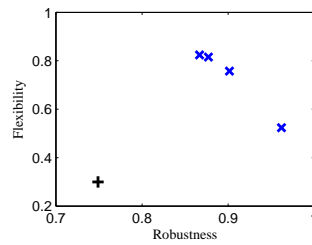


Figure 4.9: synthetic 2

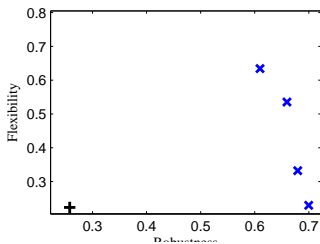


Figure 4.10: synthetic 3

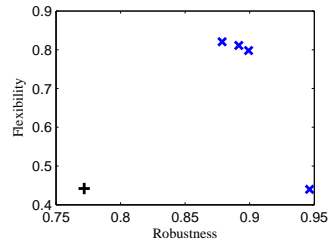


Figure 4.11: synthetic 4

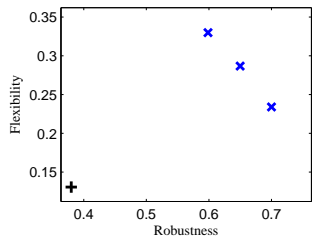


Figure 4.12: synthetic 5

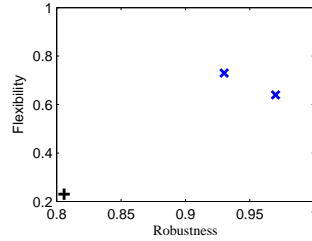


Figure 4.13: synthetic 6

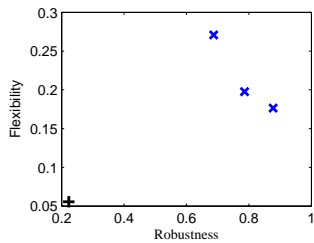


Figure 4.14: synthetic 7

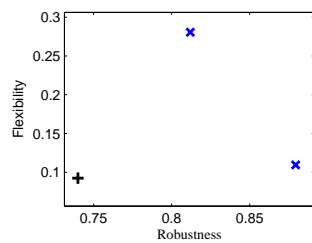


Figure 4.15: synthetic 8

Pareto-front of all real-life benchmarks in columns 7–10 of Table 4.4. The evaluation confirms the results obtained from the synthetic benchmarks.

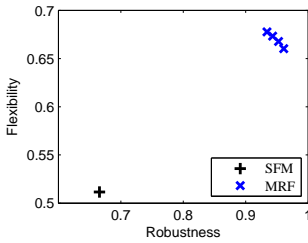


Figure 4.16: *consumer-cords*

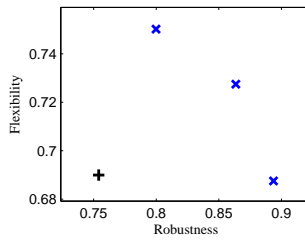


Figure 4.17: *networking-cords*

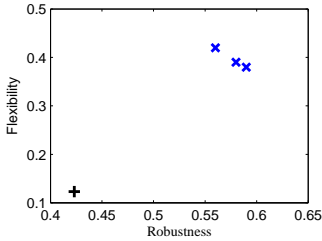


Figure 4.18: *auto-indust-cords*

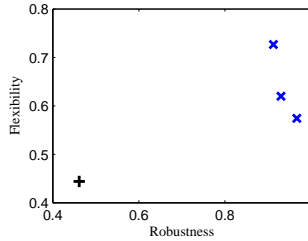


Figure 4.19: *telecom-cords*

4.8 Architecture Selection under Uncertainties

So far, we have assumed that the hardware architecture is given, and we have focused on the mapping optimization. In this section we show how the mapping problem can be extended to consider the architecture selection under uncertainties. In our case, architecture selection means deciding the hardware components, their characteristic and their interconnections.

Embedded system products have to be regularly upgraded, in order to be commercially successful in the currently highly competitive market, especially in consumer electronics. The most common upgrades are fixing bugs, introducing new functionality, and improving performance. Both applications and architecture may have been upgraded such that the new system is able to achieve better performance, compared to the old version. However, since a huge time and effort (e.g., many years) have been invested on developing and maintaining the previous system, reusing the legacy applications and architecture as much as possible is a wise solution to save the costs. Moreover, the experience we gained from the previous system can be used to reduce the design risk.

In the early design stages of building a new system version, the choice of reusing legacy architecture components, using or upgrading to new components has a significant impact on the robustness, flexibility and architecture cost. In the case we choose to use new hardware components, with improved metrics such as better performance, or lower

power dissipation, we may need to redevelop and validate the platform which results in high cost for the new architecture solution and high uncertainties in evaluating the *wcet* of tasks. In case we migrate the legacy hardware components from the previous products, the cost of such an architecture solution should be much less than that in the former case, and the task *wcet* are more certain as well, but we may not benefit from the improved metrics of the new architecture.

As mentioned in Section 4.2.2, uncertainties in evaluating *wcet* of a task may result from the fact that the complete parameters of the architecture are not yet decided in the early design stages. We have modeled the uncertainties in *wcets* and functionality requirements, and we have used the KSDE method as the basis for determining the schedulability probability of a mapping alternative. In this section, we extend our work, to consider the architecture selection and model the uncertainty in architecture cost during the early design phases.

4.8.1 Modeling Cost Uncertainty in Architecture Selection

We consider distributed heterogeneous platforms where a set of PEs are interconnected by shared communication channels. The platform may contain PEs such as General-Purpose Processor (GPP), Digital Signal Processor (DSP) or Application Specific Instruction Set (ASIP). Some of these might be legacy components. We assume that the engineer provides a library of hardware components, denoted by \mathcal{L}_H , which includes a set of PEs, and a set of buses, with different performance and cost.

The cost of the components in the library \mathcal{L}_H is also affected by uncertainties. If a new processor has not yet been developed, its cost is not yet known. If there are supply problems with one of the legacy components that have to be used, the cost could also fluctuate. We model the uncertainty in the cost of an architecture component using the percentile method [Axe05], as we have done with the *wcets*. The uncertainty in performance is captured by the *wcet* model. In this context, the *wcet* uncertainty depends also on the choice of the hardware component that will run the task, which is not yet known.

An example hardware library \mathcal{L}_H , with the cost uncertainty model, is given in Table 4.5. We capture the cost of a hardware component by two values, the 50th and 90th percentiles. This means that in 50% of the possible future implementations, the selected hardware component will have a cost smaller or equal to the 50th percentile value, while in the majority of the cases, i.e., 90%, the cost will not exceed the 90th percentile value.

Table 4.5: Uncertainties in unit cost of hardware components

PEs	50 th [€]	90 th [€]	Buses	50 th [€]	90 th [€]
N_1	30	45	B_1	4	6
N_2	20	30	B_2	3	4.5
N_3	35	105	B_3	6	18
N_4	25	75	B_4	5	15

We calculate the unit cost of an architecture solution, $w_{\mathcal{N}}$, by accumulating the unit cost of each hardware component in the architecture. Note that $w_{\mathcal{N}}$ is a distribution of values due to the variability in the unit cost of each component.

4.8.2 Architecture Selection Problem

The architecture selection and mapping problem can be formulated as follows. As an input to our problem we have (i) a library \mathcal{L}_H of hardware components (PEs and buses) with varying cost and performance, (ii) the baseline functionality \mathcal{S}_0 , and the set of future scenarios \mathcal{S}_f and (iii) a cost budget w_b . For each hardware component, we model its cost using two percentile values, and for each task we know the two *wcet* percentile values for each hardware component where it is considered for mapping.

We are interested to determine an implementation I consisting of the architecture \mathcal{N} (the set of hardware components and their interconnections) the mapping M_0 of the baseline functionality \mathcal{S}_0 on the architecture \mathcal{N} , such that the robustness and flexibility of I are maximized, and the architecture has a high chance to have its unit cost within the budget w_b . The robustness and flexibility are defined in Section 4.3.1: robustness means that the tasks in \mathcal{S}_0 have a high chance of being schedulable, and flexibility means that I has a high chance to successfully accommodate the future scenarios \mathcal{S}_f .

The two objective, i.e., robustness and flexibility are formally defined as in Eq. 4.2 and Eq. 4.3, respectively. The third objective $C_{\mathcal{N}}$ is the probability that the cost $w_{\mathcal{N}}$ of the architecture \mathcal{N} meets the imposed budget w_b . Thus, we define $C_{\mathcal{N}} = P(w_{\mathcal{N}} \leq w_b)$ as the third optimization objective.

4.8.3 Architecture Selection: Motivational Example

Let us illustrate the problem presented in the previous section using the example from Table 4.5 and Table 4.6. Table 4.5 presents the library \mathcal{L}_H of hardware components. The cost is captured in € with the 50th and 90th percentiles. We assume that N_1 and N_2

Table 4.6: Uncertainties in *wcets*

Tasks	N_1		N_2		N_3		N_4		$T_i =$
	50^{th}	90^{th}	50^{th}	90^{th}	50^{th}	90^{th}	50^{th}	90^{th}	D_i
τ_1	10	20	15	30	5	15	7.5	22.5	50
τ_2	25	50	37.5	75	12.5	37.5	18.75	56.25	100
τ_3	40	60	60	90	20	40	30	60	150
τ_4	60	72	90	108	30	45	45	67.5	300

Table 4.7: Uncertainties in functionality requirements

Tasks	N_1		N_2		N_3		N_4		$T_i =$
	50^{th}	90^{th}	50^{th}	90^{th}	50^{th}	90^{th}	50^{th}	90^{th}	D_i
S_0 : Example in Table 4.6									
$S_1 = S_0 \setminus \tau_1 \cup \tau_5$ ($w_1 = 0.8$)									
τ_5	15	45	22.5	67.5	7.5	22.5	11.25	33.75	50
$S_2 = S_0 \cup \tau_6$ ($w_2 = 0.4$)									
τ_6	25	50	37.5	75	12.5	37.5	18.75	56.25	100
$S_3 = S_0 \cup \tau_7 \cup \tau_8$ ($w_3 = 0.6$)									
τ_7	30	60	45	90	15	45	22.5	67.5	300
τ_8	50	75	75	102.5	25	50	37.5	75	300
$S_4 = S_0 \setminus \tau_1 \cup \tau_5 \cup \tau_6$ ($w_4 = 0.2$)									
τ_5	15	45	22.5	67.5	7.5	22.5	11.25	33.75	50
τ_6	25	50	37.5	75	12.5	37.5	18.75	56.25	100

are legacy PEs, while N_3 and N_4 are new PEs for which we do not have their complete information yet. We assume the cost budget of the architecture solution, w_b , to be 100 €, i.e., $w_b = 100$ €.

Table 4.6 presents baseline functionality S_0 in which four tasks, τ_1 to τ_4 , can be mapped on four PEs, from N_1 to N_4 . For each task τ_i and its mapping on each PE N_j , two *wcet* values are given, i.e., the 50^{th} and 90^{th} percentile, respectively. Note that the more information is available or more experience we have about a task and the PEs where it can potentially be mapped, the smaller the difference between the two percentiles. The future scenarios S_f are captured by Table 4.7.

For comparison purposes, we introduce a ‘‘Straightforward Implementation’’ (SFI) approach, which does not take into account the uncertainties due to the architecture selection (*wcet* and cost) and functionality requirements change (future scenarios). Thus, with SFI, we consider that the *wcet* of each task and the unit cost of each component (PE and bus) are characterized by a single value (the *expected value* of their probability density function σ_i , denoted by $E(\sigma_i)$), and the future scenarios are ignored.

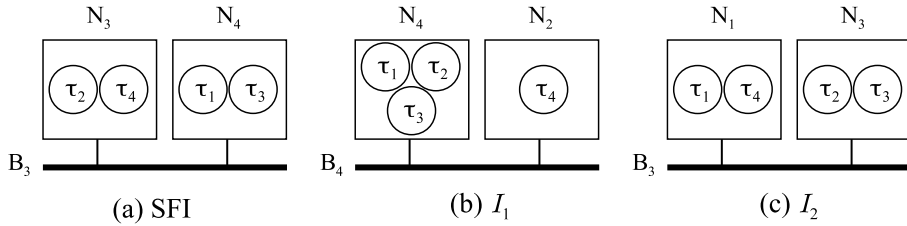


Figure 4.20: Architecture selection and mapping optimization

SFI determines that implementation which maximizes the schedulability of the tasks in \mathcal{S}_0 , i.e., minimizes $E(r_I)$, where r_I is defined as in Eq. 4.1, and meets the cost requirements, i.e., the unit cost of the selected architecture, $w_{\mathcal{N}}$, is within the budget w_b . The implementation I found by SFI is presented in Fig. 4.20(a), and consists of two PEs, N_3 and N_4 interconnected by a bus B_3 . The mapping of \mathcal{S}_0 is also shown in the figure: τ_2 and τ_4 are on N_3 and τ_1 and τ_3 are on N_4 . The values of the objective functions for this solution are $E(r_I) = -486$, and $w_{\mathcal{N}} = 80.77 \text{ €}$ which is below w_b . Note that *expected values* are used during the calculations, and we do not have the $\mathcal{C}_{\mathcal{N}}$ yet for SFI.

We evaluate the solution produced by SFI using our proposed uncertainty models (See Section 4.2.2, Section 4.2.3, and Section 4.8.1) that consider the uncertainties in *wcet*, functionality requirements, and cost. We have plotted the SFI in Fig.4.21, using a red ‘+’ symbol. Fig.4.21 shows the three objective functions of robustness, flexibility and cost on a three-dimensional space. We can see the robustness of SFI is 0.5, while the flexibility is 0.86 and the cost is 0.64, i.e., SFI has only a 50% chance to be schedulable, a 86% chance to schedule the future scenarios, and a 64% chance to meet the architecture budget w_b .

Let us consider the baseline functionality from Table 4.6, the future scenarios from Table 4.7, and the architecture selection from Table 4.5. We are interested to determine an implementation I , which maximizes the robustness, flexibility, and the chance of meeting the architecture budget w_b . Let us call this approach “Architecture selection and Mapping for Robustness, Flexibility and Cost” (AMRFC). In contrast with SFI, AMRFC takes into account the uncertainty models when deriving the architecture and mapping of the implementation I . The Pareto-optimal solutions found after an exhaustive search using AMRFC, are depicted by blue ‘*’ symbols in Fig.4.21.

We can see from Fig.4.21 that, by using the uncertainty models to take decisions on the architecture and mapping, it is possible to have much better quality solutions which have been simultaneously optimized for robustness, flexibility and cost. One of the Pareto-optimal solutions, marked as I_1 in Fig.4.21, has a 100% chance to be schedulable, a 84% chance to accommodate the future scenarios, and a 77% chance to meet

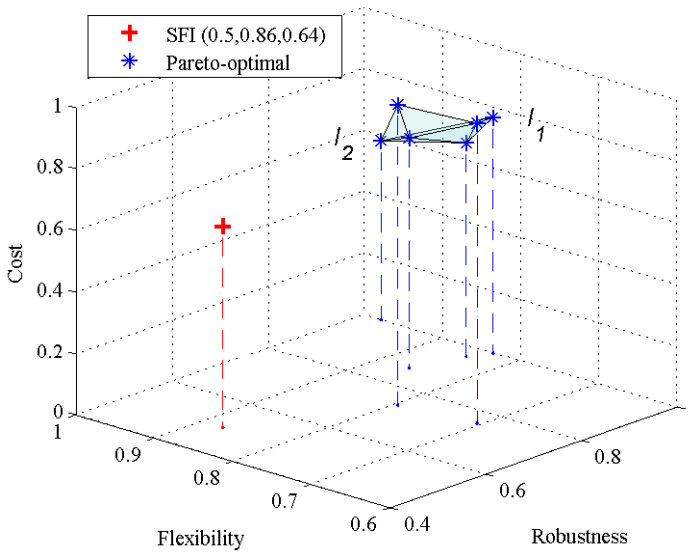


Figure 4.21: SFI and Pareto-optimal

the architecture budget w_b . Compared to SFI, I_1 improves the robustness with 50% percent, and increases with 13% the chance to meet the cost requirement, losing only 2% percentage in flexibility. The detailed implementation I_1 is depicted in Fig.4.21(b).

Another Pareto-optimal solution, marked as I_2 in Fig.4.21, has the best robustness (100%) and flexibility (98%), at a reduction of only 6% chances to meet the cost requirement, compared to SFI. The detailed implementation I_2 is depicted in Fig.4.21(c).

4.8.4 GA-based Approach for Architecture Selection

We have extended the Genetic Algorithm (GA)-based approach, MRF, from Section 4.6.1 to solve the optimization problem formulated in Section 4.8.2 to consider also the architecture selection. We call this extended GA-based approach “Architecture selection and Mapping for Robustness, Flexibility and Cost” (AMRFC). AMRFC has the same structure as MRF (see Fig. 4.6), but we have modified the way of encoding and decoding the chromosomes to include the architecture selection.

In Section 4.6, the architecture was known, and the chromosome has encoded the mapping, such that each gene has represented an index to the PEs in the architecture. In this

section, the chromosome has to include also the architecture selection. Note that we do not know the number of PEs in the architecture: we have to decide both the number of PEs and their types. However, we consider we have a single shared bus, connecting all PEs. Thus, we consider that a chromosome has $\upsilon + \omega + 1$ of genes, where the first υ genes decide the PEs selection, the followed by ω genes used for the task mapping, and the last gene is used for bus selection.

To simplify the implementation of our GA, we assume that the engineer gives an upper bound υ on the number of PEs that can be used in the architecture. Thus, the design space exploration is semi-automatic: the engineer will have to call our proposed AFRFC approach with different values for υ , until he or she is satisfied with the solution. The genes υ used for the architecture selection encode the index to the PEs in the component library \mathcal{L}_H , and the last gene is an index to the buses in \mathcal{L}_H .

We show two chromosomes used for our motivational example in the “Encoding” box of Fig. 4.22. In these two chromosomes, the first four genes indicate that, at the maximum, four PEs can be considered in the architecture. In each of these genes, the index of a selected PE is recorded. The following genes, depicted in gray, represent the mapping of tasks in the baseline functionality \mathcal{S}_0 , and the genes encode an index to the first four genes encoding the PEs. For example, in the first chromosome of Fig. 4.22, task τ_1 in the first gray gene is mapped on the PE encoded in the 1st gene of the chromosome, which points to the PE N_4 in the library \mathcal{L}_H from Table 4.5.

We perform the crossover and mutation similar to Section 4.6.1. These are illustrated in Fig. 4.22. However, the decoding is different compared to Section 4.6.1. Taking the first chromosome in the “Mutation” result as an example, τ_1 is considered to map on N_4 , τ_2 is on N_4 , τ_3 is on N_4 , and τ_4 is on N_2 . The 4th gene in the architecture genes is encoding N_3 , but none of mapping genes is assigned a ‘4’, so the 4th gene of the architecture genes is not actually used. In the current chromosome example, a bus B_4 is selected to interconnect the two PEs N_4 and N_2 .

We mark a design alternative as an infeasible solution in the case its bus utilization over 100%, and a infeasible solution is discarded during the design space exploration and will not be presented in the final Pareto-front of solutions. The utilization is calculated from the bus speed, the size of the messages exchanged over the bus and period of the sender tasks.

We have the decoded strings, shown in the “Decoding” box of Fig. 4.22, where the first i^{th} ($i \in \omega$, $\tau_i \in \mathcal{S}_0$) places in the string are the index j of the PE N_j that task τ_i will be mapped on. The last place is the index i of the selected bus B_i in the architecture. The implementation of these two decoded strings, I_1 and I_2 , are demonstrated in Fig. 4.20(b) and (c), respectively.

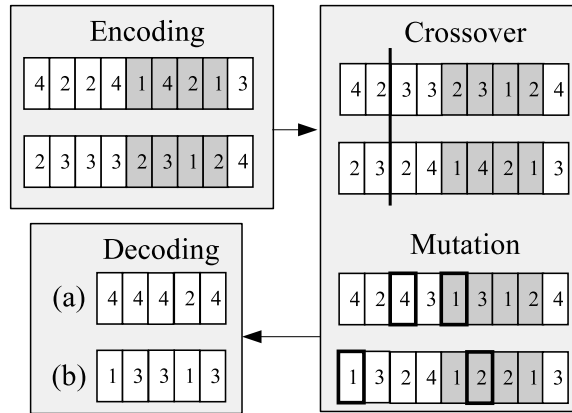


Figure 4.22: Chromosome encoding and decoding

We perform the same strategy for the mapping of future scenarios as in Section 4.6.2. The proposed GA approach has been run for 10 minutes to obtain the Pareto-front solutions, which are exactly the same as the Pareto-optimal solutions determined by performing an exhaustive search that takes 2 hours, for the motivational example presented in Section 4.8.3.

4.9 Conclusion

In this paper, we have addressed the architecture selection and the mapping of hard real-time applications on distributed heterogeneous architectures, during the early design phases. We have considered a fixed-priority preemptive scheduling policy, where the system schedulability is determined using a response time analysis. We have modeled the uncertainties in *wcets*, functionality requirements, and hardware component costs. We have used the Kernel Smoothing Density Estimate as the basis for determining the schedulability probability of a mapping alternative, and the chance of an architecture alternative meeting the cost budget. We have proposed a GA-based optimization for architecture selection and task mapping, targeting robustness, flexibility and cost, which takes into account the uncertainties in *wcets*, functionality requirements and hardware component costs, respectively. The results obtained on the synthetic and real-life benchmarks show the importance of modeling the uncertainties during the early design phases, and taking them into account during design space exploration.

List of Abbreviations

Table A.1: List of Abbreviations

Abbreviations	Description
ABS	Anti-lock Braking Systems
AMRFC	Architecture selection and Mapping for Robustness, Flexibility and Cost (algorithm)
ASIL	Automotive Safety Integrity Level
BCET	Best-Case Execution Time
CMO	Criticality-Aware Mapping Optimization
CDMO	Criticality-Aware Functional Decomposition and Mapping Optimization
DAG	Directed Acyclic Graphs
DAL	Design Assurance Level
DSE	Design Space Exploration
DVFS	Dynamic Voltage and Frequency Scaling
DVS	Dynamic Voltage Scaling
E3S	Embedded System Synthesis Benchmark Suite
FPS	Fixed-Priority Preemptive Scheduling
GA	Genetic Algorithm
GFSR	Global System Failure Rate
IMA	Integrated Modular Avionics

KSDE	Kernel Smoothing Density Estimate
MoC	Models of Computation
MRF	Mapping for Robustness and Flexibility (algorithm)
ms	milliseconds
MSC	Monte Carlo Simulation
MVFS	Mapping, Voltage and Frequency Scaling (algorithm)
NRE	Non-Recurring Engineering (cost)
NSGA-II	Non-dominated Sorting Genetic Algorithm-II
OVFS	Online Voltage and Frequency Scaling (algorithm)
PE	Processing Element
RTA	Response Time Analysis
SFI	Straightforward Implementation
SFM	Straightforward Mapping
SFS	Straightforward Solution
SIL	Safety-Integrity Level
TGFF	Task Graphs For Free
TiSS	Trusted Interface Subsystem
TRM	Trusted Resource Manager
TS	Tabu Search
VaM	Validation Middleware
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time

APPENDIX B

List of Notations

Table B.1: List of Notations used in the thesis

Abbreviations	Description
Γ	a set of applications
\mathcal{N}	a set of PEs in the architecture
$M : \Gamma \rightarrow \mathcal{N}$	the mapping of task to PE
A_i	an application
τ_i	a task
m_i	a message
s_{m_i}	the size of m_i
N_j	a PE
T_i	period
p_c	the probability of crossover
p_m	the probability of mutation
m	population size for GA
$C_i^{N_j}$	WCET of τ_i mapped on N_j
r_i	WCRT of τ_i
$hp(\tau_i)$	the set of tasks that have a priority higher than the priority of τ_i
\mathcal{S}	an implementation (a solution)
$r_{\mathcal{S}}$	the degree of schedulability of \mathcal{S}

Table B.2: List of Notations used in Paper A

Abbreviations	Description
d_i	deadline of functional blocks or tasks
$F_i \in \mathcal{F}$	a functional block
$e_{ij} \in \mathcal{E}$	data dependency between F_i and F_j
$\mathcal{G}(\mathcal{F}, \mathcal{E})$	a graph models an application
\mathcal{D}_i	a set of decomposition options for F_i
$D(F_i) : \mathcal{V}_i \rightarrow \mathcal{D}_i$	the decomposition function of F_i
$G_i(\mathcal{V}_i, \mathcal{E}_i)$	a task graph models a decomposed function F_i
\mathcal{L}_A	a library of function-to-task decompositions
\mathcal{L}_H	a library of architecture implementations for PEs
$H(N_j) \rightarrow N_j^k$	N_j with SIL k is used in the architecture
k	safety criticality level from 0 to 4
R	safety requirements in terms of SIL 0 to SIL 4
P_{ji}	the i th partition on the PE N_j
W_{τ_i}	the cost to develop and certify τ_i to its required SIL
w_j^k	the unit cost of N_j with a reliability corresponding to a SIL k
w_S	total cost of S
C_o	the timing overhead required by the VaM

Table B.3: List of Notations used in Paper B

Abbreviations	Description
d	PE-specific constant
D_i	deadline of τ_i
$f_j^{N_i}$	the operating clock frequency of N_i running in mode j
$v_j^{N_i}$	the supply voltage of N_i running in mode j
$p_j^{N_i}$	the power spent measured of N_i running in mode j
$\Lambda_j^{N_i} = (f_j^{N_i}, v_j^{N_i}, p_j^{N_i})$	the operating mode of N_i running in mode j
F	the normalized frequency
V	the normalized voltage
ℓ	the operating mode
k_i	the redundancy level
$\mathcal{F}: \mathcal{F}(\tau_i) = k_i$	the function to specify the redundancy level of τ_i
J_{ij}	the j th job of τ_i
$\mathcal{L}: \Gamma \rightarrow \left\{ \Lambda_\ell^{M(\tau_i)} \right\}$	the function to assign operating mode for τ_i
R_i	reliability of τ_i
λ	the permanent fault rate of the PE running τ_i
λ_j^ℓ	the transient fault rate of N_j running in ℓ
λ_j^0	the minimum fault rate
λ_j^{max}	the maximum fault rate
μ_j^ℓ	a shape parameter in Eq.3.2
α and β	shape parameters in Eq.3.7
T_{cycle}	the period that the application is repeated periodically
T_S	the mission duration
$c_{ij}^{N_j}$	the execution time of J_{ij} in the fastest operating mode
R_S	the reliability of the system for a period of time T_{cycle}
E_S	energy consumption of finishing all jobs during T_{cycle}
R_g	the reliability goal
S_0	the reference implementation
E_0	the energy consumption of S_0
R_0	the reliability of S_0
W_R and W_S	penalty weights in cost function Eq.3.9
τ_i'	the replica of τ_i
Γ'	the complete task set includes original task and the replicas
\mathcal{C}	the candidate set
ϕ_S	the GSFR of a system
U_S	the sum of the execution times for all jobs executed so far
θ	the degeneration of the system reliability
E_Δ	the saved energy percentage

Table B.4: List of Notations used in Paper C

Abbreviations	Description
D_i	dealine of τ_i
μ and β	Gumbel distribution parameters
S_0	baseline functionality
M_0	the mapping solution for the baseline functionality S_0
S_f	a set of future scenarios
w_i	the probability of a future scenario becoming a reality
\mathcal{R}_{M_k}	the robustness of a mapping M_k
\mathcal{F}_{M_0}	the flexibility of M_0
e_i	execution time of τ_i
ξ_i	probability distribution function of e_i
c_i	stochastic variables of C_i
σ_i	probability distribution function of c_i
u_i	utilization of S_i
\mathcal{L}_H	a library of hardware components
$w_{\mathcal{N}}$	the unit cost of an architecture solution
w_b	cost budget of an architecture solution
$\mathcal{C}_{\mathcal{N}}$	the probability that $w_{\mathcal{N}}$ of \mathcal{N} meets the imposed budget w_b
€	Euro

Bibliography

- [AB98] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of Real-Time Systems Symposium*, pages 4–13, 1998.
- [AFOTH06] J. Alves-Foss, P. W. Oman, C. Taylor, and W. S. Harrison. The mils architecture for high-assurance embedded systems. *International journal of embedded systems*, 2(3):239–247, 2006.
- [AGK11] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability, power consumption, and execution time. *Computer Safety, Reliability, and Security*, pages 437–451, 2011.
- [AGK12] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. *International Journal on Software Tools for Technology Transfer*, pages 1–17, 2012.
- [ALR⁺01] A. Avizienis, J-C Laprie, B. Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.
- [AMMMA01] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium, 2001. Proceedings of 22nd IEEE*, pages 95–105. IEEE, 2001.
- [APW⁺13] L. S. Azevedo, D. Parker, M. Walker, Y. Papadopoulos, and R. E. Araújo. Automatic decomposition of safety integrity levels: Optimiza-

- tion by tabu search. In *Workshop on Critical Automotive applications: Robustness and Safety*, 2013.
- [Ari97] *ARINC 651-1: Design Guidance for Integrated Modular Avionics*. ARINC (Aeronautical Radio, Inc), 1997.
- [Ari13] *ARINC 653P0: Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653*. ARINC (Aeronautical Radio, Inc), 2013.
- [ARP10] SAE ARP4754A. Guidelines for development of civil aircraft and systems, 2010.
- [AS 11] AS 6802. Time-Triggered Ethernet. SAE International, 2011.
- [Axe05] Jakob Axelsson. A method for evaluating uncertainties in the early development phases of embedded real-time systems. In *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*, pages 72–75. IEEE, 2005.
- [Axe06] Jakob Axelsson. Cost models with explicit uncertainties for electronic architecture trade-off and risk analysis. *Intl. Council on Systems Engineering (INCOSE)*, 2006.
- [AZ09] H. Aydin and D. Zhu. Reliability-aware energy management for periodic real-time tasks. *Computers, IEEE Transactions on*, 58(10):1382–1397, 2009.
- [BA97] A. W. Bowman and A. Azzalini. *Applied smoothing techniques for data analysis: The kernel approach with S-Plus illustrations*. Oxford University Press, USA, 1997.
- [BAC00] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches—A survey. *Annals of Software Engineering*, 10(1):177–205, 2000.
- [BBB⁺09] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. A research agenda for mixed-criticality systems. In *Cyber-Physical Systems Week*, 2009.
- [BBL09] E. Bini, G. Buttazzo, and G. Lipari. Minimizing cpu energy in real-time systems with discrete speed management. *ACM Transactions on Embedded Computing Systems (TECS)*, 8(4):31, 2009.
- [BCC⁺05] R. Bloomfield, J. Cazin, D. Craigen, N. Juristo, E. Kessler, and J. Voas. Validation, verification and certification of embedded systems. 2005.

- [BDP96] A. Burns, R. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *Real-Time Systems, 1996., Proceedings of the Eighth Euromicro Workshop on*, pages 29–33. IEEE, 1996.
- [BDS11] P. Bieber, Ré. Delmas, and C. Seguin. D calculus—theory and tool for development assurance level allocation. In *Computer Safety, Reliability, and Security*, pages 43–56. Springer, 2011.
- [BE06] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 221–230. IEEE, 2006.
- [Bib77] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, DTIC Document, 1977.
- [BL73] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.
- [BLTZ03] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. Pisa – a platform and programming language independent interface for search algorithms. pages 1–1, 2003.
- [BM94] A. A. Bertossi and L. V. Mancini. Scheduling algorithms for fault-tolerance in hard-real-time systems. *Real-Time Systems*, 7(3):229–245, 1994.
- [BMS00] B. W. Boehm, R. Madachy, and B. Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.
- [Boe88] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.
- [Bos91] Robert Bosch. Can specification version 2.0. *Robert Bosch GmbH, Stuttgart*, 1991.
- [BSB⁺01] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.
- [Bue11] D. M. Buede. *The engineering design of systems: models and methods*, volume 55. John Wiley & Sons, 2011.
- [But97] Giorgio Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.

- [But11] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer, 2011.
- [CAS01] CAST. CAST-2: Guidelines for assessing software partitioning/protection schemes. Position Paper, Federal Aviation Administration, 2001.
- [CGL⁺07] P. Cuenot, S. Gerard, H. Lonn, M-O Reiser, D. Servat, C-J Sjostedt, R. T. Kolagari, M. Torngren, M. Weber, et al. Managing complexity of automotive electronics using the east-adl. In *12th IEEE International Conference on Engineering Complex Computer Systems*, pages 353–358, 2007.
- [Cha09] R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.
- [CHK06] J. J. Chen, H. R. Hsu, and T. W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 408–417. IEEE, 2006.
- [CK07] J. J. Chen and C. F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Embedded and Real-Time Computing Systems and Applications, 2007. Conference on*, pages 28–38. IEEE, 2007.
- [CMR92] A. Campbell, P. McDonald, and K. Ray. Single event upset rates in space. *Nuclear Science, IEEE Transactions on*, 39(6):1828–1835, 1992.
- [CMS82] X. Castillo, S. R. McConnel, and D. P. Siewiorek. Derivation and calibration of a transient error reliability model. *Computers, IEEE Transactions on*, 100(7):658–671, 1982.
- [Con03] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *Micro, IEEE*, 23(4):14–19, 2003.
- [CRM⁺06] A. K. Coskun, T. S. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici. Analysis and optimization of mpsoc reliability. *Journal of Low Power Electronics*, 2(1):56–69, 2006.
- [CYK06] J. J. Che, C. Y. Yang, and T. W. Kuo. Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. Conference on*, volume 1. IEEE, 2006.
- [DAPM00] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, pages 849–858. Springer, 2000.

- [Dic02] R. Dick. Embedded system synthesis benchmarks suite, 2002.
- [Dic08] Robert Dick. Embedded system synthesis benchmarks suites (e3s), 2008.
- [DMG97] J. A. Debardelaben, V. K. Madiseti, and A. J. Gadiant. Incorporating cost modeling in embedded-system design. *IEEE Design and Test of Computers*, 14:24–35, July 1997.
- [DRW98] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101. IEEE Computer Society, 1998.
- [EAHS⁺06] A. Ejlali, B. M. Al-Hashimi, M. T. Schmitz, P. Rosinger, and S. G. Miremadi. Combined time and information redundancy for seutolerance in energy-efficient real-time systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(4):323–335, 2006.
- [EJ09] C. Ebert and C. Jones. Embedded software: Facts, figures, and future. *Computer*, 42(4):42–52, 2009.
- [ELLSV97] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Proceedings of the IEEE*, 85(3):366–390, 1997.
- [Ern10] Rolf Ernst. Certification of Trusted MPSoC Platforms. 10th International Forum on Embedded MPSoC and Multicore, 2010.
- [Est07] Jeff A. Estefan. Survey of model-based systems engineering (mbse) methodologies. *IncoSE MBSE Focus Group*, 25, 2007.
- [EY97] R. Ernst and W. Ye. Embedded program timing analysis based on path clustering and architecture classification. In *Computer-Aided Design, 1997. Digest of Technical Papers., 1997 Conference on*, pages 598–604. IEEE, 1997.
- [FÅS04] J. Fredriksson, M. Åkerholm, and K. Sandström. Calculating resource trade-offs when mapping component services to real-time tasks. In *Fourth Conference on Software Engineering Research and Practice in Sweden Linköping, Sweden*, 2004.
- [FDT05] Sé. Faucou, A-M Déplanche, and Y. Trinquet. An adl centric approach for the formal design of real-time systems. In *Architecture Description Languages*, pages 67–82. Springer, 2005.
- [FH04] C. Ferdinand and R. Heckmann. ait: Worst-case execution time prediction by static program analysis. In *Building the Information Society*, pages 377–383. Springer, 2004.

- [FHKS09] J. Friedrich, U. Hammerschall, M. Kuhrmann, and M. Sihling. Das v-modell xt. In *Das V-Modell® XT*, pages 1–32. Springer, 2009.
- [Fid98] C. J. Fidge. Real-time schedulability tests for preemptive multitasking. *Real-Time Systems*, 14(1):61–93, 1998.
- [GAGS09] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Publishing Company, Incorporated, 2009.
- [GK83] D. D. Gajski and R. H. Kuhn. Guest editors’ introduction: New vlsi tools. *Computer*, 16(12):11–14, 1983.
- [GK03] F. Gruian and K. Kuchcinski. Uncertainty-based scheduling: Energy-efficient ordering for tasks with variable execution time. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 465–468, 2003.
- [GK09] A. Girault and H. Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *Dependable and Secure Computing, IEEE Transactions on*, 6(4):241–254, 2009.
- [Glo89] Fred Glover. Tabu search: part i. *ORSA Journal on computing*, 1(3):190–206, 1989.
- [Gru01] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 46–51, August 6–7 2001.
- [GZDNBH10] A. Ghosal, H. Zeng, M. Di Natale, and Y. Ben-Haim. Computing robustness of flexray schedules to uncertainties in design parameters. In *Proc. DATE*, pages 550–555, 2010.
- [Hai11] Yacov Y Haimes. *Risk modeling, assessment, and management*. John Wiley & Sons, 2011.
- [HD93] K. Hoyme and K. Driscoll. SAFEbus. *IEEE Aerospace Electronic Systems Magazine*, 8:34–39, 1993.
- [HS06] T.A. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM 2006: Formal Methods*, pages 1–15. Springer, 2006.
- [HTRE02] C. Haubelt, J. Teich, K. Richter, and R. Ernst. System design for flexibility. In *Proc. DATE*, pages 854–861, 2002.
- [IBM10] IBM. DO-178B compliance: turn an overhead expense into a competitive advantage. White paper, IBM Rational, 2010.

- [IEC98] IEC. 61508 functional safety of electrical/electronic/programmable electronic safety-related systems. *International electrotechnical commission*, 1998.
- [IEC10] IEC 61508. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission, 2010.
- [IPP⁺09] V. Izosimov, I. Polian, P. Pop, P. Eles, and Z. Peng. Analysis and optimization of fault-tolerant embedded systems with hardened processors. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 682–687. IEEE, 2009.
- [ISG03] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 37–46. Society for Industrial and Applied Mathematics, 2003.
- [ISO08] ISO 9001. Quality management systems - Requirements. International Organization for Standardization, 2008.
- [ISO09] ISO/DIS 26262. ISO/DIS 26262 - Road vehicles — Functional safety. International Organization for Standardization / Technical Committee 22 (ISO/TC 22), 2009.
- [ISO11] CD ISO. 26262, road vehicles—functional safety. *International Standard ISO/FDIS, 26262*, 2011.
- [JS07] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, 2007.
- [KB03] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.
- [KDB⁺05] K. Kim, J. L. Diaz, L. L. Bello, J. M. Lopez, C. G. Lee, and S. L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *Computers, IEEE Transactions on*, 54(11):1460–1466, 2005.
- [KK10] I. Koren and C. M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2010.
- [KN00] S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications*. World Scientific Publishing Company, 2000.
- [Kop11a] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2011.

- [Kop11b] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science+ Business Media, 2011.
- [KSSB11] A. Kossiakoff, W. N. Sweet, S. Seymour, and S. M. Biemer. *Systems engineering principles and practice*, volume 83. Wiley. com, 2011.
- [KWS03] S. Kodase, S. Wang, and K.G. Shin. Transforming structural model to runtime model of embedded software with real-time constraints. In *Proceedings of the conference on Design, Automation and Test in Europe: Designers' Forum-Volume 2*, page 20170. IEEE Computer Society, 2003.
- [LGT10] M. Lukasiwycz, M. Glaß, and J. Teich. Robust design of embedded systems. In *Proc. DATE*, pages 1578–1583, 2010.
- [LP05] L. Lavagno and C. Passerone. Design of embedded systems. *Embedded Systems Handbook*, 2005.
- [LS04] J. R. Lorch and A. J. Smith. Pace: A new approach to dynamic voltage scaling. *Computers, IEEE Transactions on*, 53(7):856–869, 2004.
- [LSH10] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8. USENIX Association, 2010.
- [LSOH07] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber. A Comparison of Partitioning Operating Systems for Integrated Systems. *Computer Safety, Reliability, and Security*, pages 342–355, 2007.
- [Mar11] Peter Marwedel. *Embedded system design*. Springer, 2011.
- [MEP06] S. Manolache, P. Eles, and Z. Peng. Buffer space optimisation with communication synthesis and traffic shaping for nocs. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pages 718–723. European Design and Automation Association, 2006.
- [MME04] R. Melhem, D. Mossé, and E. Elnozahy. The interplay of power management and fault recovery in real-time systems. *Computers, IEEE Transactions on*, 53(2):217–231, 2004.
- [PEP02] P. Pop, P. Eles, and Z. Peng. Flexibility driven scheduling and mapping for distributed real-time systems. In *RTCSA*, 2002.
- [PEP04] P Pop, P Eles, and Z Peng. *Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems*. Kluwer Academic Publishers, 2004.

- [PEPP04] P. Pop, P. Eles, Z. Peng, and T. Pop. Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(8):793–811, 2004.
- [PGH98] J. C. Palencia and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 26–37. IEEE, 1998.
- [PIEP09] P. Pop, V. Izosimov, P. Eles, and Z. Peng. Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(3):389–402, 2009.
- [Pik] Pikeos: <http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/>.
- [POK] Pok: <http://pok.safety-critical.net>.
- [PPE⁺08] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems*, 39(1-3):205–235, 2008.
- [PPIE07] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 233–238. ACM, 2007.
- [PS01] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, pages 89–102. ACM, 2001.
- [PTV⁺13] P. Pop, L. Tsiopoulos, S. Voss, O. Slotosch, C. Ficek, U. Nyman, and A. Ruiz. Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: the RECOMP approach. In *Proceedings of the Workshop of Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2013.
- [PWA⁺13] D. Parker, M. Walker, L. Azevedo, Y. Papadopoulos, and R. Araújo. Automatic decomposition and allocation of safety integrity levels using a penalty-based genetic algorithm. In Moonis Ali, Tibor Bosse, Koen V. Hindriks, Mark Hoogendoorn, Catholijn M. Jonker, and Jan Treur, editors, *Recent Trends in Applied Artificial Intelligence*, volume 7906 of *Lecture Notes in Computer Science*, pages 449–459. Springer Berlin Heidelberg, 2013.

- [PWR⁺10a] Y. Papadopoulos, M. Walker, M-O Reiser, M. Weber, D. Chen, M. Törngren, D. Servat, A. Abele, F. Stappert, H. Lonn, et al. Automatic allocation of safety integrity levels. In *Proceedings of the 1st workshop on critical automotive applications: robustness & safety*, pages 7–10. ACM, 2010.
- [PWR⁺10b] Y. Papadopoulos, M. Walker, M.-O. Reiser, M. Weber, D. Chen, M. Törngren, David Servat, A. Abele, F. Stappert, H. Lonn, L. Berntsson, Rolf Johansson, F. Tagliabo, S. Torchiaro, and Anders Sandberg. Automatic allocation of safety integrity levels. In *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness and Safety*, pages 7–10, 2010.
- [QNHM04] G. Quan, L. Niu, X. S. Hu, and B. Mochocki. Fixed priority scheduling for reducing overall energy on variable voltage processors. In *Real-Time Systems Symposium, 2004. Proceedings of 25th IEEE International*, pages 309–318. IEEE, 2004.
- [Res12] Transparency Market Research. Embedded system market - global industry analysis, size, share, growth, trends and forecast, 2012 - 2018. 2012.
- [RHE06] R. Racu, A. Hamann, and R. Ernst. A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10–pp. IEEE, 2006.
- [Roa09] ITRS Roadmap. International technology roadmap for semiconductors, 2009 edn. *Executive Summary. Semiconductor Industry Association*, 2009.
- [Roc09] Rockwell-Collins. Certification cost estimates for future communication radio platforms. Technical report, Rockwell-Collins, 2009.
- [RTC92] RTCA DO-178B. Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics (RTCA), 1992.
- [Rus99] John Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.
- [Rus07] John Rushby. Just-in-time certification. In *12th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, pages 15–24, 2007.
- [SAHE03] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. *System-level design techniques for energy-efficient embedded systems*. Springer, 2003.

- [SAHE04] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. *System-level design techniques for energy-efficient embedded systems*. Springer, 2004.
- [SBK] D. Sojer, C. Buckl, and A. Knoll. Propagation, transformation and refinement of safety requirements.
- [SPM09] P.K. Saraswat, P Pop, and J Madsen. *Task Migration for Fault-Tolerance in Mixed-Criticality Embedded Systems*, volume 6, Number 3. 2009.
- [SR11] A.P. Sage and W.B. Rouse. *Handbook of systems engineering and management*. Wiley. com, 2011.
- [Sto96] N.R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.
- [Tas02] Gregory Tassej. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project, 7007*, 2002.
- [TBDP98] E. Totel, J-P Blanquart, Y. Deswarte, and D. Powell. Supporting multiple levels of criticality. In *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pages 70–79. IEEE, 1998.
- [TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an np-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.
- [TCN00] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000.
- [TSP11a] D. Tamas-Selicean and P. Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 24–33. IEEE, 2011.
- [TSP11b] D. Tamas-Selicean and P. Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 24–33. IEEE, 2011.
- [TSPS12a] D. Tamas-Selicean, P. Pop, and W. Steiner. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proceedings of conference on Hardware/software codesign and system synthesis*, pages 473–482. ACM, 2012.

- [TSPS12b] D. Tămaş-Selicean, P. Pop, and W. Steiner. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *Proceedings of the International Conference on Hardware/software Code-sign and System Synthesis*, pages 473–482, 2012.
- [Vah06] Frank Vahid. *Embedded system design: a unified hardware/software introduction*. John Wiley & Sons, 2006.
- [Was14] Armin Wasicek. The across integrity model. In *IAENG Transactions on Engineering Technologies*, pages 333–348. Springer, 2014.
- [WC12] D. D. Ward and S. E. Crozier. The uses and abuses of asil decomposition in iso 26262. In *System Safety, incorporating the Cyber Security Conference 2012, 7th IET International Conference on*, pages 1–6. IET, 2012.
- [WESK10] A. Wasicek, C. El-Salloum, and H. Kopetz. A system-on-a-chip platform for mixed-criticality applications. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on*, pages 210–216. IEEE, 2010.
- [WMWZ12] T. Wei, P. Mishra, K. Wu, and J. Zhou. Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems. *Journal of Systems and Software*, 2012.
- [WRPG11] J. Wei, L. Rashid, K. Pattabiraman, and S. Gopalakrishnan. Comparing the effects of intermittent and transient hardware faults on programs. In *Dependable Systems and Networks Workshops, Conference on*, pages 53–58. IEEE, 2011.
- [ZAZ13] B. Zhao, H. Aydin, and D. Zhu. Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):23, 2013.
- [ZC03] Y. Zhang and K. Chakrabarty. Energy-aware adaptive checkpointing in embedded real-time systems. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 918–923. IEEE, 2003.
- [ZC04] Y. Zhang and K. Chakrabarty. Dynamic adaptation for fault tolerance and power management in embedded real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(2):336–360, 2004.
- [ZC06] Y. Zhang and K. Chakrabarty. A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(1):111–125, 2006.

-
- [ZCP⁺05] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *Application of Concurrency to System Design*, pages 132–141, 2005.
- [ZMM04] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 35–40. IEEE, 2004.