Novel Scheduling Strategies for Future NoC and MPSoC Architectures

DISSERTATION
zur Erlangung des Grades eines Doktors
der Naturwissenschaften

vorgelegt von
Dipl.-Math. Christian Schöler

betreut von
Prof. Dr. René Krenz-Bååth
Prof. Dr.-Ing. habil. Roman Obermaisser

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2017

Datum der Disputation: 6. Juni 2017

gedruckt auf alterungsbeständigem holz- und säurefreiem Papier

Abstract English

Time-Triggered Network-on-Chip (TTNoC) and Multi-Processor-System on a Chip (MPSoC) are networking concepts aiming at providing both predictable and high-throughput communication for modern multiprocessor systems. Time-triggered networks play an important role in safety-critical systems, where their inherent properties such as temporal predictability, fault tolerance and composability improve safety and reduce certification costs. Time-triggered networks use timeplans, which define the points in time of all message exchanges with respect to a global time base. In multi-cluster time-triggered systems of large embedded systems (e.g. automotive, avionics), conflict-free paths along switches and endsystems are defined for each message. The conflict-free temporal and spatial allocation of communication resources in combination with an intelligent communication network (e.g. local and central guardians) prevents interference between messages from different components upon integration and in the presence of faults.

Therefore, message scheduling in TTNoCs is one of the major challenges, where the points in time for the transmission of a message with conflict-free paths through the switches are determined. As the scheduling problem is $\mathcal{NP}$-complete this work introduces a novel scheduling framework based on the latest advancements of theorem solvers such as Satisfiability Modulo Theories (SMT) techniques which have successfully been applied to problem instances of this complexity class.

In addition, this work also investigates different concepts to partition the problem instances allowing the application of parallel computing to further accelerate the proposed scheduling framework.

MPSoC architectures and their specific architectural properties will require scheduling tools capable of dealing with the increasing complexity of the systems. To meet these challenges, we will outline how the proposed scheduling framework performs after it has been ported to an MPSoC emulating target system. We compare its performance to state-of-the-art schedulers based on CPLEX. Furthermore, we will analyze how the proposed scheduling framework can be deployed to recover from faults by re-scheduling the system under consideration at runtime.

Kurzzusammenfassung Deutsch

Zur Erfüllung steigender Anforderungen hinsichtlich Sicherheit, Komfort und Effizienz an Mobilitätssysteme spielt die Elektronik eine immer größere Rolle. Dem daraus resultierenden Mehrbedarf an Performanz sowie der Reduzierung von Energieverbrauch, Größe und Gewicht können nur Multicoresysteme gerecht werden. Der Einsatz solcher Systeme in sicherheitskritischen Mobilitätsbereichen bringt allerdings noch viele offene Fragen mit sich:

Mit den steigenden Anforderungen an die Verfügbarkeit von Funktionen speziell in Zukunftsthemen wie (teil-)automatisiertem Fahren oder Internet of Things, müssen neue Architekturpattern, sogenannte Multi-Processor-Systems on a Chip (MP-SoCs), entwickelt werden. Diese Patterns sollen eine hochgradige Verfügbarkeit von Funktionen sicherstellen und gleichzeitig kostengünstig umgesetzt werden.

Eine der zentralen Fragestellungen für moderne Mehrkernarchitekturen ist eine effektive Umsetzung der internen Kommunikationsprozesse. Die vorliegende Arbeit präsentiert einen Scheduler, der optimale Zeitpläne berechnen kann und auch auf einem eingebetteten System mit eingeschränkter Rechenkapazität eingesetzt werden kann. Darüber hinaus diskutieren wir Strategien, um diesen Scheduler effizient einzusetzen. Als Grundlage für die Berechnung von optimalen Schedules nutzen wir moderne Tools aus der automatisierten Verifikationstheorie, da das optimale Scheduling Problem aus der Komplexitätsklasse der $\mathcal{NP}-$vollständigen Probleme stammt. Wir erläutern, warum Verifikationstools aus diesem Gebiet geeignet sind, um optimale Schedules für zeitgesteuerten Systemen, sogenannten TTNoCs, zu berechnen.

Der Einsatz dieser Programme bietet zwei zentrale Vorteilen gegenüber der herkömmlichen Berechnung mit Multi-Integer-Linear-Programming (MILP) basierten Schedulern: Zum einen können wir die Laufzeit und den Speicherbedarf reduzieren und auf der anderen Seite unser Scheduling Framework auch direkt auf dem MPSoC verwenden. Neben dem Einsatz auf dem Zielsystem evaluieren wir, wie unser Scheduler zur Laufzeit eingesetzt werden kann und die Fehlertoleranz des betrachteten Zielsystems verbessert. Dabei analysieren wir insbesondere die Performance unseres Schedulers im Vergleich zu einem weit verbreiteten heuristischen Ansatz.

Declaration of Authorship

Publications

Below are listed the peer-reviewed publications that were published during the writing of this thesis between 2014 and 2016. Parts of the presented dissertation are based on these references. The contributions are listed in descending order of publication date:

- Christian Schöler, René Krenz-Bååth, Roman Obermaisser (2015). A Novel Formal Verification Framework for Future MPSoC Architectures. In *Proc. on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN/ETS) Workshop, co-located with DATE 2015, ISBN*, Grenoble, France [1].

- Christian Schöler, Ayman Murshed, René Krenz-Bååth, Roman Obermaisser (2015). Optimal SAT-based Scheduler for Time-Triggered Networks -on-a-Chip. In *Proceedings of 10th IEEE International Symposium on Industrial Embedded Systems*, Siegen, Germany [2].

- Christian Schöler, Ayman Murshed, René Krenz-Bååth, Roman Obermaisser (2016). Computing Optimal Communication Schedules for Time-Triggered Networks Using an SMT Solver. In *Proceedings of 11th IEEE International Symposium on Industrial Embedded Systems*, Krakow, Poland [3].

- Christian Schöler, René Krenz-Bååth, Roman Obermaisser (2016). A Dominator-Based Partitioning for Efficient Scheduling in Time-Triggered NoCs. In *Proceedings of 42nd Euromicro DSD/SEAA 2016*, Limassol, Cyprus [4].

In addition, the topic of this thesis was accepted for poster presentation at the PhD Forum of IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC) in Tallinn, Estonia 2016. Our contribution can be accessed in the proceedings of the conference:

- Christian Schöler, René Krenz-Bååth, Roman Obermaisser (2016). Novel Scheduling Strategies for future NoC and MPSoC Architectures. In *Proceedings of 24th IFIP/IEEE International Conference on Very Large Scale Integration*, Tallinn, Estonia [5].

Acknowledgements

First of all, I owe thanks to my doctor-fathers Professor René Krenz-Bååth and Professor Roman Obermaisser for suggesting this interesting research topic, for their endless patience and for letting me pursue the topic so freely. Furthermore, I want to express my gratitude to the Hochschule Hamm-Lippstadt (HSHL), University of Applied Sciences, for funding my work and giving all organizational support required. Furthermore, I wish to take the opportunity to thank Microsoft Research and Princeton University for their generous financial sponsorship allowing me to attend the Conference for Computer Aided Verification (CAV) in Toronto in July 2016.

I also wish to express my gratitude explicitly to Prof. Klaus Zeppenfeld and Karl-Heinz Sandknop representing the Board of HSHL for funding this research project over 36 months. Also I want to thank all members of HSHL-staff for their continuous and passionate support over the past three years. Without their help I would not have been able to realize this thesis.

In the same way I also want to mention my colleagues from the Fachgruppe Embedded Systems at Siegen University who supported my studies in every possible way. Not only did they kindly grant access to all their facilities but also made me feel at home whenever I visited. Special credit is of course given to Ayman Murshed, my respected co-author of two publications, whose expert knowledge on CPLEX was absolutely valuable for my work. Also I want to thank Hamidreza Ahmadian for giving me a detailed introduction into the models we consider in the scope of this work.

Obviously this project would have been impossible without the constant, caring and reassuring support of my family, especially my wife Sarah, my parents and my parents-in-law. They were always helping out and found the time to look after our son Julian, who is of course a precious gift and has been a valuable source of diversion whenever needed.

Finally, this work was partially supported by the European project DREAMS under the Grant Agreement No. 610640. Thank you.

Contents

1 Introduction

1.1 Motivation

This thesis is located in the field of multicore architectures implemented on a single chip like Multi Processor System-on-Chip (MPSoC) based on Network-on-Chip (NoC). NoC and MPSoC started a new computing era but brought a twofold challenge: On the one hand a new hardware paradigm emerged which required a layout pattern easy to use for software designers. On the other hand efficient NoC and MPSoC architectures require efficient software capable of utilizing the full potential of the high degree of parallelisation. Crucial to both research issues is the provision of a powerful communication infrastructure on the architectures under consideration. This work focuses on NoC and MPSoC architectures deploying a time-triggered communication protocol and introduces a novel scheduling framework providing optimal schedules by applying verification engines such as Boolean satisfiability (SAT) solvers and Satisfiability Modulo Theories (SMT) solvers.

Efficient scheduling on time-triggered NoC and MPSoC architectures is a prominent research subject especially as multicore chips play an important role in safety-critical systems, where their inherent properties such as temporal predictability, fault tolerance and composability [6] improve safety and reduce certification costs [7]. Time-triggered networks on and off chip use timeplans, which define the points in time of all message exchanges with respect to a global time base. In multi-cluster time-triggered systems of large embedded systems (e.g., automotive, avionics), conflict-free paths along switches and endsystems are defined for each message. The conflict-free temporal and spatial allocation of communication resources in combination with an intelligent communication network (e.g., local and central guardians) prevents interference between messages from different components upon integration and in the presence of faults.

However, the computation of the timeplans is time consuming and computationally complex [8]. Feasible schedules need to avoid conflicts of communication resources, meet deadlines, satisfy precedence constraints and ensure implicit synchronization [9]. Different optimal and heuristic scheduling algorithms have been devised in the state-of-the art (e.g. enumerative methods, mathematical programming, simulated annealing, genetic algorithms, tabu search, neural networks). While heuristics often fail to find feasible schedules, the runtime of optimal algorithms becomes prohibitive upon large problem sizes. This is due to the fact that computation of optimal schedules is one of the hardest problems in algorithmic and one of the first problems proven to be $\mathcal{NP}$-complete.

In order to address this challenge we introduce a communication model which

allows the application of the latest advances in SAT and SMT-solving which have undergone tremendous progress in the past two decades. SAT and SMT solvers are the backbone of a wide range of academic and industrial research activities and are widely used to tackle complex mathematical problems. On their basis optimal schedules can be computed in reasonable time for different types of MPSoCs.
There are various fields of applications for time triggered NoC and MPSoC architectures in industry for instance in (safety-critical) embedded systems in the automotive industry [10] as well as in academia [11]. Different types of embedded time-triggered execution platforms can be distinguished in practice [12, 13, 14].

1.2 Problem Statement

The main purpose of this work is the development of a scheduling framework tailored to meet the tremendous complexity of evolving NoC and MPSoC architectures and their specific architectural properties. As modern NoC and MPSoC architectures are highly parallel systems an efficient communication infrastructure is of vital importance. In this thesis we optimize the schedule with respect to the given system requirement of minimizing the end-to-end latency. We can also formulate this by referring to the minimal transmission time for all messages or the minimal makespan. We introduce a model which allows the application of state-of-the art tools from the area of hardware verification like SAT and SMT to find optimal schedules.
Finding optimal schedules is one the most complex problems in computer science and is known to be $\mathcal{NP}$-complete. Therefore, we also analyse the tradeoff between optimal solutions which may be costly to compute in terms of runtime and heuristical methods which may be fast to calculate but result in a larger end-to-end-latency.

1.3 Thesis Contribution

This thesis contributes to the field of design, implementation and verification of Multi-Processor System-on-a-Chip (MPSoC) architectures realizing embedded real-time systems. We develop a concept to deploy a set of SAT and SMT solvers to meet two different challenges: On the one hand we want to guarantee safety after reconfiguration and on the other hand we can find optimal solutions at runtime for scheduling problems within a network, which allows tolerating failing nodes or broken links and guarantees correctness within a given interval. Finally the proposed framework has successfully been ported to an architecture resembling an embedded real-time system.

For a set of tasks depending on each other and represented by a directed acyclic graph, we present a model where the scheduling for endsystems and switches can be combined to find an optimal solution with respect to total transmission time. As the scheduling problem is $\mathcal{NP}$-complete we investigate different methods to improve scalability in order to apply the proposed framework to novel NoC and MPSoC architectures.
At first we model the properties of the scheduling problem in a way to apply modern SAT and SMT solvers respectively. We compute optimal schedules for different systems under consideration. Furthermore, the generation of benchmarks arising from scheduling problems on TT-NoCs is outlined and it is explained how the solvers can be applied either during the design process or on the target system itself. By enabling scheduling at runtime we increase the fault tolerance of the system under consideration. We can increase the flexibility to recover from faults by rescheduling the all required tasks if a component (i.e. an endsystem, a switch or a communication link) fails at runtime. For demonstrating purposes we use an MPSoC emulating target system.
We will report on the evaluation of performance and scalability of the proposed scheduler and suggest different techniques to tackle the $\mathcal{NP}$-complete optimal scheduling problem such as parallelising certain algorithmic instances or relaxing optimality constraints. We will compare runtime and resource requirements of the proposed framework with state-of-the-art-tools such as as the MILP tool CPLEX. Furthermore, we will demonstrate how the proposed scheduler can significantly improve the quality of schedules in comparison to a state-of-the-art approach deploying the popular list-scheduling heuristic if components of the system under investigation become faulty.

1.4 Methodology

After providing the theoretical background on propositional logic, optimization and complexity theory we develop a time discrete model which resembles the communication of the system under consideration. We will then outline how the scheduling problem can be formulated as an optimization problem with constraints and an objective function which is minimized in order to compute the minimal makespan i.e. the shortest possible transmission time for all messages. We will use SAT and SMT solver to compute optimal schedules and verify the solutions using Mixed integer linear programming. We will evaluate the results with respect to runtime and memory footprint and illustrate how the performance of our proposed scheduling framework can be further enhanced by the applying parallelisation.
Due to $\mathcal{NP}$-completeness of the scheduling problem we will also analyse strategies

to partition problems and compute feasible - not necessarily optimal - solutions in parallel.
Finally, we will also evaluate the proposed scheduling framework by comparing it to a scheduler based on heuristical methods called list scheduling (LS). We will apply both schedulers to a fault-tolerant MPSoC architecture and show how our proposed scheduling frameworks outperforms LS with respect to a minimal makespan if the number of faulty components in the system under consideration is increased significantly.

1.5 Thesis Structure

This thesis is structured as follows: In the first Chapter we have already outlined the motivation, objective and contribution of this thesis as well as the methodologies applied.
In Chapter 2 introduces the technical background against this thesis is set. It describes the current state-of-the-art of requirements, challenges and applications of so called real-time embedded systems which become increasingly complex and are crucial for the implementation of safety-critical systems.
Chapter 3 then lays the mathematical and algorithmic foundations needed to model the scheduling problem as an (pseudo-Boolean-)optimization problem. Here we discuss all preliminaries regarding Boolean functions, mathematical optimization and complexity theory.
Chapter 4 we will demonstrate how the scheduling problem can be modelled in a way that allows the application of SAT-solving techniques to compute optimal solutions. The results are verified and evaluated using state-of-the-art tools.
On this basis Chapter 5 discusses enhancements to the proposed model and motivates the use of SMT solvers which are a generalization of SAT solvers. After further evaluation the SMT-based scheduling framework is ported to an MPSoC evaluating target system.
Two possible techniques to parallelize the solving process are discussed in Chapter 6. Both approaches have a significant impact on the runtime of the solvers.
In Section 7 we introduce the concept of fault tolerance and illustrate the advantages of the proposed scheduling framework if components of the architecture under consideration are subject to permanent failures. We will outline how critical deadlines can still be met thus sustaining functionality.
Finally Chapter 8 concludes this thesis and summarizes the results of our contribution. We also provide an outlook on how open problems may be tackled in future.

2 State-of-the-Art

This introductory chapter outlines the technological background in which this thesis is set. We will describe the current state-of-the-art in theory and application of distributed embedded computer systems, which are realized using time-triggered communication networks. In this context the structural elements of an embedded computer system with a time-triggered communication network are explained and a terminology is established. We will continue with an introduction into modern scheduling methods and briefly present how feasible schedules can be computed deploying commercial software.
Especially the computation of optimal solutions to the scheduling problem has been a popular research topic for the past decade and tremendous progress has been achieved in recent years. Therefore, we will conclude this chapter by discussing the influential contributions on this vibrant research subject.

2.1 Applications of Embedded Systems

Digitalisation is one of the major challenges for a modern society. As computers become increasingly dominant in every-day life the most inconspicuous objects are by now equipped with integrated computer chips. Furthermore, the growing demand of digital networking and communication, i.e. Internet of Things (IoT), will lead to increasing advances in technology and the need for small, cheap and yet reliable hardware. Already today embedded computers are by far the most common type of computer in use and experts estimate that nowadays ninety-eight per cent of all computing devices are embedded in different kinds of electronic equipment such as automotive, industrial automation, telecommunications, consumer electronics and health/medical systems [7].
However, there exists no single model for building embedded systems. This is due to many different and, partially, contradicting requirements for instance trade-offs such as resource adequacy versus best-effort strategies or predictability versus flexibility. As a consequence the system model depends strongly on the requirements of the application.
In any given application the purpose of the embedded computer system is defined by the requirements at the interface to the controlled object. Thus we will give more details on the specific requirements of architectures under investigation in the scope of this thesis: We consider *real-time* embedded systems referring to systems gathering their required information at runtime. It is commonly distinguished between two different communication protocols in multicore systems: On the one hand there is *event-triggered* communication which is dynamic and flexible and on

the other end *time-triggered* communication, whose salient features are monitoring ability and temporal predictability. The focus of this work is on the latter, the de facto standard for safety critical applications, explained in more detail in Section 2.3.
Such real-time applications are also called cyber-physical systems in order to reflect the integration of computation and physical processes. Such a distributed embedded system contains a set of node computers (nodes for short). In this work each node is considered a self-contained composite hardware/software subsystem, which communicates with each other over a time-triggered communication network.
At present, time-triggered off-chip networks such as TTP [12], FlexRay [13] and TTEthernet [14] are deployed in automotive, aerospace and railway applications. In recent years, time-triggered (TT) architectures [KB01] have gained momentum for platform-based applications. Time-triggered system architecture, especially FlexRay and TTEthernet are widely used in embedded systems for safety-critical applications. The FlexRay communication standard for instance has gained industry-wide acceptance as the next-generation automotive networking standard. Likewise, multi-processor architectures based on time-triggered on-chip networks have been introduced for safety-critical systems (e.g., GENESYS MPSoC [15], AEthereal [16]).
In the next section we will outline a novel design paradigm how these systems are actually realized in practice meeting the increasing demands of complexity, functionality and efficiency. We will then continue and discuss how the proposed architectural properties can be exploited to compute optimal schedules on the device itself.

2.2 SoC, NoC and MPSoC Architectures

The increasing complexity of multicore systems, together with the unending demand for higher performance and less energy consumption, keeps pushing the trend of shrinking device sizes and increasing the number of endsystems integrated on a single chip. Moore's famous law does not only describe the increasing density of transistors permitted by technological advances. It also imposes new requirements and challenges. System complexity increases at the same speed. Nowadays systems could never be designed using the same approaches applied 30 years ago [17]. New architectures are and must be continuously conceived. Hence Multiprocessor systems-on-chips (MPSoCs) are the latest incarnation of very large-scale integration (VLSI) technology. A single integrated circuit can contain over 100 million transistors, and the International Technology Roadmap for Semiconductors predicts that chips with a billion transistors are within reach [18].

In order to comply to the needs outlined designers are forced to move beyond logic design into computer architecture. The demands placed on these chips by applications require designers to face problems not confronted by traditional computer architecture: harsh operational conditions, very low-power operation and as in the scope of this work real-time deadlines. These opportunities and challenges make MPSoC design an important field of research especially regarding communication paradigms on these architectures.
In order to define MPSoC we first have to define a system-on-chip (SoC). A SoC is an integrated circuit that implements most of the functions of a complete electronic system. The most fundamental characteristic of a SoC is complexity. A memory chip may have many transistors, but its regular structure makes it a component and not a system. Exactly what components are assembled on the SoC varies with the application. It is predicted that future embedded SoCs will probably be made up of tens or hundreds of heterogeneous endsystems, which will be able to execute one parallel application or even several applications running in parallel [19].
SoCs first came up in the mid-nineties. One of the main challenges then was the way to interconnect all these devices efficiently as due to the increasing complexity the bus interconnect structure soon reached its limits and was no longer appropriate to meet the challenges of modern design patterns. Therefore a new interconnection paradigm emerged, the so-called Network-on-Chip (NoC). Basically a NoC can be described as a communication subsystem on an integrated circuit in a SoC. Therefore NoC architectures can be regarded as the solution for the scalability problem of SoCs.
SoCs can be found in many product categories ranging from every day consumer devices to industrial systems:

- Cell phones use several programmable processors to handle the signalprocessing and protocol tasks required by telephony. These architectures must be designed to operate at the very low-power levels provided by batteries.
- Telecommunications and networking use specialized SoCs, such as network processors, to handle the huge data rates presented by modern transmission equipment.
- Digital televisions and set-top boxes use sophisticated multiprocessors to perform real-time video and audio decoding and user interface functions.
- Television production equipment uses systems-on-chips to encode videos.

Now we can define an MPSoC as a system-on-chip containing multiple processors (CPUs). In practice, most SoCs are MPSoCs because it is too difficult to design a complex system-on-chip without making use of multiple CPUs. In MPSoC we can in general distinguish between two communication protocols i.e. an event-triggered communication protocol where messages are triggered by an external event opposed to a time-triggered communication protocol. This thesis will focus on the latter. Details are provided in Section 2.3.

Considering architectures equipped with TT communication protocols offers several advantages i.e. monitoring aspects or the capability of providing fault tolerance: A faulty endsystem for example cannot affect the message exchange between other endsystems if the execution of jobs is scheduled to different parts of the on-chip-network still operating free from defects. On-chip fault isolation is a prerequisite for fault-tolerance through active redundancy [20] and the integration of mixed-criticality applications on a single chip [21]. This concepts and subsequent applications of our proposed framework will be introduced extensively in Chapter 7.2.

Just like SoCs MPSoCs have been used to realize a a wide range of new products and services in many areas. Their popularity arises from the average high performance. MPSoC design has been regarded an emerging research area for the last few years [22] and subsequently tremendous research is being conducted on MPSoCs. Areas of special interest regard critical issues like computational capabilities, programmability, flexibility, scalability and power consumption. Using parallel programming techniques, more efficient computational capabilities can be achieved. Such parallel task execution models have been studied for parallel computing machines during the past decades.

There are numerous examples where MPSoCs are already applied in practice especially in cost-sensitive, real-time systems: Consider for example cellular phone, game stations or high-definition digital television (HDTV). Due to the application scenarios designers face real-time performance requirements as well as stringent cost requirements (chip area, energy consumption). To satisfy those requirements, applications executed on MPSoCs need to be optimized in terms of code size, energy consumption and execution time. In this thesis we will focus on the latter and introduce a scheduling framework not only tailored to meet the requirements of MPSoC architectures described but are also executable on the target architecture itself thus meeting the requirements for software designers outlined [23].

In conclusion it can be said that MPSoC is an emerging research area and deservedly in the focus of industry and academia. As there is a huge variety of different approaches we have confined for the scope of this work that all processors of the MPSoC are assumed to be homogeneous. We will emphasize our work on scalability of scheduling within the architecture under consideration. We will investigate how the optimal scheduling problem on MPSoCs can be solved efficiently deploying parallel computation. In this context we examine how the multicore structure can be exploited for parallel execution.

2.3 Time-Triggered Communication Protocol

For distributed embedded systems there currently exist two fundamentally different paradigms for the design of real-time systems: In an event triggered system a processing activity is initiated as a consequence of internal or external stimuli such as the reception of a message or a rise of temperature. On the other hand a time-triggered communication protocol strictly controls the entire communication within a network determining exactly at which point in time a message is sent and received. This thesis will focus on the latter paradigm because time-triggered networks [12, 13, 14] play an important role in *safety-critical systems*, where their inherent properties such as temporal predictability, fault tolerance and composability [6] improve safety and reduce certification costs [7]. Time-triggered networks use timeplans, which define the points in time of all message exchanges with respect to a global time base. In multi-cluster time-triggered systems of large embedded systems (e.g., automotive, avionics), conflict-free paths along switches and endsystems are defined for each message. The conflict-free temporal and spatial allocation of communication resources in combination with an intelligent communication network (e.g., local and central guardians) prevents interference between messages from different components upon integration and in the presence of faults.

For the time-triggered communication network different topologies can be distinguished such as bus, star and ring topologies. Independently of the topology, different redundancy degrees of the communication network are possible. A single communication channel is typically used in non safety-critical applications. In *safety critical systems*, redundant communication channels support the masking of channel failures. For example, the computational components can be allocated to the nodes of a distributed system, where the network serves as the communication infrastructure between these components. Another possibility, which has been enabled by the advances of Multi-Processor System-on-a-Chips (MPSoCs), is the allocation of computational components to endsystems that are interconnected switches [17] which will be discussed in more detail in the next chapter.

Time-triggered, safety critical, distributed systems, the scope of this work, have established themselves as a de facto standard for numerous applications i.e. chassis control systems and power train communication. Therefore it seem sensible to analyse the impact of optimal schedules on the performance of the systems under investigation.

2.4 Scheduling in Time-Triggered Distributed Systems

Scheduling is the act of creating a schedule, which is a timetable for planned occurrences. Scheduling may also involve allocating resources to activities over time. A scheduling problem can be viewed as a constraint satisfaction problem or as a constrained optimization problem [24], but regardless of how it is viewed, a scheduling problem is defined by:

1. A set of time intervals, i.e. definitions of activities, operations or tasks to be completed.
2. A set of temporal constraints, i.e. definitions of possible relationships between the start and end times of the intervals.
3. A set of specialized constraints i.e. definitions of the complex relationships on a set of intervals due to the state and finite capacity of resources.

In the state-of-the-art scheduling techniques for time-triggered networks have been introduced with different optimal techniques and heuristics. However, the scalability of optimal techniques is limited with the number of endsystems, switches and messages. Even in case of sufficient resources, heuristics cannot guarantee the computation of a feasible schedule. Thus scheduling is important but $\mathcal{NP}$-complete. Therefore scheduling has been a very important research subject for the past thirty years. Only recently a very attractive alternative to simulation and testing has emerged [25]: The application of formal verification tools to the scheduling problem. While simulation and testing explore only some of the possible behaviours and scenarios, formal verification conducts an exhaustive exploration of all possibilities and thus guarantees that an optimal solution is found, if it actually exists.
In this thesis, we deploy tools from the area of formal verification, such as SAT and SMT solvers, by which a desired behavioural property of a defined system model is analysed. Therefore we will develop a time-discrete model representing the tasks the on-chip-system under investigation has to perform. We consider logical dependencies as well as architectural properties, which are both outlined in Chapter 4.
One of the main objectives of this work is the application of our proposed scheduling framework on real-time systems. A real-time system must produce the intended results at the intended instant in real time [26]. Time-triggered real-time systems

are popular in safety-critical applications where temporal predictability is an important concern. In order to describe the tasks which have to be performed on the system under consideration, we use the term jobs to denote a logical unit of computation and we represent a program as a set of jobs. In literature, terms as jobs, task and processes. are used interchangeably. In many traditional real-time applications, especially in the aeronautic or automotive sector, it is assumed that the set of tasks does not change at runtime. We adopt this paradigm and throughout this thesis it is our aim to schedule a set J of n jobs, i.e.

$$J = \{j_1, j_2, \cdots j_n\}.$$

The so-called **job shop scheduling problem** is an optimization problem. It attempts to find a minimal makespan, where the **makespan** denotes the total length of the schedule. Therefore the makespan depends on the allocation of jobs to endsystems and the routes of the time-triggered messages sent between them. Both allocation and paths are fully determined by the proposed scheduling framework. Details on optimization problems and their complex challenges can be gathered from Sections 3.2 and 3.3.
In the literature it is distinguished between four different scheduling classes, which we will briefly outline:

1. **Static scheduling**: all scheduling decisions are based on fixed parameters, assigned to tasks before their activation. Static scheduling needs a priori knowledge of all task attributes. Therefore it is less flexible.

2. **Dynamic scheduling**: all scheduling decisions are based on dynamic parameters that might change at runtime. Dynamic scheduling can provide a better processor utilization and supports non-predicted events such as failing components, but it has a higher runtime overhead than static scheduling.

3. **Off-line scheduling**: All scheduling decisions are computed at compile time and stored in a dispatcher table which is located in a Trusted Research Manager (TRM), d device that will not become faulty by construction. At runtime no scheduler is needed, but only a dispatcher which takes the next entry from the table. Off-line scheduling is also called table-driven scheduling, which incorporates a table determining which tasks to execute at which points in time. Thus, feasibility is proven constructively. Off-line scheduling methods are capable of managing distributed applications with complex constraints (e.g. precedence or end-to-end deadlines). On the other hand, the a-priori knowledge about all system activities may be hard or impossible to obtain. Its rigidity enables deterministic behaviour, but drastically limits flexibility.

4. **On-line scheduling**: All scheduling decisions are made at runtime, meaning that the scheduler decides when a new task is released or when a task terminates its execution. Nevertheless, on-line scheduling anomalies have to be handled.

This thesis focuses on static scheduling, because the architecture and allocation of tasks to endsystems. The off-line scheduling approach is the one usually associated with time-triggered architectures. When enhancing the proposed scheduler, however, we will also consider dynamic, on-line scheduling, especially when regarding fault tolerant architectures.
Currently timeplans are computed during the design process employing high performance computers and state-of-the-art software like IBM ILOG CPLEX Optimization Studio (often informally referred to simply as CPLEX) are deployed. We will outline how CPLEX solves scheduling problems in the next section.
Cyper-physical systems are also equipped with redundant alternative schedules to react to system failures. However these errors can never be covered completely and non-predictable failures may still cause permanent errors of the system. As memory capacities on an embedded system are limited, an effective way to compute alternative schedules meeting all deadline constraints have to be developed. Especially if optimal solutions are required and state-of-the-art-solvers cannot be deployed on the architecture under consideration due to the hardware limitations outlined above the process can be very time-consuming. Otherwise if feasible solutions are sufficient, it may prove sensible to apply heuristics to the scheduling problem. This may significantly reduce runtime but may fail to find solutions meeting all deadline constraints even if they exist.

2.5 MILP-based Scheduling

Scheduling is known to be a tough challenge which may prove to be time-consuming, especially if the architecture under investigation relies on optimal schedules. Therefore a common way to compute possible schedules in practice is the application of high-performance computer clusters during the design process. Hence static off-line scheduling - see previous section - is mainly deployed in the state-of-the-art. In this process efficient hard- and software computes a number of valid schedules which are then copied onto the device and stored in a TRM [27]. If components fail or the system has to be reconfigured the schedules are substituted. The TRM accepts proposals for new communication schedules, which are provided by the components. The TRM checks the validity of a supplied time-triggered schedule by checking whether it is free of collisions. However, as the computation of schedules

is not dynamic there is a high probability that the entire system may fail if the schedules stored in the TRM are not capable to mask the fault reported.
The main software adopted in the research area of embedded systems to compute schedules is IBM ILOG CPLEX Optimization Studio. CPLEX Studio is a rapid development system for optimization models with interfaces to embed models into standalone applications. We will briefly refer to this software as CPLEX deploy the version released in 2014 [28].
CPLEX is an optimization software package based originally on the Simplex algorithm. Today it solves integer programming problems using the simplex method or the barrier interior point method, convex and non-convex quadratic programming problems and convex quadratically constrained problems. CPLEX is referred to be the leading software product in the field of mathematical optimization and operational research. This work mainly relies on CPLEX as a reference solver which is on the one hand deployed to validate our results. On the other hand we compare the memory footprint as well as the computation time of our proposed scheduling framework to CPLEX. In contrast to CPLEX all other tools proposed in the course of this work are open source and free of charge.

2.6 Related Work

Time-triggered networks play an important role in safety-critical systems, where their inherent properties such as temporal predictability, fault tolerance and composability [6] improve safety and reduce certification costs [7]. Time-triggered networks use timeplans, which define the points in time of all message exchanges with respect to a global time base. In multi-cluster time-triggered systems of large embedded systems (e.g., automotive, avionics), conflict-free paths along switches and endsystems are defined for each message. The conflict-free temporal and spatial allocation of communication resources in combination with an intelligent communication network (e.g., local and central guardians) prevents interference between messages from different components upon integration and in the presence of faults.
However, the computation of the timeplans is time consuming and computationally complex [8]. Feasible schedules need to avoid conflicts of communication resources, meet deadlines, satisfy precedence constraints and ensure implicit synchronization [9]. Different optimal and heuristic scheduling algorithms have been devised in the state-of-the art (e.g., enumerative methods, mathematical programming, simulated annealing, genetic algorithms, tabu search, neural networks). While the runtime of optimal algorithms becomes prohibitive upon large problem sizes, heuristics often fail to find feasible schedules.

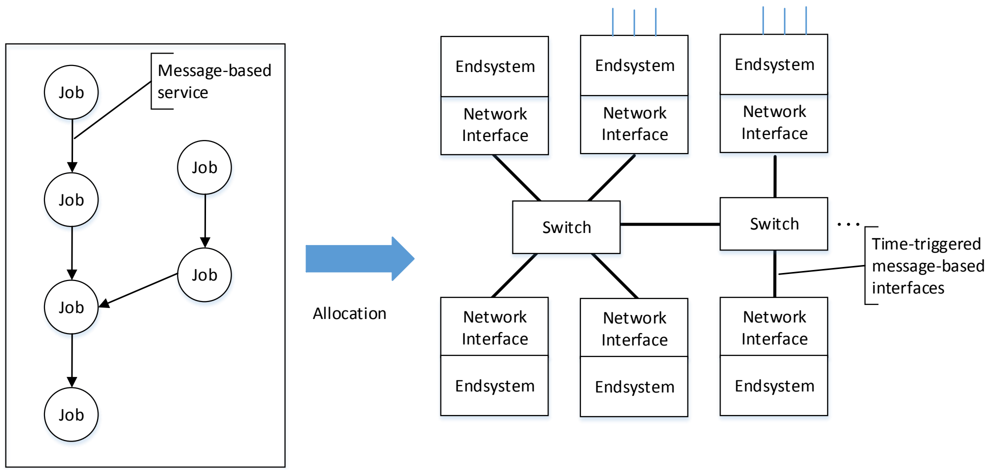


Figure 1: Logical Application Model (left) mapped to Physical Multicore Platform (right)

For the past five years the application of tools from hard- and software verification to the scheduling problem has been studied extensively [25]. Especially so-called SAT solvers have been applied. Modern SAT solvers use heuristics to efficiently compute solutions for the Boolean Satisfiability Problem, one of most-researched NP-complete problem in computer science [29] - see also Section 3.3. SAT solvers have demonstrated to be effective in solving $\mathcal{NP}$-complete problems for many applications, including planning, circuit design and also scheduling [30]. This widespread adoption is the result of the efficiency gains made during the last decade [31]. Modern SAT-solvers use different heuristics to check whether a given Boolean formula $F(x_1, \cdots, x_n)$ is satisfiable and if so returns a satisfying assignment that makes F true.
SMT solvers, which extend SAT with richer theories, can handle an even broader range of problems [32] by allowing the input of (in)-equalities expressed in classical first-order logic. SMT solvers have proved to be powerful and expressive backend engines for formal verification in many contexts, including the verification of software, hardware, and of timed and hybrid systems [33] and have been successfully used to compute feasible schedules for time-triggered networks [25]. Because of the high computation time required by existing schedulers, a technique that applies an SMT-solver to compute an optimal solution to a scheduling problem with time-triggered messages is proposed. It has been previously investigated how the scheduling problem can be translated into a conjunctive normal form (CNF) allowing the application of a SAT-solver to compute satisfiable solutions that were optimal with respect to the total transmission time to receive all messages [2].

This model was however limited by the constant allocation of jobs to nodes. In this work the model is enhanced such that an arbitrary allocation of jobs to endsystems is allowed. The proposed scheduler is optimizing the allocation of jobs as well as the message paths with respect to the minimal transmission times. Furthermore the proposed SMT scheduler is able to handle multiple messages from the same sender which represents a more realistic behaviour.
In the past SMT solvers have been successfully applied to verify the TT-Ethernet synchronisation function [25]. Furthermore SMT solvers were also applicable for the generation of (even large-scale) time-triggered schedules [34]. This scheduling model proposed in [34] has also recently been evaluated using the SMT solver YICES2 [35].
Due to its multicore heritage MPSoC can also be deployed for parallel computing [36] and high performance parallel-programming frameworks for MPSoC architectures such as [37] have recently been developed. Hence we will also study the impact of parallelization of the proposed scheduling framework considering different approaches for the computation of optimal and feasible schedules. We have already previously investigated the parallel distribution of different problem instances (i.e. SAT benchmarks) to different endsystems [1]. Therefore distributing the scheduling problem and approaches to solve the partitions in parallel is also analysed in the scope of this work.
For years parallel computing has been a popular research subject. Since the switch to multicore and manycore processors industry has laid out a roadmap for multicore designs [38]. Also the use of SAT solvers [39, 40] in various application areas is on the rise since in addition to the traditional hardware and software verification domains, SAT solvers are gaining popularity in new domains. This widespread adoption is the result of the efficiency gains made during the last decade [41]. Tremendous efforts have been taken to combine the two disciplines and to apply the concept of parallelisation to improve performance of SAT solvers even further [42].
In addition to this industrial partners have defined key challenges in parallel SAT solving [41], which shows how important efficient SAT solving methods are for real-world applications.

3 Concepts and Terms

This chapter presents the mathematical terms and concepts used in this thesis: we explicitly introduce three basic concepts: propositional logic, mathematical optimization including the special case of pseudo-Boolean optimization problems and complexity classes for evaluation purposes.
We begin with the introduction of an algebra whose original purpose, dating back to Aristotle, was to model reasoning. In more recent times this algebra, like many other algebras, has proved useful as a design tool. We will outline the formalities and notations used to describe the Scheduling Problem using Boolean Functions and their representations only. In the same way it is demonstrated how the Scheduling Problem can be transformed into a mathematical optimization problems using so-called pseudo-Boolean variables only. Because of this we will be able to apply state-of-the-art tools from the field of computer verification to find solutions, to the Scheduling problem. In order to classify these solutions we define optimization problems. The chapter is concluded with an introduction to complexity theory which is needed to evaluate the scalability and sustainability of our proposed scheduling framework.

3.1 Boolean Functions and their representations

In this section we will define a Boolean algebra, introduce logical expressions with Boolean-valued operands and logical operators such as `AND`, `OR` and `NOT` operating on Boolean values. The sequential arrangement is inspired by [43].
A Boolean value is one with two choices: true or false, yes or no, 1 or 0. In computer science the Boolean data type is a data type, having two values intended to represent the truth values of logic in an algebra. This algebra is often called Boolean algebra after George Boole, the logician who first defined the underlying concept formally. In mathematical logic, a *propositional variable* (also called a sentential variable or sentential letter) is a variable which can either be true or false. Propositional variables are the basic building-blocks of propositional formulae, used in propositional and higher order logics.
Propositional logic is a mathematical model that allows us to reason about the truth or falsehood of logical expressions. This can be formalized by defining logical expressions as follows:

Definition 3.1. *Propositional variables and the logical constants,* `TRUE` *and* `FALSE`*, are* ***logical expressions****. These are the so-called atomic operands.*

Remark 3.2. *Let E and F be logical expression. Then so are:*

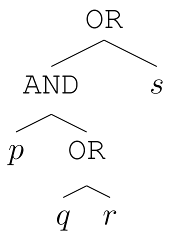


Figure 2: Expression tree for the logical expression p AND $(q$ OR $r)$ OR s

- *E AND F. The value of this expression is TRUE if and only if both E and F are TRUE. It is FALSE, otherwise.*
- *E OR F. The value of this expression is TRUE if either E or F or both are TRUE. It is FALSE, if both E and F are FALSE.*
- *$\overline{E}$. The value of this expression is TRUE if E is FALSE, and vice versa.*

Thus logical expressions can be built from the binary infix operators AND and OR as well as the unary prefix operator NOT.
When all of the propositional variables in a logical expression are assigned truth values, the expression itself acquires a truth value. We can evaluate a logical expression just as we would with arithmetic or a relational expression. A good way to visualize this fact is the deployment of so-called *expression trees* as depicted in Figure 2. For a given truth assignment of each variable the tree is worked up from the bottom producing truth values for each node. The truth value at the root is then the truth value of the expression as a whole.
Formally a logical expression can be described as a function from the values of its arguments to a value of the whole expression. To be more precise a logical expression's meaning is a function that takes truth assignments as arguments and returns either TRUE or FALSE. Such functions are called Boolean functions and defined below. Like arithmetic expressions, Boolean expressions can be thought of as sets of pairs. The first component of each pair is a truth assignment, that is a tuple giving the truth value of each propositional variable in some specified order. The second component of the pair is the value of the expression for that truth assignment.
Before we can explicitly define Boolean functions we have to introduce Boolean domains:

Definition 3.3. *A **Boolean domain** is a set consisting of exactly two elements usually written as $\{0, 1\}$ or $\{$TRUE, FALSE$\}$.*

Throughout this work we will denote Boolean domains as $\mathbb{B}$. Hence Boolean functions can be defined as follows:

Definition 3.4. *A* ***Boolean function*** *is a function of the form*

$$f : \mathbb{B}^k \longrightarrow \mathbb{B},$$

where $\mathbb{B}$ is a Boolean domain. The non-negative integer k is called the ***arity*** *of the Boolean function. If $k = 0$ the Boolean function f is constant which means it always evaluates to the same element of $\mathbb{B}$.*

In Boolean functions we will use the following symbols to denote infix operators: $\vee$ for `OR` and $\wedge$ for `AND`. Thus our introductory example can be expressed as a Boolean function in the following way:

$$f : \mathbb{B}^4 \longrightarrow \mathbb{B} : \ (p, q, r, s) \in \mathbb{B}^4 \mapsto p \vee (q \wedge r) \vee s.$$

It is convenient to display a Boolean function as a *truth table*, in which the rows correspond to all possible combinations of truth values for the arguments. There is a column for each argument and a column for the value of the function. Our introductory example is evaluated in the truth table depicted in Table 1.

The example illustrates that four variables require $2^4 = 16$ assignments in order to fully evaluate all possibilities. This conjunction holds and in fact the truth table for Boolean function of arity k consists of 2^k row, one for each truth assignment. Therefore it is highly complex to compare two Boolean functions as well as to fully evaluate Boolean functions or to prove that no valid truth assignment exits for a given Boolean function. We will discuss the complexity of this problem in more detail in Section 3.3.
The importance of Boolean algebras for computer science goes back to 1938, when it was proven that a two-valued Boolean algebra can describe the operation of two-valued electrical switching circuits [44]. Table 2 depicts the truth table for the $2^{2^2} = 16$ possible Boolean functions of two binary variables of the form:

$$f : \mathbb{B}^2 \longrightarrow \mathbb{B}, \ (p, q) \in \mathbb{B}^2 \mapsto f(p, q).$$

In addition to the two Boolean functions we have already met, hence `AND` i.e. F_7 and `OR` i.e. F_1, we want to draw attention to five additional Boolean functions of two arguments, which will prove useful in the context of this thesis:

p	q	r	s	$q \vee r$	$p \wedge (q \vee r)$	$p \wedge (q \vee r) \vee s$
1	1	1	1	1	1	1
1	1	1	0	1	1	1
1	1	0	1	1	1	1
1	1	0	0	1	1	1
1	0	1	1	1	1	1
1	0	1	0	1	1	1
1	0	0	1	0	0	1
1	0	0	0	0	0	0
0	1	1	1	1	0	1
0	1	1	0	1	0	0
0	1	0	1	1	0	1
0	1	0	0	1	0	0
0	0	1	1	1	0	1
0	0	1	0	1	0	0
0	0	0	1	0	0	1
0	0	0	0	0	0	0

Table 1: Truth Table for for the logical expression $p \wedge (q \vee r) \vee s$

p	q	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0

p	q	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0

Table 2: Truth Table for all possible Boolean functions $F_i, i \in \{0, 1, \cdots 15\}$ of two binary variables

- F_4, denoted with Boolean expression $\overline{p} \vee q$ also referred to as *Implication* in propositional logic, denoted with $\Longrightarrow$. The respective truth table is also depicted in Table 3a.
- F_6, denoted with Boolean expression $(p \wedge q) \vee (\overline{p} \wedge \overline{q})$ also referred to as *Equivalence* in propositional logic, denoted with $\Leftrightarrow$ or `XNOR` in digital logic. The respective truth table is also depicted in Table 3b.
- F_8 denoted with Boolean expression $\overline{p \wedge q}$ also referred to as `NAND`-operator. The respective truth table is also depicted in Table 3c.
- F_9 denoted with the Boolean expression $(p \wedge \overline{q}) \vee (\overline{p} \wedge q)$ also expressed as **Exclusive or**, denoted $\oplus$ in digital logic. The respective truth table is depicted in Table 3d.

p	q	$\overline{p} \vee q$
1	1	1
1	0	0
0	1	1
0	0	1

(a) F_4 i.e.$\overline{p} \vee q$

p	q	$(p \wedge q) \vee (\overline{p} \wedge \overline{q}$
1	1	1
1	0	0
0	1	0
0	0	1

(b) F_6 i.e. $(p \wedge q) \vee (\overline{p} \wedge \overline{q}$

p	q	$\overline{p \wedge q}$
1	1	0
1	0	0
0	1	0
0	0	1

(c) F_8 i.e. $\overline{p \wedge q}$

p	q	$(p \wedge \overline{q}) \vee (\overline{p} \wedge q)$
1	1	0
1	0	1
0	1	1
0	0	0

(d) F_9 i.e. $(p \wedge \overline{q}) \vee (\overline{p} \wedge q)$

p	q	$\overline{(p \vee q)}$
1	1	0
1	0	1
0	1	1
0	0	1

(e) F_{14} i.e. $\overline{(p \vee q)}$

Table 3: Truth Tables for Additional Logical Operators.

The representation of a Boolean function is not unique. In fact it can be represented by an infinite number of Boolean formulae. Consider for example the simple Boolean function $f = p \wedge q$, which is the same as

$$f = q \wedge p = (p \wedge q) \wedge (p \wedge q) = \cdots$$

A truth table is the simplest way of representing a Boolean function and starting from here any Boolean function whatsoever can be represented by a logical expression using the operators `AND, OR`, and `NOT`. Finding the simplest expression for a given Boolean function is generally hard. However, we can easily construct some expression for any Boolean function. Starting with the truth table for the function, we construct a logical expression of the form

$$m_1 \vee m_2 \vee \cdots \vee m_n.$$

Each m_i is a term that corresponds to one of the rows in the truth table evaluating to 1. The terms m_i are of special form and have to obey certain rules, which we want to outline in the following:
A **literal** is a Boolean expression that is either a single propositional variable, such as p, or negated variable `NOT` p, which we will in future denote $\overline{p}$. Consider a truth table consisting of k variables then each m_i is the composition of k literals. If in row i the variable p is of value 1 select the literal p otherwise choose $\overline{p}$. If we continue in this fashion m_i is the logical conjunction of the literals. Clearly

m_i can only evaluate to `TRUE` if all variables have the values that appear in the corresponding row of the truth table.

Definition 3.5. *If m_i is constructed in the way described m_i will is called* ***minterm****. If all minterms are combined using the operator* `OR` *the resulting expression*

$$m_1 \vee m_2 \vee \cdots \vee m_n$$

is called the ***disjunctive normal form (DNF)****.*
The expression describes the Boolean function under consideration, because it has the value 1 exactly when there is a minterm with value 1. A minterm cannot be 1 unless the values of the variables correspond to the row of the truth table for that minterm. Thus the representation is correct.
In a dual way we can also construct a representation by defining a **maxterm**, which is the disjunction of those literals disagreeing with the value of one of the argument variables in that row. More precisely this means: if the row has value 0 for variable p, select the literal p and if the value of that row for p is 1, choose $\overline{p}$. If the maxterms are combined conjunctively the expression is described in so-called **conjunctive normal form (CNF)**.
We can summarize the above as follows: A Boolean function f can be represented using its minterms and maxterms. This disjunction of all minterms, where F is mapped to 1 results in a DNF. Similarly, the conjunction of all maxterms, where f evaluates to 0, results in a CNF.
As an example consider again

$$f : \mathbb{B}^4 \longrightarrow \mathbb{B} : \ (p, q, r, s) \in \mathbb{B}^4 \mapsto p \vee (q \wedge r) \vee s.$$

whose truth table has been illustrated in Table 1. We extend this truth table by adding an extra column to describe the corresponding minterms and maxterms. The result is depicted in Table 4. Hence we can now formulate the two normal forms:

1. DNF: $f = m_1 \vee m_2 \vee m_3 \vee m_4 \vee m_5 \vee m_7 \vee m_9 \vee m_{11} \vee m_{13} \vee m_{15}$.

2. CNF: $f = m_8 \wedge m_{10} \wedge m_{12} \wedge m_{14} \wedge m_{16}$.

In the same way we can also formulate normal forms for the Boolean functions explicitly introduced in Table 3:

- F_4:

    1. DNF: $(p \wedge q) \vee (\overline{p} \wedge q) \vee (\overline{p} \wedge \overline{q})$.

2. CNF: $(\overline{p} \vee q)$.

- F_6:
  1. DNF: $(p \wedge q) \vee (\overline{p} \wedge \overline{q})$.
  2. CNF: $(\overline{p} \vee q) \wedge (p \vee \overline{q})$
- F_8:
  1. DNF: $(\overline{p} \wedge \overline{q})$.
  2. CNF: $(\overline{p} \vee q) \wedge (p \vee \overline{q}) \wedge (\overline{p} \vee \overline{q})$
- F_9:
  1. DNF: $(p \wedge q) \vee (\overline{p} \wedge \overline{q})$.
  2. CNF: $(\overline{p} \vee \overline{q}) \wedge (p \vee q)$
- F_{14}:
  1. DNF: $(p \wedge \overline{q}) \vee (\overline{p} \wedge q) \vee (\overline{p} \wedge \overline{q})$.
  2. CNF: $(\overline{p} \vee \overline{q})$

We are now in the position to find a representation in CNF or DNF for every Boolean function. Throughout this work we will restrict our attention to CNFs. Finding assignments to the variables of a Boolean formula expressed in CNF is a very prominent research subject. These problems are called **Boolean Satisfiability Problems** or **SAT**-problems. They ask whether for a given Boolean formula there exits an assignment for all variables involved such that the whole expression evaluates to `TRUE`. In other words, the SAT-problem asks whether the variables of a given Boolean formula can be consistently replaced by the values `TRUE` or `FALSE` in such a way that the formula evaluates to `TRUE`. If such an assignment exists the formula is called **satisfiable** or **sat**. On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the formula is thus **unsatisfiable** or **unsat**. This decision problem is of central importance in various areas of computer science, including theoretical computer science, complexity theory, algorithmics, cryptography and artificial intelligence.

In the following chapters we will propose the application of modern SAT solvers to the scheduling problem. SAT solvers are very powerful tools from the field of Electronic Design Automation and have contributed to dramatic advances in our ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints (i.e. clauses). An extension that has gained significant popularity since 2003 is Satisfiability modulo theories (SMT) that can enrich CNF formulae with linear constraints, arrays, all-different constraints or uninterpreted functions.

In Chapter 4 we will apply the techniques introduced here to formulate equations modelling the scheduling problem and deploy SAT solvers to find solutions. This approach is enhanced in Chapter 5 where we will also apply SMT solvers to the scheduling problem.

p	q	r	s	$q \vee r$	$p \wedge (q \vee r)$	$p \wedge (q \vee r) \vee s$	terms
1	1	1	1	1	1	1	$m_1 = p \wedge q \wedge r \wedge s$
1	1	1	0	1	1	1	$m_2 = p \wedge q \wedge r \wedge \overline{s}$
1	1	0	1	1	1	1	$m_3 = p \wedge q \wedge \overline{r} \wedge s$
1	1	0	0	1	1	1	$m_4 = p \wedge q \wedge \overline{r} \wedge \overline{s}$
1	0	1	1	1	1	1	$m_5 = p \wedge \overline{q} \wedge r \wedge s$
1	0	1	0	1	1	1	$m_6 = p \wedge \overline{q} \wedge r \wedge \overline{s}$
1	0	0	1	0	0	1	$m_7 = p \wedge \overline{q} \wedge \overline{r} \wedge s$
1	0	0	0	0	0	0	$m_8 = \overline{p} \vee q \vee r \vee s$
0	1	1	1	1	0	1	$m_9 = \overline{p} \wedge q \wedge r \wedge s$
0	1	1	0	1	0	0	$m_{10} = p \vee \overline{q} \vee \overline{r} \vee s$
0	1	0	1	1	0	1	$m_{11} = \overline{p} \wedge q \wedge \overline{r} \wedge s$
0	1	0	0	1	0	0	$m_{12} = p \wedge \overline{q} \wedge r \wedge s$
0	0	1	1	1	0	1	$m_{13} = \overline{p} \wedge \overline{q} \wedge r \wedge s$
0	0	1	0	1	0	0	$m_{14} = p \vee q \vee \overline{r} \vee s$
0	0	0	1	0	0	1	$m_{15} = \overline{p} \wedge \overline{q} \wedge \overline{r} \wedge s$
0	0	0	0	0	0	0	$m_{16} = p \vee q \vee r \vee s$

Table 4: Truth Table for the logical expression $p \wedge (q \vee r) \vee s$ including minterms and maxterms.

3.2 Optimization

This subsection presents the concept of mathematical optimization which we want to deploy in order to compute optimal communication schedules for the architectures under investigation. This introduction to optimization problems has been inspired by [45].

Mathematical Optimisation

An amazing variety of practical problems involving decision making (or system design, analysis and operation) can be cast in the form of a mathematical optimisation problem. Indeed, mathematical optimisation has become an important concept in many areas. It is widely used in engineering, in electronic design automation, automatic control systems, and optimal design problems arising in civil, chemical, mechanical, and aerospace engineering. optimisation is used for problems arising in network design and operation, finance, supply chain management, scheduling, and many other areas.
In the scope of this work we propose a way to find the best solution to the scheduling problem as with the proliferation of computers embedded in products, we have seen a rapid growth in embedded optimisation. In these embedded applications, optimisation is used to automatically make real-time choices and even carry out the associated actions with no (or little) human intervention or oversight.
Embedded real-time optimisation raises some new challenges: in particular, it requires solution methods that are extremely reliable, and solve problems in a predictable amount of time and memory. The proposed optimal scheduling framework tackles both challenges and results in significant advancements with respect to computation times and memory footprints.
Linear Programs play a central role in the modelling of optimisation problems and therefore we will also go into further detail on the properties of CPLEX, the state-of-the-art-solver introduced in Section 2.5.
In general a mathematical optimisation problem can be defined as follows:

Definition 3.6. *A **mathematical optimisation problem** has the form:*

$$\textit{minimize } f_0(x)$$
$$\textit{subject to } f_i(x) \leq b_i,\ i = 1 \cdots, m.$$

where

- $f_0 : \mathbb{R}^n \longrightarrow \mathbb{R}$ *is called the **objective function**,*

- *$f_i : \mathbb{R}^n \longrightarrow \mathbb{R},\ i = 1, \cdots, m$ are called **constraint functions** or **constraints**,*
- *$x = (x_1, x_2, \cdots x_n)$ is called the **optimisation variable***
- *and the constants $b_1, b_2, \cdots, b_n$ are called the **bounds of the constraints**.*

Definition 3.7. *A vector x^* is called **optimal** if it has the smallest objective value among all vectors that satisfy the constraints, i.e.:*

$$\forall z \in \mathbb{R}^n \textit{ with } f_1(z) \leq b_1, f_2(z) \leq b_2, \cdots, f_m(z) \leq b_m : f_0(z) \geq f_0(x^*).$$

The optimisation problem is an abstraction of the problem of making the best possible choice of a vector in $\mathbb{R}^n$ from a set of candidate choices. The variable x represents the choice made; the constraints $f_i(x) \leq b_i$ represent firm requirements or specifications that limit the possible choices, and the objective value $f_0(x)$ represents the cost of choosing x. (We can also think of $f_0(x)$ as representing the value, or utility, of choosing x.)
A solution of the optimisation problem corresponds to a choice that has minimum cost (or maximum utility), among all choices that meet the firm requirements.

Linear Optimisation and Simplex

As mentioned above an important class of optimisation problems is linear programming (LP), in which the objective and all constraint functions are linear. Corresponding to Definition 3.6 we can define linear optimisation problems:

Definition 3.8. *A **linear optimisation problem** is of the following form:*

$$\textit{minimize } c^T x$$
$$\textit{subject to } \alpha_i^T x \leq b_i,\ i = 1 \cdots, m.$$

where the vectors $c, a_1, \cdots, a_n \in \mathbb{R}^n$ and scalars $b_1, \cdots, b_m$ are problem parameters specifying the objective and constraint functions.

There is no simple analytical formula for the solution of a linear problem (as there is for a least-squares problem), but there are a variety of very effective methods for solving them, including the so-called simplex method, which is a common tool included in state-of-the-art optimizing software. As the tools proposed in this thesis also rely on this popular method we will outline its functionality in the following. First of all we can determine that every LP can be formulated in a standard form suitable for algorithmic implementations. In the standard form the objective function is maximized and formulated as an equation referred to as **row 0**, the constraints are equalities and the variables are all non-negative. In order to achieve this the following steps are required:

- If the problem is minimize $c^T x$, convert it to maximize $-c^T x$.
- Let z denote the value of the objective function, i.e. $z = c^T x$. Hence row 0 can be denoted as:
$$z - c^T x = 0.$$
- If a constraint is an inequality of the form
$$\alpha_{i1} x_1 + \alpha_{i2} x_2 + \cdots \alpha_{in} x_n \leq b_i$$
introduce a non-negative **slack variable** s_i to convert it into an equality:
$$\alpha_{i1} x_1 + \alpha_{i2} x_2 + \cdots \alpha_{in} x_n + s_i = b_i.$$
- If a constraint is of the form
$$\alpha_{i1} x_1 + \alpha_{i2} x_2 + \cdots \alpha_{in} x_n \geq b_i$$
convert it in to an equality constraint by subtracting a non-negative **surplus variable** s_i. The resulting constraint becomes:
$$\alpha_{i1} x_1 + \alpha_{i2} x_2 + \cdots \alpha_{in} x_n - s_i = b_i$$

Usually the nature of optimisation problems requires all variables to be positive or at least zero as they usually model some sort of produced material. For completeness sake, however, we also consider variables unrestricted in sign:

- If some variable x_j is unrestricted in sign, replace it everywhere in the formulation with $x'_j - x''_j$ where $x'_j, x''_j \geq 0$ and $x_j = x'_j - x''_j$.

Once a linear optimisation problem has been converted into the standard form we basically have to solve a system of linear equations, which is a standard routine. However, if a solution exists, it has to be ensured that it is optimal with respect to the objective function.
In order to find a feasible solution in the first place all **non-basic variables**, i.e. those occurring in more than one equation, are set to zero. From here a **basic solution** can be obtained by immediately. Now if all coefficients in row 0 are

non-negative the current basic solution is already optimal because z cannot be increased any further. Otherwise pick a variable x_j with a negative coefficient in row 0, which we will call **entering variable**. The choice of this entering variable is arbitrary as long as its sign is negative. The idea is to pivot in order to make a non-basic variable a basic one. This change of basis is done using the famous Gauss-Jordan procedure. Next the so-called **pivot element** is chosen according to the ratio of the right hand side of the equation and the coefficient of the entering variable. The pivot element is chosen as being the one of minimum ratio. Once the pivot element has been identified we perform a Gauss-Jordan pivot. This results in a new basic solution, which is again tested for optimality. This process is repeated until all coefficients in row 0 are positive. Subsequently the current basic solution will be optimal.
We will demonstrate the functionality of the simplex method:

Example 3.9. *Consider the following linear optimisation problem:*

$$\begin{aligned} \textit{maximize } x_1 + x_2 & \\ 2x_1 + x_2 \leq & \ 4 \\ x_1 + 2x_2 \leq & \ 3 \\ x_1 \geq 0, x_2 \geq 0. & \end{aligned}$$

Applying the techniques outlined in the description of the simplex method the problem can be converted into standard form and we obtain a system of linear equations:

$$\begin{aligned} z - x_1 \ -x_2 \qquad\quad &= 0 \qquad (1) \\ 2x_1 \quad +x_2 + x_3 \qquad &= 4 \qquad (2) \\ x_1 \quad +2x_2 \qquad +x_4 &= 3 \qquad (3) \end{aligned}$$

In this system of linear equations x_3 and x_4 are the slack variables while $z = x_1 + x_2$ denotes the value of the objective function. Now the goal is to maximize z while satisfying all equations. Complying to the notation introduced above x_1 and x_2 are the non-basic variables and thus x_3 and x_4 are set to zero to obtain a basic solution. Here this yields:

$$x_1 = x_2 = 0, x_3 = 4, x_4 = 3, z = 0.$$

It is obvious that z can be increased by increasing x_1 or x_2 because both signs are negative. As thus either x_1 or x_2 is applicable as entering variable. We will choose

x_1 and compute the ratios of the right hand side of the equation and the coefficient of the entering variable. We obtain:

$$r_1 = \frac{4}{2} = 2 \textit{ in (2) and } r_2 = \frac{3}{1} = 3 \textit{ in (3).}$$

As $r_1 < r_2$ the choice for the pivot element is row (2) and thus the system of linear equations can be transferred into

$$\begin{aligned} z \quad -\frac{1}{2}x_2 + \frac{1}{2}x_3 \qquad &= 2 && (4)\\ x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 \qquad &= 2 && (5)\\ +\frac{3}{2}x_2 - \frac{1}{2}x_3 + x_4 &= 1 && (6) \end{aligned}$$

with basic solution

$$x_1 = 2, x_2 = x_3 = 0, x_4 = 1, z = 2.$$

As x_2 in line (4) is still of negative sign the current basic solution is not necessarily optimal. Therefore, x_2 becomes the new entering variable and simplex method is performed again obtaining the following system of linear equations:

$$\begin{aligned} z \qquad + \frac{1}{3}x_3 \ \ \frac{1}{3}x_4 &= \frac{7}{3} && (7)\\ x_1 \quad + \frac{2}{3}x_3 - \frac{1}{3}x_4 &= \frac{5}{3} && (8)\\ x_2 - \frac{1}{3}x_3 + \frac{2}{3}x_4 &= \frac{2}{3} && (9) \end{aligned}$$

yielding in the basic solution

$$x_1 = \frac{5}{3}, x_2 = \frac{2}{3}, x_3 = x_4 = 0, z = \frac{7}{3}. \tag{10}$$

This solution is feasible as it fulfils all conditions of the linear optimization problem. It is also optimal because all coefficients in row (7) are positive.

Pseudo-Boolean Optimisation

One of the key contributions of this thesis is the development of a model for the optimal scheduling problem which purely relies on Boolean variables. Hence we concentrate on the the pseudo-Boolean (PB) optimisation problem. Therefore, we define PB constraints as follows:

Definition 3.10. *A **PB-constraint** is an inequality of the form*

$$C_0p_0 + C_1p_1 + C_{n-1}p_{n-i} \leq b_i$$

where, for all $i \in \mathbb{N}$ p_i is a literal and C_i and b_i are integer coefficients.

A `TRUE` literal is interpreted as the value 1, a `FALSE` literal as 0. In particular $\overline{x} = (1 - x)$. As defined in Definition 3.6 a PB optimisation problem may consist of an arbitrary number of PB constraints.

Definition 3.11. *A coefficient C_i is called **activated under a partial assignment** if its corresponding literal p_i is assigned to `TRUE`. A PB-constraint is said to be **satisfied under an assignment** if the sum of its activated coefficients exceeds or is equal to the right-hand side constant C_n.*

In a PB optimisation problem an objective function is a sum of weighted literals on the same form as the left hand side on the constraints. The PB optimisation problem is the task of finding a satisfying assignment to a set of PB-constraints that minimizes a given objective function which in this work will be the total transmission time for all messages of a given problem.
To be more precise assume we have a PB minimization problem with an objective function $f_0(x)$. A minimal satisfying assignment can readily be found by iterative calls to the solver. First run the solver on the set of constraints (without considering the objective function) to get an initial solution $f_0(x_0) = k$.
Then add a new constraint $f_0(x) \leq k$ and restart the procedure. If the problem is UNSAT, k is the optimum solution. If not, the process is repeated with the new smaller solution.
Throughout this work we will distinguish between **feasible** and **optimal** solutions. Feasible solutions refer to all values that satisfy the (PB-)constraints used to formulate the optimisation problem. An optimal solution however is a feasible solution that also fulfils the properties of an optimal vale defined in Definition 3.6. To be more precise: An optimal solution is a solution which is feasible and for which there exists no feasible solution which evaluates to a smaller value of the objective function.
There are several ways how either optimisers for constraint programming - such as CPLEX - or verification tools - i.e. Minisat+ or the SMT solver YICES2 - can be deployed to compute optimal solutions. We will briefly outline the techniques used in this work.
Basically in the scope of this work we can differentiate between three different approaches:

1. The first approach is the use of the state-of-the-art optimisation software package CPLEX briefly introduced in Section 2.5. It uses different methods of integer programming such as the popular Simplex algorithm. Usually CPLEX is satisfied with feasible solutions if no better solutions can be computed within a given interval which is user specified. Yet CPLEX can be forced to take the entire search space into consideration which can have a significant impact on the computation time. CPLEX performs its calculations in a highly parallelised way using as many simultaneous threads as possible.

2. The second approach translates the problem into PB-constraints and uses a SAT-based 0-1 integer linear programming (0-1 ILP) solver which is based on a mixture of heuristics and Conflict-Driven Clause Learning (CDCL), which is an algorithm for solving the SAT problem. It basically uses the iterative calls to the solver as described above in an efficient way.

3. Finally we consider a Mixed Integer Linear Programming (MILP) problem involving integers variables as well as non-integers. The main difference to the approach described in 3.2 is that we can use integer variables rather than only constants. This results on the one hand in a more compact representation and on the other hand can also significantly reduce runtime in comparison to CPLEX. Details will be outlined in the respective Section. Inspired by 3.2 we also suggest an incremental approach. This time we start with a given lower bound and prove that no feasible (and hence no optimal) solution can be found. Then the lower bound is incremented until the solver reports SAT. The solution is then optimal by construction.

More details are provided on SAT and SMT in the respective chapters. We will also show in detail how the scheduling problem can be modelled and solved as an optimisation problem in Chapter 4.

3.3 Complexity

We use complexity theory to evaluate our proposed scheduler and to analyse its scalability. Complexity theory is branch of theoretical computer science which classifies computational problems according to their difficulty. Instinctively a problem is regarded as inherently difficult if its solution requires significant resources regardless of the algorithm used. In the following section we will formalise this intuition by introducing different complexity classes and relating these classes to each other. We will outline why computational problems which seem at first glance simple my still be hard to solve.

As a first example consider the simplex method introduced in the previous Section. This algorithm is reliable and can easily solve problems with hundreds of variables and thousands of constraints on a small desktop computer in a matter of seconds. If the problem is sparse or has some other exploitable structure, we can often solve problems with tens or hundreds of thousands of variables and constraints.
On the other hand even simple problems may be very hard to solve. Consider the so-called **Tautology Problem**. A tautology is a logical expression whose value is true regardless of the values of its propositional variables. There is a straightforward way to test if a given logical expression is a tautology. Construct a truth table with one row for each possible assignment of truth values to the variables of the expression. The expression is a tautology if and only if every row evaluates to `TRUE`. If the expression has k variables, then the table has 2^k rows. Thus a straightforward implementation of this algorithm is in the complexity $\mathcal{O}(2^k)$. This means for 30 variables, there are already more than billion rows. These observations are typical of what happens when one uses an exponential-time algorithm. An important part of this thesis is concerned with the analysis of so-called **scalability** of our proposed scheduling framework.
In order to classify scalability and the subsequent computational costs we want to introduce the concept of **complexity classes**. For notation we stick to [46]. The definition of SAT problems as a special decision problem introduced in Section 3.1 can be generalized as follows:

Definition 3.12. *A **search problem** is specified by an algorithm $\mathcal{C}$ requiring two inputs, an instance $\mathcal{I}$ and a proposed solution $\mathcal{S}$, and runs in time polynomial in $n = \mid \mathcal{I} \mid$, which defines the length of the instance. Furthermore $\mathcal{S}$ is called a **solution to** $\mathcal{I}$ if and only if $\mathcal{C}(\mathcal{I}, \mathcal{S}) =$ `TRUE`.*

This means a search problem has two defining characteristics: On the one hand the correctness of any proposed solution $\mathcal{S}$ can quickly be verified. This is done by applying the polynomial algorithm $\mathcal{C}$ which takes the given instance $\mathcal{I}$ and the proposed solution $\mathcal{S}$ as an input and returns `TRUE` if and only if $\mathcal{S}$ is really a solution to $\mathcal{I}$. On the other hand the runtime of $\mathcal{C}(\mathcal{I}, \mathcal{S})$ is bounded by a polynomial in n, the length of the instance.

Example 3.13. *SAT is a search problem. A given Boolean formula in CNF can be regarded as an instance $\mathcal{I}$. It is our goal to find a solution $\mathcal{S}$, which meets the particular specification to satisfy each clause. Therefore, $\mathcal{S}$ contains an assignment for every variable in $\mathcal{I}$. $\mathcal{S}$ is concise and its length is polynomially bounded by that of $\mathcal{I}$ because it cannot consist of more variables. The polynomial-time algorithm $\mathcal{C}$, which takes $\mathcal{I}$ and $\mathcal{S}$ as an input, just has to check whether the assignment specified by $\mathcal{S}$ indeed satisfies every clause in $\mathcal{I}$.*

Definition 3.14. *The class of all search problems defined in 3.12 is denoted $\mathcal{NP}$.*

There are many other examples of $\mathcal{NP}$ search problems that can be solved in polynomial time such as finding shortest paths or minimum spanning trees of a graph or maximal flows in a network. In all these cases, there is an algorithm that takes as input an instance $\mathcal{I}$ and has a running time polynomial in n. If $\mathcal{I}$ is satisfiable the algorithm returns a solution. Vice versa if no solution exists the algorithm correctly returns `UNSAT`.

Definition 3.15. *The class of all search problems that can be solved in polynomial time is denoted $\mathcal{P}$.*

Clearly $\mathcal{P}$ is a subset of $\mathcal{NP}$, i.e.

$$\mathcal{P} \subset \mathcal{NP} \tag{11}$$

$\mathcal{P}$ is the abbreviation for polynomial, $\mathcal{NP}$ stands for "nondeterministic polynomial time", a term going back to the roots of complexity theory.
Given the importance of the SAT search problem, researchers over the past 50 years have tried hard to implement efficient ways to solve it. Tremendous progress has been made but due to the $\mathcal{NP}$-complete nature of the problem, there are always cases which cannot be solved fast. This means that even the fastest algorithms currently in use may still perform exponentially on their worst-case inputs.
In order to fully classify the SAT problem in a complexity class we have to define the reduction of a search problem:

Definition 3.16. *Given two search problems A and B we can define a **reduction** from a A to B as a polynomial-time algorithm f that transforms any instance $\mathcal{I}$ of A to an instance $f(\mathcal{I})$ of B together with a second polynomial-time algorithm h mapping any solution $\mathcal{S}$ of $f(\mathcal{I})$ to a solution $h(\mathcal{S})$ of $\mathcal{I}$.*

If $f(\mathcal{I})$ has no solution, then neither does $\mathcal{I}$. The two translation functions f and h imply that any algorithm for B can be converted into an algorithm for A. We can now define the class of the hardest search problems:

Definition 3.17. *A search problem is called **$\mathcal{NP}$-complete** if all other search problems reduce to it.*

In fact SAT has been one of the first problems that was proven to be $\mathcal{NP}$-complete [29]. This means that all problems in the complexity class $\mathcal{NP}$, which includes a wide range of natural decision and optimization problems, are at most as difficult to solve as SAT. There is no known algorithm that efficiently solves each SAT problem and it is generally believed that no such algorithm exists. However, this

belief has not been proven mathematically. Resolving the question whether SAT has a polynomial-time algorithm is equivalent to the $\mathcal{P}$ versus $\mathcal{NP}$ problem, which is a famous open problem in the theory of computing and the converse inclusion of equation 11.
Finally we can summarize the complexity classes essential for the scope of this work:

1. $\mathcal{P}$: Solvable in polynomial time.
2. $\mathcal{NP}$: Positive answers can be verified in polynomial time.
3. $\mathcal{NP}$-complete: The hardest or most expressive problems in $\mathcal{NP}$.

Many other complexity classes can be characterized in terms of mathematical logic but are not relevant to the algorithms explored in this thesis. This is due to the fact that the optimal scheduling problem is also $\mathcal{NP}$-complete and therefore all algorithms tackling the SAT problem can be applied.
In order to prove this fact we will briefly describe one of the most famous problems from complexity theory: The **travelling salesman problem** (TSP) asks the following question:

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

It is assumed that the problem was firstly formulated in the 1930s and it is still one of the most intensively studied problems in optimization [47]. In the theory of computational complexity, the decision version of the TSP (where, given a length L, the task is to decide whether the graph has any tour shorter than L) belongs to the class of $\mathcal{NP}$-complete problems. Thus, it is possible that the worst-case running time for any algorithm for the TSP increases superpolynomially (but no more than exponentially) with the number of cities. The job scheduling problem is closely linked to the TSP if the following transformation is considered: the cities can be regarded as jobs which have to be scheduled to an endsystem (salesman). A detailed proof can be obtained from [48].

3.4 SAT Solving

We will conclude this theoretical chapter by introducing the basic principles of modern SAT-solvers which intersect the theory of Boolean Function and Complexity issues. SAT solvers determine whether a given Boolean formula can be satisfied and state a valid solution if one exists. For our purposes we expand SAT-solving techniques to even find optimal solutions i.e. for PB-optimization problems.

The Boolean Satisfiability problem introduced in Section 3.1 is a well-known constraint satisfaction problem with many applications in the fields of VLSI Computer-Aided Design (CAD) and Artificial Intelligence (AI). As outlined in the previous Section it is known to be $\mathcal{NP}$-complete. Still in practice there have been tremendous advancements in the past two decades [49], which make modern SAT-solvers applicable to huge problems consisting of millions of variables and constraints. As we deploy modern SAT solvers throughout this work it is sensible to introduce functionality and impact of these solvers. We will dedicate special emphasis to the circumstances where they can quickly find results for $\mathcal{NP}$-complete problems. A simple brute force exponential approach can successively assign all variables. For n variables this would require 2^n different assignments in the worst case to determine whether a solution exists. However, in practice modern SAT-solvers are much more efficient:

In order to explain how modern SAT-solvers work we will define the terminology of this Section following [50]. Most SAT-solvers use the CNF representation discussed in Section 3.1. Recall that in CNF a Boolean Formula is represented as a conjunction of clauses where each clause itself is a disjunction of literals. A literal is either a Boolean variable or its negation. Note that a formula in CNF is only satisfied if each clause is satisfied i.e. at least one literal in each clause must be assigned to `TRUE`. A clause with at least two unassigned literals is called **unsatisfied**. A **unit clause** is a clause where all literals are `FALSE` except one which is unassigned referred to as the **unit literal**. Finally we denote a clause where all literals are `FALSE` as a **conflicting clause**.

Most modern SAT-solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm illustrated in Algorithm 1. It was introduced in 1962 and is a complete, backtracking-based search algorithm for deciding the satisfiability for solving the SAT problem. The DPLL procedure is shown as *SATsolve* in Algorithm 4 and is based on three main engines: *decision, deduction* and *diagnosis*. The basic SAT algorithm is a branch-and-search algorithm. An unassigned variable is called **free**. Initially all variables are free.

All results computed in this thesis rely on conflict driven clause learning (CDCL) SAT-solvers. Explicitly we deploy MiniSat 1.4 [39] and an integrated SAT solver of Yices SMT which is similar to MiniSat [51]. As the YICES2 Tool paper [52] states the integrated SAT solver "uses the CDCL approach. It is similar in performance and implementation to solvers such as Minisat 1.4 (...) with extensions to communicate with the theory solvers". Theory solvers are in detail explained in Chapter 5.2.

Algorithm 1 DPLL-style SAT solver

```
procedure SATSOLVE(Boolean Problem in CNF)
    if (Deduce()=CONFLICT) then return UNSAT;            ▷ Pre-process
    end if
    while (Decide()=SUCCESS) do                          ▷ branch
        while (Deduce()=CONFLICT) do                     ▷ constraint propagation
            blevel= Diagnose();                          ▷ conflict-driven learning
            if blevel==0 then
                return UNSAT
            else
                backtrack(blevel);
            end if
        end while
    end while
    return SAT
end procedure
```

Therefore, both SAT solvers deployed in this work follow the patterns outlined in Algorithm 1. In both cases pre-processing is done by a deduction engine without making any decision. Besides other techniques this engine applies the so-called **unit clause rule** [53] repeatedly on unit clauses until no such clause exists or a conflicting clause is detected. This rule basically works by identifying unit clauses and setting the unit literal to `TRUE` in order to satisfy the corresponding clause. Consecutive application of this rule is also denoted by Boolean Constraint Propagation (BCP) or unit propagation. Furthermore the deduction engine can also apply the so-called **pure-literal rule**, which identifies variables that appear only as positive (or negative) literals in remaining unsatisfied clauses and assigning them to `TRUE` (or `FALSE` respectively).

The main loop beginning in line 4 starts by invoking the decision engine wherein a free variable is assigned to a value. This process is called branching literal and the choice of this branching literal is crucial to the performance of the SAT solver. After the branch the *deduction* engine is called which applies BCP until no more unit clauses exist or a conflict occurs. In the first case the decision engine is called again to make a decision at the next level, in the latter the *diagnosis* engine is invoked to resolve the conflict. This involves learning reasons for the conflict and backtracking. Abstractly speaking backtracking will undo assignments made under BCP on a previous level on the decision stack derived from conflict analysis. Both MiniSAT and the Yices' integrated SAT-solver rely on a clause database where

so-called **learnt clauses** are stored. This database is created when a constraint becomes conflicting under the current assignment. The conflicting constraint is then asked for a set of variable assignments that make it contradictory. For a clause this would be all the literals of the clause which are `FALSE` under a conflict.
However, as the set of learnt clauses increase, propagation is slowed down. Therefore, learnt clauses are deleted after a certain amount of time. We have also investigated the influence of learnt clauses during our research but the results are out of the scope of this work [54].
In both solvers under investigation the decision phase will continue until either all variables have been assigned, in which case we have a model, or a conflict has occurred. On conflicts, the learning the backtracking level is computed, the procedure will be invoked and a conflict clause produced. Then the process is repeated until either the solver can deduce that no valid assignment exists or all clauses are satisfied. In the first case the solver will return `UNSAT`. In the second case we can again distinguish between two possibilities: If all variables are assigned the solver will precisely return this assignment - called a model - and `SAT`. However, there may also be models where several variables are not assigned because all clauses are already satisfied without explicitly assigning every single variable. For instance the Boolean expression $x_1 \vee x_2$ is satisfied regardless of the assignment of x_2 if $x_1 = \texttt{TRUE}$.
The next chapter is dedicated to precisely describing the setup in which we can apply these SAT-solvers. We will investigate how to formulate constraints as well as objective functions in an appropriate way to express the $\mathcal{NP}$-complete - and thus computationally difficult problem - using Boolean expressions only. We will outline how these state-of-the-art tools can be applied to solve the optimisation problem and discuss limitations arising from scalability issues.

4 An Optimal SAT based Scheduler

4.1 Introduction

This section discusses two major contributions of this thesis: At first we develop a time-discrete model to illustrate the scheduling problem in an arbitrary on-chip network. The endsystems are connected via switches to each other such as in modern MPSoCs. In a second step we will demonstrate how an objective function and subsequent constraints can be formulated in a way respecting the properties of a PB-optimization problem as introduced in Chapter 3.2.
We will then apply MiniSat+, a pseudo-Boolean SAT-solver, in order to compute optimal solutions with respect to minimal transmission times. We will compare our results to the state-of-the-art software CPLEX and evaluate the performance.
We will show that the proposed scheduling framework is very efficient when applied to small and medium sized benchmarks. In comparison to a CPLEX based scheduler we can significantly accelerate runtime. Furthermore the decided model presented reduces the number of constraints fed into the SAT-based solver by identifying obsolete connections on the architecture under consideration. These results have partially been published in [2].

4.2 System Model

Preliminaries

In a time-triggered network all message transmission is triggered by the progression of time. Schedule tables define the points in time of all message transmissions with respect to a global time base. The global time base can be a low frequency global clock signal or the result of clock synchronization, thus also permitting different high-frequency clock domains.
As we have discussed in Chapter 2.3 these schedule tables are either located in network interfaces (e.g., GENESYS MPSoC [15])or in the switches (e.g. Athereal [16]). The point in the time of the injection of a message into the NoC or MPSoC as well as the points in time for the redirection of messages between switches are solely controlled by the schedule tables and independent of the behaviour of the endsystems.
Therefore the timing of the NoC or MPSoC can be analysed in isolation, because the bandwidths, latencies and jitter are determined by the schedule tables. The autonomy of the NoC or MPSoC simplifies worst-case execution time analysis and facilitates temporal composability [6]. Therefore a sensible model must be

developed by considering the allocation of jobs to resources of the network at every point in time.
In this thesis we will model the multicore system under consideration as a bidirectional graph. It is distinguished between nodes n, jobs j and messages m where n, j and m describe the total number respectively. The number of nodes n consists of the number of switches s and the number of endsystems e resembling the NoC or MPSoC under consideration, thus $n = e + s$.
Throughout this work communication will be modelled by messages being sent from one job to another. There are two ways in which communication can be optimized: On the one hand jobs can be allocated to different nodes and on the other hand it has to be determined by the schedule which path a message chooses to travel from sending to receiving jobs. To restrict the degree of freedom we will initially focus on the latter property and assign jobs constantly to nodes during this chapter. This condition will be relaxed from Chapter 5 onwards.
Basically a message can be defined by specifying the sender and recipient. In this context we also we allow logical dependencies of jobs i.e. messages may have to wait for one another.
Throughout this work time is considered to be discrete therefore different points in time - so-called *timeframes* - can be identified and easily distinguished. This guarantees we can easily examine the status of each message and its location during every timeframe. The model thus depends on n, j, m and the number of timeframes, denoted by t, which are needed to complete all jobs. Our goal is to calculate the minimal value for t.
The activities in the schedule tables can be temporally aligned in such a way that collisions are avoided, thereby avoiding any need for dynamic arbitration in the NoC or MPSoC as long as the system under consideration functions faultlessly which is assumed throughout this chapter. Faults are randomly injected into the system in Chapter 8 and the impact of our proposed scheduling framework on fault-tolerant architectures is discussed in detail then.

Restriction to Boolean Variables

In this subsection we will substantiate how the application model described in the previous subsection can be realized in order to apply Boolean SAT solving techniques which will enable us to find an optimal solution to the scheduling problem within an on-chip network. The schedule tables in the network interfaces require temporal coordination of the injection times and paths of messages. Likewise, switch configurations need to coordinate in such a way that the routing decisions prevent collisions.

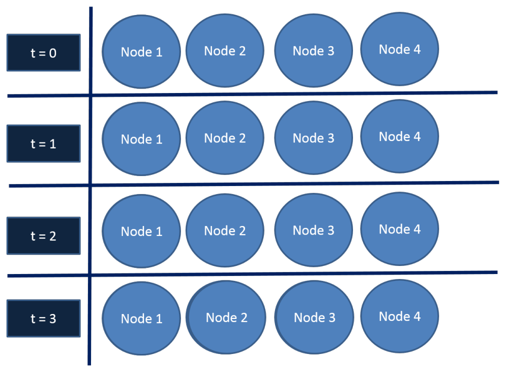
Constant name	Description
n	Number of nodes
e	Number of endsystems
s	Number of switches
j	Number of jobs
m	Number of messages
t	Number of timeframes

Table 5: Overview table with constants

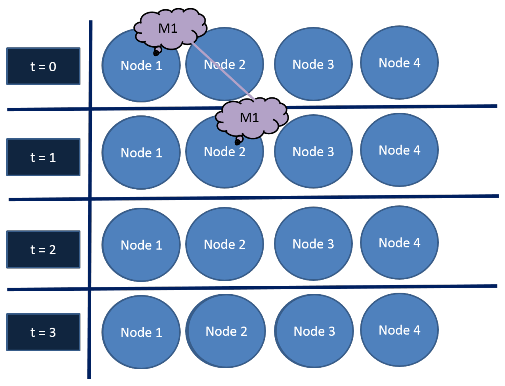
Applying the proposed time discrete model guarantees the network can be specifically analysed. At every point in time it can be deduced which jobs are assigned to an endsystem, which switches are used by messages and which nodes are idle. By construction jobs communicate with each other via messages which are triggered on behalf of the schedule computed. Messages have to respect logical dependencies determined by the problem instance. The logical dependencies can be illustrated using a directed acyclic graph (DAG). Jobs cannot be executed on the same endsystem simultaneously. Therefore an endsystem can be labelled with the respective job number or zero if the endsystem is idle. The traversal of a message thorough the system under consideration can be depicted by stating every node visited on the way from source to destination in the respective timeframe. Considering this the following representation of the scheduling problem can be obtained:

$$\begin{array}{cccc}
n_1^0 & n_2^0 & n_3^0 & n_4^0 \cdots \\
n_1^1 & n_2^1 & n_3^1 & n_4^1 \cdots \\
\vdots & \vdots & \vdots & \vdots \\
n_1^t & n_2^t & n_3^t & n_4^t . \cdots
\end{array}$$

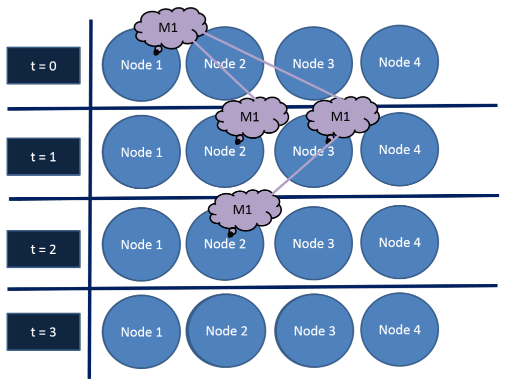
where n_i^s describes the messages currently assigned to n_i during timeframe s.
Let there be a link between two nodes n_1 and n_2. For demonstration purposes we do not distinguish between switches and endsystems. Then executing for instance a job j_1 during timeframe t_0 may result in a message being sent from node 1 which arrives at node 2 in the consecutive timeframe $t + 1$.
A path travelled by a message through the network can therefore be described by formulating constraints describing how the behaviour of a certain node n_i in timeframe t influences the behaviour of another node n_j in the subsequent



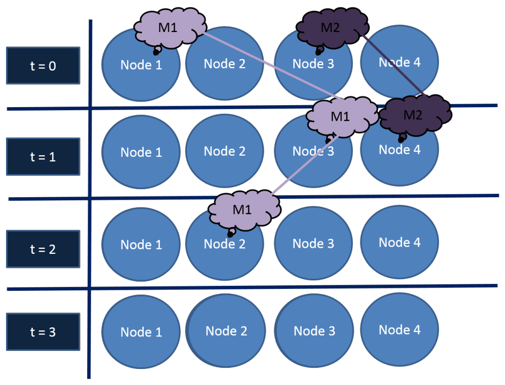
(a) Idle nodes 1 to 4 in consecutive timeframes.



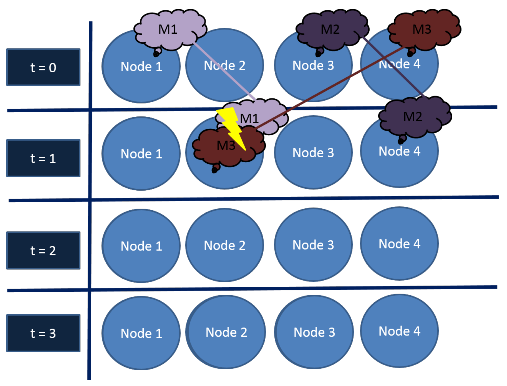
(b) Message M1 traveling from node 1 to node 2 in two consecutive timeframes.



(c) Different possibilities for message M1 to travel from node 1 to node 2.



(d) Messages M1 and M2 traveling through the system simultaneously without collision.



(e) Collisions of two messages on the same node in the same timeframe must be avoided.

Figure 3: Schematic representation of the proposed model.

timeframe $t+1$. A detailed schematic representation is illustrated in Figure 3 which also considers different scenarios:

- In Figure 3a all nodes (i.e. switches and endsystems) are idle in all considered timeframes.
- In Figure 3b a message is sent from node 1 to node 2 via a link connecting both nodes.
- Likewise Figure 3c outlines two different possibilities for a message travelling between two links. On the one hand the messages could possibly be delivered directly from node 1 to node 2. On the other hand there also exists an alternative route via node 3. Which route is favourable is determined by the proposed scheduling framework. It optimises the total makespan and guarantees to find the minimal number of timeframes required to complete all jobs by sending all messages.
- Figure 3d depicts how messages can traverse the multicore system under consideration simultaneously. Collisions on every node (i.e. switch or endsystem) must be avoided at all times.
- The proposed scheduling framework identifies collisions as illustrated in Figure 3e and only returns conflict-free schedules without collisions.

Translation into Boolean expressions

In order to apply the theory of PB-optimization outlined in Chapter 3.2 to the model we have to translate the constraints into PB variables. Hence it is not sufficient to assign a job j_i to a node n_j by simply labelling n_j with the j_i. Thus in order to introduce Boolean variables we now assign j_i performed by n_j using *one-hot encoded* variables. This means every node has to be described by m variables in every timeframe, where m denotes the total number of messages that will be triggered in the network. Given two jobs j_1 and j_2 that send messages m_1 and m_2 and a path between two nodes n_1 and n_2 we require a set of 4 Boolean variables in each timeframe to describe the different possibilities of allocating the jobs as illustrated in Table 6.
For every message and every node we use a Boolean variable which is set to 1 if the corresponding message is currently allocated to the node. Otherwise the Boolean variable is set to zero. Currently the model is subject to the following limitations:

Timeframe 1	Node 1 x_1, x_2	Node 2 x_3, x_4
Timeframe 2	Node 1 x_5, x_6	Node 2 x_7, x_8
$\vdots$	$\vdots$	$\vdots$
Timeframe t	Node 1 x_{4t-3}, x_{4t-2}	Node 2 x_{4t-1}, x_{4t}

Table 6: Scheduling two messages in t timeframes using Boolean variables only

1. A certain node n_i can only perform one job j_i per timeframe implying that only one message can be sent at maximum by each node during this specific point in time.

2. In order to be able to use our representation communication itself is modelled by a message being allocated to different nodes in consecutive timeframes. Therefore messages can not be buffered and have to travel through the network immediately.

3. The allocation of jobs to nodes is a constant and the jobs are distributed before the optimal solution is calculated.

Taking this into account we can compute the minimum travel time of all messages in the system by maximizing the number of timeframes in which all nodes are idle. Stemming from the ILP community, PB problems often contain an objective function, a linear term that should be minimized or maximized under the given constraints [55]. Adding an objective function is also an extension to standard SAT, where there is no ranking between different satisfiable assignments. A detailed explanation for the formulation of applicable constraints and the objective function will be given in Section 4.3.

4.3 MiniSat+

In this section we will provide details how to apply constraints and formulate an objective function using only Boolean variables. We follow the structure developed in Section 4.2 and explain the transformation from MILP into Boolean formulae because MiniSat+ requires the problem to be formulated in PB format, a mix of inequality constraints over $0 - 1$ variables. As we have seen in Chapter 3.2 an instance of a linear pseudo Boolean optimization problem can be formally defined as follows:

$$\textit{minimize} \qquad \sum_{j=1}^{n} c_j x_j \tag{12}$$

$$\textit{subject to} \qquad \sum_{j=1}^{n} a_{ij} l_j \geq b_j \tag{13}$$

$$x_j \in \{0,1\}, a_{ij}, b_j \in \mathbb{Z}, i \in \{1, \cdots m\}. \tag{14}$$

where c_j is a non-negative integer cost associated with variable $x_j; 1 \leq j \leq n$ and a_{ij} denote the coefficients of the literals l_j in the set of m linear constraints.

Constants

As illustrated in Table 9 the only numerical constants needed are the number of nodes n, the number of jobs j with the corresponding number of messages m and the of timeframes t. As it is our key objective to minimise t we need an initial value t_0 to start our analyse from. As described previously in Chapter 3.2 our aim is to find a solution to the PB-problem in t_0 timeframes and then feeding new constraints into the PB-solver reducing the number of timeframes required.
While n, m and j can be obtained from the physical model and the randomly generated input file the initial value t_0 can easily be calculated by counting the number of timeframes needed if all jobs were performed successively. This always guarantees a feasible but obviously not necessarily an optimal solution to the scheduling problem. Furthermore the allocation of jobs is fixed and cannot be altered to optimise the result even further. Finally dependencies of jobs are a constant obtained from the logical mode.

Decision Variables

In the following the variables needed in the optimisation process are introduced. Opposed to the constants discussed above decision variables can be changed during the optimisation process unless constraints are violated.

Variables to assign messages to nodes

Every node can be described by $j = m$ variables in every timeframe, where $m_i \in \{1, \cdots, m\}$ is set to 1, if job m_i is performed during the considered timeframe.

Using the formulation from Section 5.3 every n_i^s now consists of m different Boolean variables $x_1, \cdots x_m, x_k \in \{0,1\}$ for $1 \leq k \leq m$. As a neat side effect the equation

$$\sum_{k=1}^{k=m} x_k = 1 \tag{15}$$

guarantees that only one job can be performed on each node during each timeframe which complies with the regulations to the model described above. As all equations arising from (15) have to be fulfilled $m \cdot n \cdot t$ constraints are created and added to the model. In this context it is important to mention that a number of these constraints may be obsolete as they can describe executions of jobs that may never occur in practice.

Possible Paths

For each message m_i a path P_i can be fully described by adding all nodes visited during the transmission process. P_i begins with the number of the endsystem the sending job is located upon followed by the ids of the switches visited and finishes eith the id of the endsystem the receiving job is located upon.
Using our representation from above we describe a possible path using logical implications. Formally this means if the event A describes the sending of a message m_1 at node n_1 and B describes the following reception at node n_2 assuming there is a path between n_1 and n_2. Then applying formal logic this can be expressed as: $A \longrightarrow B$ or using propositional calculus as $\overline{A} \vee B$ which is obviously a Boolean expression. If we consider the one-hot encoded variables to be x_A and x_B the pseudo Boolean constraint is:

$$x_B - x_A \geq 0. \tag{16}$$

Scheduling Constraints

This part describes the constraints that are used in scheduling time-triggered and event-triggered messages.

Connectivity Constraints

These constraints use the connectivity constants C in order to build the network path topology. A message can visit the link between two nodes, a and b, only if there is a direct connection between these two nodes. In the SAT model this condition is ensured by the use of implications i.e. a message m being sent from node 1 in timeframe t can only result in the same message being received by node 2 in $t+1$ if there is a direct link between the nodes. This is however only an implication and not an equivalence as node 2 may remain idle for instance if m takes a different route. This can be formulated as a constraint

$$x_2^t - x_1^{t+1} \geq 0.$$

Collision-Free Constraint

Collision free message transfer is guaranteed on the one hand by the constraints formulated in (15) and on the other hand by ensuring that no two messages travel the same link in the same timeframe by limiting the maximum arrival of messages per node per timeframe to one.

Job Dependencies Constraints

Certain jobs cannot be triggered at any point in time but have to wait for the arrival of a certain message before those jobs can actually send messages themselves. This means constraints have to guarantee that only after the arrival of message m_1 in the fixed timeframe t_f the message m_2 can be triggered in t_f+1. In order to ensure this all possible variables $x_{m_1}^t$ describing the arrival of m_1 from $t=0$ until $t=t_1$ are added and then compared to possible starting points of m_2:

$$\sum_{t=0}^{t=t_f} x_{m_1}^t - x_{m_2}^{t+1} > 0. \tag{17}$$

(17) is only true for $x_{m_2}^{t+1} = 0$ if at least one of the other variables is true. This implies that m_2 will not be sent without the arrival of m_1 during a previous timeframe.

Objective Function

The proposed optimisation process guarantees an optimal solution with respect to time needed for all jobs to be performed successfully. Technically this means that all job corresponding messages are sent and received in the least number of timeframes possible. To realize this auxiliary Boolean variables a_1 to a_t are created for each timeframe from 1 to t. Whenever all nodes are idle during a certain timeframe $\widehat{t}$ the respective auxiliary variable $a_{\widehat{t}}$ is set to zero. If on the other hand a least one job is performed on any node during $\widehat{t}$, the auxiliary variable $a_{\widehat{t}}$ is set to one. Therefore the objective function can be formulated as:

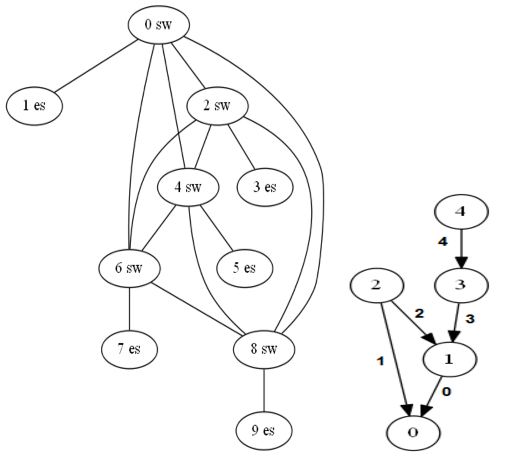
$$\textit{minimize} \sum_{k=1}^{t} a_k. \tag{18}$$

Application of MiniSat+

We propose the application of MiniSat+ [55] as a backend solver to compute optimal solutions for the scheduling problem formulated in the previous subsection. MiniSat+ has been developed from 2006 onwards as a SAT-based alternative for PB optimization problems. As thus it combines the theoretical properties outlined in Section 3.2 with the practical approach of a SAT solver discussed in Section 3.4. MiniSat+ accepts input files exactly in the form outlined and handles the constraints through translation to SAT without modifying the SAT procedure itself. As thus the solver can take advantage of the significant improvements made over the last decades as referred to in Section 2.6 dedicated to related work and in more detail outlined in Section 3.4. The constraints are incrementally solved and the solution is evaluated with respect to the objective function formulated in (18).
MiniSat+ is open source and executable on machines providing limited computational and memory resources. Therefore, it is also suitable for execution on embedded systems which are usually subject to restricted performance.

4.4 Results

The comparison between SAT and CPLEX models is based on 9 example scenarios that are fed equally into both model. These scenarios are generated using Stanford Network Analyzer Platform library (SNAP) [56] which is widely used in numerous academic researches. Each scenario comprises of the constants explained in detail in Table 9 and can be depicted by a physical and a logical models. The physical model consists of a bidirectional graph where the nodes resemble the endsystems and switches and the edges show the connections between them. The logical model is illustrated by a DAG where the nodes represent the jobs and the directed edges specify the dependencies.
As an example consider an on-chip network with 10 nodes consisting of 5 switches and 5 endsystems in which 5 jobs communicate and need to send 5 time-triggered messages. The physical and logical models are depicted in Figure 7, where Figure 7a depicts the physical connection among nodes with bi-directional links



(a) Physical Model (b) TT Logical Model

Figure 4: Model example with 10 Nodes, 5 Jobs and 5 messages.

and Figures 7b depicts the time-triggered logical model. The nodes represent the jobs and the arrows represent the messages sent from one job to another job.

Table 9 shows the CPLEX input constants for the model in Figure 7 according to Section 4.2. The first constant describes the network model $(10, 5, 5)$ in which the model consists of 10 nodes and 5 jobs sending 5 messages. The second constant defines the node connectivity in the network C with bi-directional links. For example the first node is connected to node $1, 2, 4, 6$ and 8 and the last node is connected to node 8. Constant 3 shows the jobs that send the messages, where each job can send more than one message. For example job 1 is the sender of the first message and job 2 is the sender of the messages 1 and 2.

The fourth constant describes the receiving jobs of messages. For example job 0 is the receiver of messages 0 and 1, whereas job 1 is the receiver of messages 2 and 3. The hop-to-hop transmission time T is defined in constant 6. For simplicity and better understanding of the example all times of T are set to a constant with the value of 3. Constant 7 describes the job execution times E and also here we set them all to a constant value of 3.

The constant $ALLOC$ defines the allocation of jobs to endsystems where jobs cannot be allocated to switches (i.e., 0, 2, 4, 6 and 8). Also only one job can be assigned to an endsystem (i.e., nodes 1,3,5,7 and 9).

On the SAT side there is no further distinction between switches and endsystems, this is why we will refer to both as nodes in the following. In the SAT model the number of nodes (10), the number of jobs (5) and the number of maximum

Constant-No.	Constant Name	Data
1	n, j, m	[10,5,5]
2	C	[[0,1,1,0,1,0,1,0,1,0], [1,0,0,0,0,0,0,0,0,0], [1,0,0,1,1,0,1,0,1,0], [0,0,1,0,0,0,0,0,0,0], [1,0,1,0,0,1,1,0,1,0], [0,0,0,0,1,0,0,0,0,0], [1,0,1,0,1,0,0,1,1,0], [0,0,0,0,0,0,1,0,0,0], [1,0,1,0,1,0,1,0,0,1], [0,0,0,0,0,0,0,0,1,0]]
3	S	[1,2,2,3,4]
4	D	[[1,0,0,0,0], [1,0,0,0,0], [0,1,0,0,0], [0,1,0,0,0], [0,0,0,1,0]]
5	T	30
6	U	3
7	E	3
8	ALLOC	[1,3,7,5,9]

Table 7: CPLEX Input constants for model in Figure 7.

timeframes (20) result in a total introduction of $10 \cdot 5 \cdot 20 = 1000$ variables used to describe the temporal SAT model described in Section 4.2.

As outlined constraints to each node in each timeframe are formulated to guarantee sending and reception of all messages triggered by the jobs performed on the nodes. This is done with respect to the randomly generated allocation of jobs - i.e. job 0 is allocated to node 1, which means that during one of the timeframes t_0 message 1 has to be triggered here and will be received by node 1 in $t_0 + 1$. Simultaneously it has to be ensured that no other message uses the link between node 0 and node 1 and that all dependencies from the logical model are respected.

While implementing these constraints it is already considered that some possible message transfers only have to be considered in theory but never take place in practice. For instance message 4 will never travel between nodes 1 and 2 and as thus there is no necessity to formulate any constraints consisting of variables describing the correlation between the link and message 4. Actually the entire implementation of variables representing message 4 on node 1 and 2 is redundant but is still implemented due to a clear representation of results explained in more detail at the end of this section.

Runtimes for the calculations were obtained with CPLEX 12.6.1 and MiniSat+ 1.4 running on a 12 processor Intel(R) Xeon(R), 2.2 GHz server with the operating system Linux Ubuntu 14.04.1. Table 8 depicts the execution times along with the number of constraints for the different physical and logical topologies.

The experiments show on the one hand that MiniSat+ is able to reproduce the results from CPLEX in terms of finding optimal solutions with respect to time and furthermore can significantly decrease runtime of the solver. The result file, however, needs further attention before the paths of all messages can be concluded. This is due to the fact that MiniSat+ calculates the assignment of every Boolean variable introduced when formulating the problem. From this one can immediately deduct the minimal number of timeframes needed to execute all jobs from which the total time taken is calculated by simply counting the auxiliary variables assigned to 1. In order to find all the paths travelled in the network the non-auxiliary variables assigned to 1 have to be linked to the node, job and timeframe they describe. This is also possible but not done within the execution time shown in table 8. Comparing the paths is however necessary in order to review the SAT model with respect to the CPLEX solution. By evaluating the paths for each message we can definitely demonstrate that both approaches produce the same results and are therefore equivalent under the agreed circumstances.

In general one can also witness that applying further additional conditions, for instance in terms of logical dependencies, do not affect runtime considerably. This is due to the fact that logical constraints have a strong impact on the search space and eliminate a large number of possible solutions i.e. exclude the execution of simultaneous jobs. This effect can be exploited, when generating the MiniSat+ problem sheet by not allowing certain jobs to be carried out during specific time-frames. Further pre-processing the input of the SAT solver results in a significant reduction of constraints that have to be passed into the solver. This results in a further speed up when computing optimal schedules.

4.5 Summary

In this subsection we will summarize the results of comparing a SAT-based scheduler to a state-of-the-art scheduler based on CPLEX and outline how the proposed model will be enhanced in the following chapters.
This chapter has introduced a novel approach to solve the scheduling problem. The application of modern SAT solving techniques has been successfully implemented and the corresponding mapping of the problem into a PB optimization has been proposed. We have demonstrated we can not only reproduce the results based on a state-of-the-art MILP scheduler based on CPLEX but we are also able to significantly reduce runtime on the examples analysed. Furthermore we have been able to reduce computational expenditure by downsizing the numbers of constraints required to model the scheduling problem. The results give reason to further investigate the strategy to apply tools from formal verification to the scheduling problem. Our goal is the development of a framework to solve scheduling problems on MPSoC architectures at runtime.
Therefore, it will prove sensible to further adapt our communication model in two ways: On the one hand we have so far restricted our research to small sized problems only masking complexity and scalability. Yet we can already detect that due to one-hot encoding of variables the SAT-based approach will face severe problems with respect to scalability. On the other hand we have so far only confined our focus on a static allocation of jobs which considerably affects the flexibility of the model and makes it less realistic.
Due to these considerations we will dedicate the following chapter to discuss a more flexible approach deploying a state-of-the-art SMT solver allowing a more compact representation of the scheduling problem because the application of the integrated MILP-solver allows the use of integer variables which makes one-hot-encoding strategies obsolete. Furthermore we are able to arbitrarily allocate jobs to endsystems using integer variables. This increases the degree of freedom and the possibilities to optimize communication because jobs can now be allocated dynamically optimizing the number of switches needed for communication between them.

Scenario	Scenario Parameters				CPLEX		MiniSat+	
	Physical Model		Logical Model					
No.	RS	CS	Job	Msgs	runtime (s)	constraints	runtime (s)	constraints
1	3	7	5	5	0.44	1221	0.022	255
2	4	8	3	3	0.92	1387	0.032	606
3	4	8	4	4	0.46	1367	0.02	771
4	4	8	5	5	1.80	2391	0.08	1003
5	5	5	5	5	0.24	1073	0.089	255
6	5	5	4	4	0.17	840	0.064	634
7	5	9	3	3	1.69	1891	0.048	706
8	5	8	3	3	0.42	1210	0.053	613
9	6	6	3	3	0.26	922	0.07	403

Table 8: Comparing the results for 9 example scenarios

5 An Optimal SMT-based Scheduler

5.1 Introduction

This chapter extends our prior work on the application of SAT-solvers to scheduling problems. It describes the second contribution of this thesis: Application of SMT solvers to compute optimal solutions allowing for flexible allocation of jobs to nodes. We will furthermore outline how the proposed scheduling framework can be ported to an MPSoC-emulating system. We will demonstrate how to enable an arbitrary allocation of jobs to nodes as well as multiple messages between them. The proposed optimal scheduler in this section is based on an SMT problem and deploys the state-of-the-art SMT-solver YICES2 [57] in its current version 2.3 [57]. Experimental results on randomly generated scenarios show how runtime and scalability compare to a scheduler based on CPLEX. Furthermore it is simulated how the SMT-approach can be parallelised and how distribution results in a significant reduction of runtime even if the set-up is ported to a resource-constrained embedded platform with limited memory and processing capacities.
The main contributions of the this chapter are as follows:

- **Scheduling model optimized to SMT** with high number of Boolean decision variables and effective clause learning.
- **Parallelisation of incremental scheduling algorithm** by simultaneously using different makespan values at different endsystems.
- **Low memory footprint** enabling execution of scheduling algorithm on target system as required for adaptive time-triggered networks.
- **Experimental evaluation** compared the SMT-based scheduler with a reference scheduler (i.e., MILP) in order to demonstrate correctness as well as the benefits with respect to runtime and memory requirements.

The technical framework and the results of this chapter have partially been published in [3]. It extends our prior work on the application of SAT-solvers to scheduling problems, which was in detail outlined in Chapter 4.
The chapter is structured as follows: We will begin with an introduction to basic SMT theory and outline the advantages of the proposed backend-solver Yices2. Then we will explain in detail how the reference model has to be altered in order to apply SMT. After comparing the novelties and benefits of the proposed SMT-based communication model to other models we will be provide experimental results on randomly generated benchmarks. We will conclude by reporting on

the performance of our proposed scheduling framework on an MPSoC emulating architecture.

5.2 SMT Solving

An Introduction into SMT solving

Recent breakthroughs in SAT solving as outlined in more detail in Section 3.4 have enabled new approaches to software and hardware verification. Furthermore existing SAT solvers can handle problems with millions of clauses and variables that are encountered in bounded model checking, test-case generation, and certain types of planning problems. SAT solvers can especially be applied to optimal scheduling problems as described in the previous chapter. After SAT solving has become a major tool in automated analysis of hardware and other finite systems it has been subject to various enhancements. One of the evolutions of Boolean SAT is Satisfiability modulo theories (SMT), which generalises SAT by adding equality reasoning, arithmetic and other useful first-order theories to the solver.
An SMT-solver is a tool for deciding the satisfiability (or dually the validity) of formulae in these theories. They have numerous applications in theorem proving and other domains such as temporal or metric planning and test-case generation and as we will outline in the following (real-time) scheduling. Just like a SAT solver an SMT-solver proves the satisfiability of a given input formula. Yet the input files may provide an enhanced level of complexity and are not restricted to Boolean expressions: For instance propositionally complex formulae in theories such as arithmetic and uninterpreted functions with equality can be handled. The application of SMT-solvers allows us to check for satisfiability of formulae modulo first-order theories. Using theory combination techniques, we can verify systems using many-sorted first order logic where our terms are not constrained to a single domain.
SMT-solvers have emerged in the last decade beginning in 2005 when the SMT competition was held for the first time as a satellite event of the 17th International Conference on Computer Aided Verification (CAV), which is commonly regarded as the premier international conference in this prominent research area. Since then this contention has become a regular annual event and has attracted attention from prominent protagonists in industry i.e. Microsoft Research [58] and academia i.e. Stanford Research Institute [51]. Hence SMT-solvers were firstly classified in Armin Biere's *Handbook of Satisfiability* in 2009 in [59].
Nowadays SMT-solvers are generally based on a cooperation of an integrated SAT solver and a theory reasoner for the combination of theories understood by

the SMT-solver. The propositional structure of the problem is handled by the SAT solver, whereas the theory reasoner only has to deal with conjunctions of literals. It has been shown that modern SMT-solvers perform particularly well on $\mathcal{NP}$-complete problems [60].

Just like previously in Section 3.4 refinements can be done by refuting models of the propositional abstraction one at a time. This is also described in detail in [60], where it stated that "(however,) in practice it is much more productive to refute all propositional models that are spurious for the same reason. A model of the abstraction is spurious if the set of concrete literals corresponding to the abstracted literals satisfied by this model is unsatisfiable modulo the theory. Given such an unsatisfiable set of concrete literals, the disjunction of the negations of any unsatisfiable subset, also denoted an unsatisfiable core in literature, is a suitable conflict clause. By backtracking and asserting the conflict clause, the SAT-solver is prevented from generating the spurious model again". This means that the smaller the clause, the stronger it is and the more spurious models it prevents. Therefore, an optimal conflict clause, corresponding to a minimal unsatisfiable subset of literals is desirable. We will outline this concept in more detail in Example 5.1.

In recent years several SMT-solvers have been developed on the basis of the concepts outlined above. SMT-solvers apply different theories satisfying different demands. Among those Microsoft Research's contribution *Z3* has emerged to become a state-of-the-art tool [58]. Furthermore the tool developed by Stanford Research Institute named YICES or YICES2 in its current version respectively, deserves special attention. The latter one was introduced in 2006 featuring a novel simplex based algorithm integrated with the classic DPLL procedure. This provided a theory for linear and real arithmetic that established YICES as the top SMT-solver. Two years later, in 2008, the first paper on Microsoft's SMT-solver, Z3, was published. Today both solvers are regarded as the leading tools in SMT solving and only recently in January 2016 both Z3 and YICES2 were classified as "state-of-the-art" SMT-solvers by "Lecture Notes in Computer Science Theoretical Computer Science and General Issues" [61].

If the SMT competition is an indicator of how a solver performs with respect to its contenders, then Z3 is would be a good choice. From 2010 onwards Z3 dominated the competition every year even outperforming its competitors in 2012 without actually submitting a new version. However, since then the actual competition was diversified as a tribute to the various theories and applications of SMT-solvers.

Therefore, for example in the latest SMT-COMP in 2016 three tracks were offered: the conventional main track, an application (i.e., incremental) track, and an unsat-core track. Within each track there are multiple divisions, where each division uses benchmarks from a specific logic or group of logics [62]. Both solvers

mentioned explicitly have performed well when using a combination of Boolean and integer variables which in SMT-Lib terms is called applying the quantum-free, uninterpreted functions with equalities logic (QF-UF) and linear arithmetic (LA). These two theories exactly combine the necessary formulae for our approach described in more detail throughout the next Section. In this context SMT-Lib describes a standard input and theory format to compare different solvers which has been established in its latest version in 2010 due to overwhelming success of SMT solving [63]. Furthermore YICES2 also outperforms Z3 in the so-called incremental track, which means that constraints are incrementally added to a given problem. We will also deploy this technique in the following.
Currently YICES2 may be viewed as a more domain specific SMT solver in contrast to the feature rich Z3 [64]. One of these features is the property to evaluate a model, if one exists, with respect to an objective function. Thus Z3 allows the computation of optimal solutions without any alterations to its source code. However, Z3 is not open source and does not provide the source code for modification. Therefore, we have refrained from using Z3. Furthermore one of the key contributions of this work is an evaluation of the proposed scheduler on the target system itself. As we have opted to emulate an explicit MPSoC architecture as a multicore Raspberry Pi based platform for our purpose it is necessary for the SMT-solver integrated into our framework to provide ARM support. Furthermore beginning in 2010 there has already been research conducted on Time Triggered Ethernet Scheduling as an SMT problem [25]. In this context Winfried Steiner proposed to deploy YICES2 as a backend solver [34]. We will compare our approach to Steiner's latest work which originated in 2015 [35] in Section 5.4 where we will discuss the benefits and novelties of our proposed scheduling framework.
Due to the advantages outlined above we have chosen YICES2 (in the version released on December 11, 2015, YICES2.4.2) to be included in our scheduling framework. We will outline its functionality in the following section.

Properties and Functionality of YICES2

YICES2 supports the same logics as Z3, except for those involving non-linear arithmetic which we will not consider in the scope of this work. It is designed to be modular and extensible, to be efficient on a large class of problems which is exactly the type induced by the optimal scheduling problem. YICES2 includes a Boolean satisfiability solver and theory solvers for four main theories: uninterpreted functions with equalities, linear arithmetic, bitvectors, and arrays. These solvers can be combined as illustrated in Figure 5. The SAT-solver uses the CDCL approach outlined in Chapter 3.4. It is similar in performance and implementation

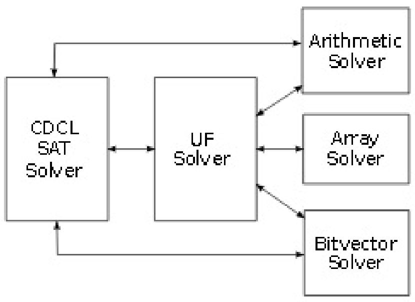


Figure 5: The modular structure of YICES2. The Arithmetic Solver deals with linear integer and real arithmetic. It implements a decision procedure based on the Simplex algorithm. The Bitvector Solver deals with the theory of bitvectors. The Array Solver implements a decision procedure for McCarthy's theory of arrays. Illustration taken from [57].

to solvers such as MiniSat which is the backend of the solver suggested for the PB-optimization problem in the previous section.
YICES2 can be conceptually decomposed into three main modules as depicted in Figure 6:

- *Term Database*
  YICES2 maintains a global term database in which all terms and types are stored. YICES2 provides an API for constructing terms, formulae and types stored in this database.

- *Context-Management*
  A context is a central data structure that stores asserted formulae. Each context contains a set of assertions to be checked for satisfiability. The context-management API supports operations for creating and initializing contexts, for asserting formulae into a context, and for checking the satisfiability of the asserted formulae. Optionally, a context can support operations for retracting assertions using a push/pop mechanism. Several contexts can be constructed and manipulated independently. Contexts are highly customisable. Each context can be configured to support a specific theory, and to use a specific solver or combination of solvers.

- *Model Management*
  If the set of formulae asserted in a context is satisfiable, then one can construct a model of the formulae. The model maps symbols of the formulae to concrete values.

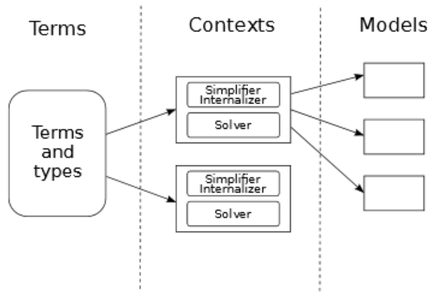


Figure 6: Illustration of the modular structure of the proposed SMT-solver YICES2, taken from [57].

The *simplifier/internaliser* component converts the format used by the term database into the internal format used by the solver. In particular, the internaliser rewrites all formulae in conjunctive normal form, which is used by the internal SAT-solver. For example, a solver for pure arithmetic can be built by directly attaching the arithmetic solver to the CDCL SAT-solver. Similarly, YICES2 can be specialised for pure bitvector problems, or for problems combining uninterpreted functions, arrays and bitvectors.
Thus as YICES2 offers the possibility to select different solvers or combinations depending on the problem we will be attaching the arithmetic solver directly to the SAT-solver. The Arithmetic Solver deals with linear integer and real arithmetic. It implements a decision procedure based on the Simplex algorithm in the version introduced in Section 3.9. This is one of the main advantages of the modular structure of YICES2. Therefore, arithmetic can be built by directly attaching the arithmetic solver to the CDCL SAT-solver.
We will outline the functionality of YICES2 in the following example, which was inspired by [51] but has been slightly altered to comply to the simplex method illustrated in Example 3.9.

Example 5.1. *Let ϕ be a quantifier-free formula consisting of the following constraints:*

- $c_1 : (2x_1 + x_2 \leq 4) \vee P_1$
- $c_2 : \overline{(x_1 + 2x_2 > 3)} \vee P_2$
- $c_3 : (x_1 \geq 0) \vee \overline{P_3}$
- $c_4 : (x_2 \geq o) \vee P_4$

where x_1, x_2, x_3, x_4 are of rational and P_1, P_2, P_3 and P_4 are Boolean. Assuming all constraints have to be fulfilled a CNF-like representation would be:

$$\phi = c_1 \wedge c_2 \wedge c_3 \wedge c_4.$$

As Example 5.1 is a generalization of Example 3.9 we know that ϕ is satisfied, for the basic solution:

$$x_1 = 0, x_2 = 0,$$

as well as the optimal solution with respect to the objective function considered:

$$x_1 = \frac{5}{3}, x_2 = \frac{2}{3},$$

and arbitrary Boolean values for P_1, P_2, P_3 and P_4.
Now YICES2 initially translates the problem into purely Boolean by introducing auxiliary functions as follows:
Let ϕ_1^{Bool} consist of the following constraints:

- $c_1 : A_1 \vee P_1$
- $c_2 : \overline{A_2} \vee P_2$
- $c_3 : A_3 \vee \overline{P_3}$
- $c_4 : A_4 \vee P_4$

with $A_1 = \texttt{TRUE}$ if and only if $2x_1 + x_2 \leq 4$, $A_2 = \texttt{TRUE}$ if and only if $x_1 + 2x_2 > 3$, $A_3 = \texttt{TRUE}$ if and only if $x_1 \geq 0$ and $A_4 = \texttt{TRUE}$ if and only if $x_2 \geq 0$.
YICES2 then calls the integrated SAT-solver (based on MiniSat) to check whether ϕ_1^{Bool} is satisfiable. In this small example in CNF it can easily find a valid solution and produce the following model:

$$P_1 = P_2 = P_4 = \texttt{TRUE} \text{ and } P_3 = \texttt{FALSE}. \tag{19}$$

Because of the P-variables the inequalities are obsolete. However, we can force the SMT-solver to consider the equations by adding further constraints: Let ϕ_2^{Bool} consist of the following constraints:

- $c_1 : A_1 \vee P_1$
- $c_2 : \overline{A_2} \vee P_2$
- $c_3 : A_3 \vee \overline{P_3}$

- $c_4 : A_4 \vee P_4$
- $c_5 : P_1 = \mathtt{FALSE}$
- $c_6 : P_2 = \mathtt{FALSE}$
- $c_5 : P_3 = \mathtt{TRUE}$
- $c_6 : P_4 = \mathtt{FALSE}$

Under these modifications the model from 19 is altered into

$$P_1 = P_2 = P_4 = \mathtt{FALSE} \textit{ and } P_3 = \mathtt{TRUE}, x_1 = x_2 = 0. \tag{20}$$

In the following section we will give details how to model the objective function explicitly using an incremental approach similar to the one introduced in Chapter 3.2.

5.3 Refinements and Implementation

In this section details how to formulate the scheduling constraints to be evaluated in YICES2 are provided. As we have discussed in detail during the precious section YICES2 is a widely recognized SMT-solver that decides the satisfiability of formulae containing uninterpreted function symbols with equality, linear real and integer arithmetic, bitvectors, scalar types, and tuples. We will limit ourselves in the following to two data types to guarantee that computations can be carried out in a reasonable time. Hence the proposed model is restricted to two data types: integer and Boolean. On the one hand this extends the SAT-based model proposed in the previous chapter. Furthermore the proposed approach allows a more compact representation of the scheduling problem accounting. Also the dynamic allocation of jobs to nodes and multiple messages between two endsystems are now modelled resulting in a more realistic representation. On the other hand the proposed SMT-based model does not use complex data structures such as arrays or tuples which are a key element in the model presented in [34]. Furthermore no quantifiers are required for the proposed model which significantly reduces the search space for possible solutions. All integer variables are bounded and the range is explicitly assigned. Further details on those boundaries and the difference between the models are provided in Section 5.4. In the following the structure of the SMT-based model and the necessary constraints are outlined.

Alterations to the Communication Model

In this subsection the general set-up of the networks analysed and the underlying communication model are described: The physical topology of the network under investigation is again described by an undirected graph $G(V, E)$ where the vertices in V consist of all the nodes and the edges E the set of communication links between them. It is differentiated between nodes, jobs and messages, where n, j and m denote their total numbers. The nodes are further classified into e endsystems and s switches such that $n = e + s$. The representation can be deducted from the physical description which is automatically generated using Stanford Network Analyzer Platform library (SNAP) [56] - see Section 5.5 for details. As an example consider Figure 1. To limit the complexity of the model only the sub-graph consisting of switches and their connecting edges is considered for formulating path-constraints: $G'(V', E')$ with $V' \subset V$ and $E' \subset E$. Jobs can only be allocated to endsystems. Messages are sent between jobs respectively and may have to obey logical dependencies. Those can be deducted from the logical model, which is also generated randomly. An example is depicted in Figure 7. In Figure 7(a) the physical model of a network consisting of 11 nodes, 8 endsystems, 3 switches and its associated physical links, is pictured. Figure 7(b) represents the jobs that have to be allocated to the endsystems. The directed edges between the jobs symbolise the messages and the logical dependencies between them. For instance the message from job 1 to job 0 can only be sent if job 1 received a message from job 4.

Jobs can communicate to one another by sending messages passing the communication links within the network visiting at least one switch between source and destination. The traversal can be described by listing all nodes visited in this process. Those lists are not necessarily equal as there may exist different paths from source to destination. The transition of a message m^i from one node k_1 to another node k_2 is denoted by $\langle m^i_{k_1}, m^i_{k_2} \rangle$.

In order to pursuit the SAT approach introduced in the previous chapter time is still considered to be discrete. Again different points in time so-called *timeframes* can be identified and easily distinguished. This guarantees that one can examine the status of each message and its allocation during every timeframe. To be consistent on the notation the size of the model depends on n, j, m and the number of required timeframes needed which are denoted by t.

As collisions of messages within an endsystem or a switch must be avoided only a single message can be allocated to a node in a given timeframe. Therefore, in every timeframe each node can be labelled with the respective message number or zero if the node is idle. A node n_1 is called idle in a given timeframe s if no message is allocated to n_1 in s. Considering this the following representation of the scheduling problem can be obtained:

$$\begin{array}{cccc}
n_1^0 & n_2^0 & n_3^0 & n_4^0 \cdots \\
n_1^1 & n_2^1 & n_3^1 & n_4^1 \cdots \\
\vdots & \vdots & \vdots & \vdots \\
n_1^t & n_2^t & n_3^t & n_4^t . \cdots
\end{array}$$

where n_i^s contains the message id of the message currently allocated to node n_i during timeframe s. If no message is allocated and the examined node is idle $n_i^s = 0$ in timeframe s.

As the SMT-based scheduler permits the use of integer valued variables a more compact representation of the problem in comparison to a SAT-based scheduler introduced in the previous chapter is achieved. Other enhancements include the arbitrary allocation of jobs to nodes and the consideration of multiple messages from one sender. As a consequence the model turns out to be much closer to reality than the SAT-based communication model. Details will be provided in the next sections.

As mentioned in Section 4.3 the SMT-based model is also subject to the following conditions:

1. One job must only be allocated to exactly one endsystem. This assignment cannot be altered in a different timeframe. Jobs must not be allocated to switches and are assigned to endsystems only.

2. Jobs can communicate with one another by sending messages. In every instance only be one message can be dealt with on a single switch.

3. Messages travel between two different jobs via switches and the links between them. Collisions must be avoided, therefore only one message can be assigned to a switch or a link in a certain timeframe. As the edges of the graph resembling the network under consideration are not directed in the model it has to be ensured that neither two switches nor a switch and an endsystem send messages to each other simultaneously.

4. Buffering and different communication channels within a single switch are not considered.

5. Execution time and travelling time from one hop to another are assumed to be one timeframe. All messages are treated as if they were equally sized and can pass through a communication link within one timeframe. Bandwidth is not considered.

Constant name	Description
n	Number of nodes
e	Number of endsystems
s	Number of switches
j	Number of jobs
m	Number of messages
t	Number of timeframes

Table 9: Overview of the input parameters for the SMT-based model

Parameters

As illustrated in Table 9 the numerical parameters needed to generate the input file are the number of nodes n, where it is distinguished between the number of endsystems e and the number of switches s, the number of jobs j, the corresponding number of messages m and the number of considered timeframes t. While n, m and j can be obtained from the physical model, t is varied within a given range. For every possible t from this range the proposed algorithm checks whether a satisfiable solution exists. A feasible solution exits if and only if all jobs can be allocated to nodes in a way that guarantees the reception of all messages considering all temporal and logical constraints. The solver then generates a model which allows the deduction of a schedule. If no allocation of jobs is possible for a given number of timeframes t the scheduler returns unsatisfiable. By varying t an optimal solution can be calculated. The necessary conditions are outlined in the following.

Incremental approach and boundaries

As described optimal solutions can be deducted by varying the number of time-frames t of the problem instances fed into the solver. In order to reduce the problem size it is necessary to determine which values for t have to be considered.
As an incremental approach is proposed a lower bound is suggested for t. A lower bound can be deducted from the logical model by identifying the longest sequence of messages depending on one another. If this sequence is denoted by l_{min} scheduling is conducted with $t = l_{\text{min}}$ as an initial value before incrementation. Because logical dependencies must be obeyed a valid schedule for $t' < l_{\text{min}}$ cannot exist. Therefore, the proposed incremental approach is complete.

Firstly it is checked if a valid schedule exists for l_{min}. If no valid schedule can be found, i.e. YICES2 returns `UNSAT`, then l_{min} is successively incremented until the solver returns `SAT` for a $t'' \geq l_{\text{min}}$ which is then by construction the minimal number of timeframes required and thus the optimal solution.

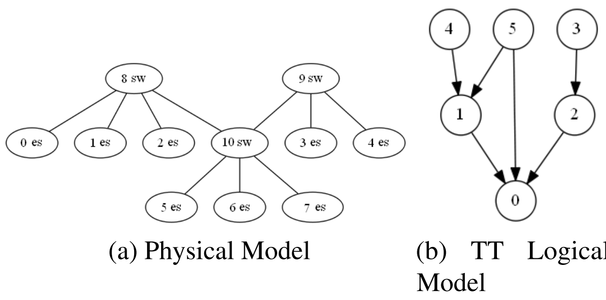
Variables and Constraints

In the following the necessary variables and the resulting constraints are outlined. Their role within the reasoning process is also analysed. The following assumptions are made in order to formulate the scheduling problem:

1. The nodes in the network are labelled from 0 to $n-1$ where $0, 1, \ldots, e-1$ denote the endsystems and $e, e+1, \ldots, n-1$ the switches.

2. For each switch s_i the number of adjoined endsystems is labelled by s_{i_e}.

3. Messages denoted $m_1, \cdots, m_n$ are not allowed to visit the same switch repeatedly which means that no loops are allowed.



(a) Physical Model (b) TT Logical Model

Figure 7: Model example with 11 nodes (3 switches, 8 endsystems), 6 jobs and 6 messages.

Assigning jobs to nodes

Considering the matrix representation established in Section 5.3 once a job j_i is allocated to an endsystem e_j this assignment cannot be altered. Therefore, for every job $j_i, i \in \{0, \ldots, j-1\}$ an integer variable in the range $\Omega := 0, 1, \ldots, e-1$ is implemented independently of the timeframe. Jobs have to be assigned distinctly and at maximum one job can be assigned to an endsystem.

Path

A message will travel from sender to destination via switches. This is modelled by assigning the message id to the nodes passed in this process. The matrix structure of the nodes can be deployed in this context. Assuming n_i and n_j are connected a message m travelling from node n_i via n_j to n_k in consecutive timeframes $s, s+1$ and $s+2$ the following condition has to be fulfilled:

$$n_i^s = m \rightarrow \left(n_j^{s+1} = m \land n_k^{s+2} = m\right).$$

If there exits more than one path between source and destination the arising constraints are formulated disjunctively. Let P_1 to P_t denote the different paths each containing a set of nodes passed as described above, then exactly one path has to be chosen.

Collision-Free Constraint

Furthermore it has to be ensured that no two messages travel through the same link in opposite directions during the same timeframe. Therefore, Boolean variables are introduced for every edge in both directions for every timeframe - denoted $l^s_{n_i n_j}$ for the link between nodes n_i and n_j in timeframe s. It is set to $TRUE$ if the link is busy. To avoid collisions the following condition must hold at all time:

$$l^s_{n_i n_j} = TRUE \rightarrow l^s_{n_j n_i} = FALSE.$$

As these constraints are modelled with Boolean variables only they can efficiently be evaluated in the SAT-solver contained in YICES2. Further details are provided in Section 5.4.

Message Trigger Constraint

For every message m_i each possible path between two endsystems e_i and e_j needs to be considered in every timeframe. For this purpose integer variables denoted by $p^{m_i s}_{e_i e_j}$ for message m_i travelling from e_i to e_j triggered in timeframe s are introduced. $p^{m_i s}_{e_i e_j}$ and can exclusively take the values 0 and message id m_i. Hence it can be ensured that every message is triggered by formulating:

$$\sum p^{m_i s}_{e_i e_j} = m_i. \tag{21}$$

Because equation (21) is integral $p^{m_i s}_{e_i e_j}$ must be integers.

Job Dependency Constraint

Certain messages cannot be triggered during arbitrary timeframes but have to wait for the arrival of another message before the receiving jobs can actually send messages themselves. This means constraints have to guarantee that only after the arrival of message m_1 in the fixed timeframe s the message m_2 can be triggered in $s+1$. In order to keep track of the messages that have reached their destination a further integer variable is introduced for each message m_i in every timeframe to monitor its arrival. Denote these variables by m_i^s for every timeframe s. It is set to the message id m_i if the message has reached its destination and is 0 otherwise. For a message m_j depending on the arrival of m_i the following condition has to hold in timeframe s':

$$m_i^{s'} = 0 \rightarrow p_{e_i e_j}^{m_i s} = 0 \; \forall s : 1 \leq s \leq s'. \tag{22}$$

Combining the constraints (21) and (22) guarantees that two messages that depend on each other are processed in the right order.

Computation of Optimal Solutions

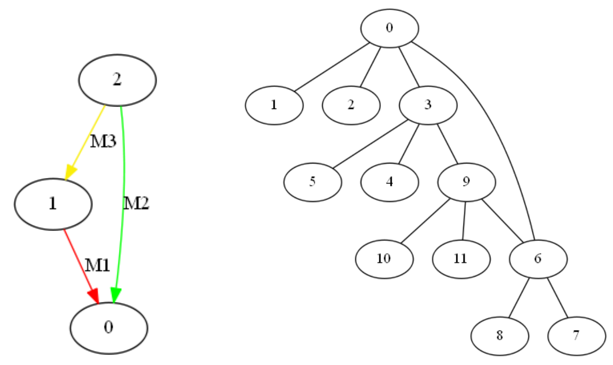
The proposed optimisation process guarantees an optimal solution with respect to time needed for all jobs to be performed successfully. In other words this means that all messages are sent and received in the least number of timeframes possible. By adapting an incremental approach every iteration generates a new problem file providing a different number of timeframes. An optimal solution is found if a problem file with s timeframes is proven to be unsatisfiable and the subsequent iteration with $s+1$ timeframes returns `SAT`.
We will demonstrate our approach by an illustrative example:

Example 5.2. *Allocate three jobs (j_0, j_1, j_2) whose logical dependencies are illustrated in Figure 8a to the architecture (physical model) depicted in Figure 8b.*
Three messages are send:

- *Message 1 from Job 1 to Job 0 (red),*
- *Message 2 from Job 2 to Job 0 (green),*
- *Message 2 from Job 2 to Job 1 (yellow).*

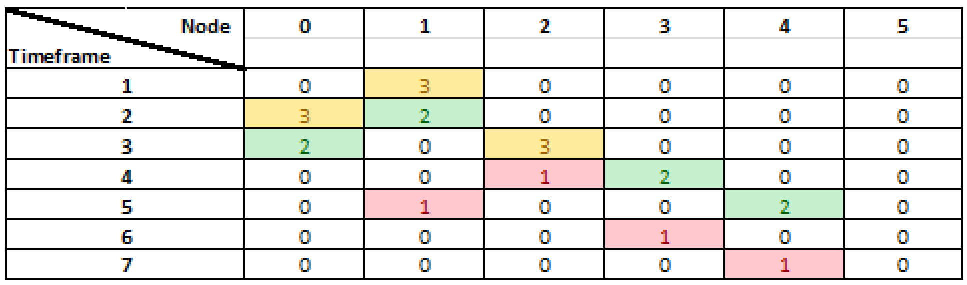
A possible optimal solution allocates the following endsystems to the job variable j_0, j_1, j_2 to illustrate where they are executed:

$$j_0 = 4, j_1 = 2, j_2 = 1.$$



(a) Logical Model (b) Physical Model

Figure 8: Illustration of Example 5.2



Timeframe \ Node	0	1	2	3	4	5
1	0	3	0	0	0	0
2	3	2	0	0	0	0
3	2	0	3	0	0	0
4	0	0	1	2	0	0
5	0	1	0	0	2	0
6	0	0	0	1	0	0
7	0	0	0	0	1	0

Figure 9: Possible Optimal Schedule for Example 5.2.

As a consequence the optimal schedule depicted in Figure 9 can be obtained. The colours represent the messages 1 (red), 2 (green) and 3 (yellow), white boxes labelled 0 *represent idle endsystems or switches in the respective timeframe.*

5.4 Application of YICES SMT

In the following the presented SMT-based scheduling problem is compared to previously introduced SMT-based schedule synthesis for time-triggered multi-hop networks [25]. Furthermore the advantages of the SMT-based representation by exploiting the theory-combination method in YICES2 will be outlined. As the presented model neither considers buffering nor different data link layers the analysis is restricted to three sorts of constraints:

Collision free constraints

In order to benefit from the effects of YICES2's integrated CDCL SAT-Solver Boolean variables and expressions which can easily be transferred into Boolean expressions are used wherever possible in the SMT-based model. The short and compact representation of the path variables in Section 5.3 and the avoidance of collisions in links in Section 5.3 guarantee a conflict free assignment of messages to nodes. Furthermore Steiner refrains from the implementation of purely Boolean constraints and guarantees contention-freedom by formulating ILP problems ensuring that no two messages from the same source can be set off during the same instance of a TT-network.

Path dependency constraints

Given a certain start job a message is triggered from an arbitrary node in a fixed timeframe and traverses the network according to one of the corresponding path constraints. In total this results in the deployment of $2 \cdot l \cdot t$ different Boolean variables where t is the number of timeframes and l denotes the number of links in the network. The corresponding path constraint assigns numerical identifiers to the switches passed by a message. Therefore the required integer variables are in the range r such that $0 \leq r \leq m$ and thus limited to $m + 1$ values. Instead in Steiner's and the presented CPLEX model the path-dependent constraints for the dataflow path are formulated using tuples of integers. The traversal of message is described by a tuple containing the links passed between sender and destination. For each message every single link visited is stored. The transition from one node to another is again controlled by ensuring that a message has to arrive at a node at least one instance before leaving it. This results in an integral constraint for every message and possible link between two nodes.

Message trigger constraints

In the presented model a pseudo-Boolean representation as illustrated in Section 5.3 is deployed to ensure that all messages are actually triggered. The solver will internally convert this into a Boolean problem. Steiner uses end-to-end transmission constraints to describe the maximum allowed end-to-end latency for a message as linear equations for every message. Therefore an integer theory logic has to be applied by the solver.
The SMT solver YICES2 used in the proposed scheduler includes a Boolean satisfiability solver and in addition theory solvers for four main theories: uninterpreted functions with equalities (UF), linear arithmetic (LA), bitvectors (BV) and arrays.

Details can be found in [57]. In YICES2, theory combination always involves UF on one side and either AV or BV. Due to the use of tuples and arrays this approach cannot applied to Steiner's model.
As quantifiers are not used in the SMT-based model a range can be assigned to all necessary variables. This significantly reduces the search space for possible solutions. The domain used in all experiments conducted ranges from 0 to the number of messages m where 0 denotes an idle node.

5.5 Results

The comparison between YICES2 and the CPLEX model is based on 17 example scenarios that are applied to both discussed models. These scenarios are randomly generated based on the SNAP library [56], where networks are constructed from a given number of switches, endsystems and jobs required. The necessary messages are generated based on the information of the jobs. Each scenario comprises of a physical and a logical model, whose properties are described in detail in Section 5.3.
Experiments are executed using CPLEX 12.6.1 and YICES2.4 both running on an Intel(R) Xeon(R), CPU E5645, 2.40GHz, 64 GB RAM with operating system Linux Mint Release 13 (maya) 64-bit. This setup allows a comparison of performance and an evaluation of the two models outlined.

Sequential approach

Table 10 depicts the execution times along with the number of constraints for the different physical and logical topologies on both target systems in columns labelled CPLEX and YICES2. As explained in detail in Chapter 5.3 scenarios are distinguished by the number of switches (SW), endsystems (ES), jobs and messages. The seventh column labelled "Timeframes" shows the least number of timeframes needed to complete all jobs. The difference between the number of timeframes and the lower bound allows to conclude how many iterations are necessary to compute a result with the incremental solver proposed. The results have been verified with the commercial MILP problem solver IBM CPLEX.
The results clearly indicate that the proposed SMT-based scheduler can compete with state-of-the-art schedulers and is able to produce optimal schedules for the scenarios investigated. This shows that the model presented in Section 5 is an appropriate basis to compute schedules for distributed systems. Starting the incremental search for a valid assignment of a lower bound l_{min} proved to be sensible as on the one hand the number of variables and constraints is smaller if less time-

frames are available and on the other hand it has been observed that contradictions i.e. unsatisfiable assignments are faster to find with YICES2. In over 70% of the benchmarks the SMT-based scheduler generates a schedule in less time than the CPLEX-based scheduler.
The last two columns of Table 10 also outline how runtimes of the SMT-based solver increase when the framework is executed on a Raspberry. In general a slowdown of at least one magnitude can be witnessed.

Memory Footprint

During the conducted experiments a significant reduction of the required amount of memory was observed when applying the YICES2-based SMT solver. Experimental evidence is provided in Table 11. Here the peak amount of memory required by the CPLEX-based scheduler is depicted and compared to memory footprints of the SMT-based scheduler. It has been witnessed that the required memory of the YICES2-based scheduler peaks when the satisfiability of the optimal solution is verified i.e. when the minimal number of timeframes is found.
The effect described above can be witnessed most clearly when comparing scenarios for which the computation time of an optimal schedule exceeds 2 seconds. This is usually the case if the number of messages is increased. Therefore the SMT-based scheduler is more appropriate for application on a distributed embedded system with limited processing and memory resources as the memory consumption can be decreased by more than factor 100 in certain scenarios.

5.6 Execution on MPSoC-emulating target systems

It may be of vital importance to compute a schedule on the embedded system under consideration itself. For instance in safety-critical environments components may fail and therefore jobs cannot be executed on the endsystems they were originally allocated to.
In order to emulate the application of the proposed SMT-based scheduler we have successfully ported the framework on a Raspberry Pi2 Model B @ CPU 900MHz x 4, with 927MiB system memory. This enables the evaluation of performance on an ARM-cortex. In order to analyse the performance of this MPSoC emulating architecture we repeated the execution of the experiments from the previous section. We followed the same setup and also generated the problem files on the target platform according to the description in Section 5.5. In order to provide a better overview the execution times are provided in the final column of Table 10.
The first notable fact is the ability to actually compute optimal schedules on this

Scenario	**Nodes**	**ES**	**SW**	**Jobs**	**Messages**	**Timeframes**	**Lower Bound**	**CPLEX**	**YICES2**	**YICES2 R2**
S1	18	14	4	5	6	9	3	**0.16**	1.18	72.97
S2	17	14	3	11	17	13	4	181.69	**9.63**	212.19
S3	26	21	5	11	16	21	7	**9.70**	567.22	4,707.00
S4	20	16	4	10	13	14	4	130,376.96	**3.87**	211.51
S5	12	9	3	9	10	12	3	**4.88**	5.14	55.82
S6	17	13	4	8	11	14	4	7.26	**3.03**	182.06
S7	15	12	3	8	8	12	4	3.39	**0.42**	34.69
S8	10	8	2	7	11	13	4	**0.37**	0.57	18.74
S9	12	10	2	6	8	10	3	**0.20**	0.61	12.83
S10	25	20	5	5	4	6	2	0.20	**0.19**	55.94
S11	25	20	5	6	7	14	4	1.71	**1.07**	178.15
S12	25	20	5	8	11	14	4	9.53	**4.54**	521.09
S13	24	20	4	10	12	12	4	3.89	**1.74**	199.57
S14	24	20	4	15	25	17	5	>2days	**107.65**	8,813.17
S15	24	20	4	19	26	19	6	>2days	**46.67**	1,427.97
S16	30	27	3	20	35	26	6	>2days	**523.06**	7,887.53
S17	27	24	3	24	41	32	7	>2days	**24,140.88**	117,721.69

Table 10: Results for 17 scenarios comparing CPLEX- and YICES-based schedulers on different platforms, all execution times in seconds.

Scenario	**CPLEX**	**YICES2**	**Proportion**
S1	<1	6	-
S2	50	9	5.56
S3	153	38	4.03
S4	351	15	23.40
S5	20	6	3.33
S6	24	12	2.00
S7	<1	6	-
S8	15	5	3.00
S9	<1	4	-
S10	<1	5	-
S11	15	12	1.25
S12	26	17	1.53
S13	25	12	2.08
S14	>3,500	35	>100
S15	>1,600	39	>41
S16	>11,000	29	>379
S17	>8,400	941	>8

Table 11: Demonstration of the Memory peaks for CPLEX- and YICES2-based schedulers in MB.

limited architecture. CPLEX does not provide support for ARM-cortices and can therefore not be used as reference or in fact to compute new schedules on the architecture itself if necessary. Due to the limited resources however we witness the loss of one magnitude on average which still allows the computation of results in reasonable time for small and medium-sized examples.
In order to further accelerate the computation of optimal schedules on the target architecture itself we will propose different approaches to execute the proposed framework in parallel in the next chapter, because modern multicore chips usually provide idle resources which can be exploited.

5.7 Summary

The growing size of TT-networks and their increasing complexity demand high-performance schedulers to plan the communication schedule. In this chapter the formal scheduling constraints that apply in a time-triggered network-on-chip have been defined in a way the state-of-the-art SMT solver YICES2 can be deployed to compute valid schedules. Additionally an SMT-based scheduler, which is able to compute optimal schedules for time-triggered architectures incrementally, has been introduced.
The scheduler has been validated using an ILOG CPLEX based MILP model. It has been shown that the scheduler based on the SMT-solver YICES2 is an effective alternative to state-of-the-art schedulers. Furthermore it has been outlined how porting the scheduler to a distributed embedded system enables the SMT-based solver to execute the scenarios on the target system itself. Finally it has also been demonstrated that applying the SMT-based scheduler could significantly reduce the amount of memory required especially on larger benchmarks.
We successfully implemented a complete scheduler which also allocates jobs. The incremental approach chosen guarantees an optimal allocation of jobs minimising the overall makespan. The general problem of finding optimal communication schedules is, as discussed in detail previously, $\mathcal{NP}$-complete and therefore scalability is an important issue especially as it is estimated that the number of endsystems on future architectures will grow exponentially.
The development of an efficient scheduling framework which can be deployed to full extend even in resource constraint systems (i.e. low memory or small CPUs) becomes of vital importance. This is due to the fact that safety-critical embedded systems may only have to perform unpretentious tasks requiring limited hardware and rather focus on reliability and persistence.
Therefore we will dedicate the next Chapter to discuss different parallel approaches to accelerate the proposed scheduling framework and reduce computation time.

We will thus show how parallel checks for satisfiability of problem instances can not only compensate but significantly reduce runtime.
Obviously it is important to investigate different strategies to further reduce computation time of optimal schedules on the one hand. On the other hand it may prove sensible to debate under which conditions feasible solutions are sufficient. Therefore we will investigate the trade-off between computation costs and quality of the solution if different, not necessarily optimal methods are applied, to the scheduling problem. During this process we will intensively compare the proposed SMT-based scheduler to the application of heuristics to compute feasible but no longer optimal schedules.
Finally our objective will be to work towards online verification in safety-critical systems where for instance jobs could be rescheduled during runtime if either a communication link, a switch or an endsystem fails. We will investigate this complex in Chapter 7.

6 Parallel Computation of Schedules

6.1 Introduction

The objective of this chapter is the discussion of different attempts to use parallel computing to accelerate the computation of schedules for the systems under consideration. We will begin by investigating the impact of parallel computing on optimal schedules applying the proposed scheduling framework by simultaneously checking different problem instances for satisfiability. We will demonstrate how the parallel execution of the proposed scheduling framework actually allows us to compare the performance on an MPSoC emulating target system to runtimes obtained from regular machines with large memory and chip resources.
In addition to this we will also show how the so-called dominator concept can be applied to split the logical dependencies before computing schedules. This technique may sacrifice optimality but on the other results in a significant reduction of the problem size. This is due to the fact that the logical dependencies are distributed into two problem instances which can be scheduled in parallel. We will evaluate both approaches on large and medium sized benchmarks. For this purpose we will also develop a three-dimensional, regular mesh to provide the necessary freedom of allocation for the second approach.
Both approaches have already been partially published in [3] and [4] respectively.

6.2 Parallel Checks for Satisfiability

Recalling the results from Table 10 in Section 5.5 it can be detected that porting the framework to the Raspberry-based embedded platform results in a significant deterioration in performance by at least one magnitude for each scenario. However, the incremental approach chosen can easily be distributed to different available computational cores by simultaneously checking the problem instance for satisifiability allowing a different number of timeframes in parallel.
More explicitly this means in the most basic version to check simultaneously whether a given scheduling problem can be solved in t_1 or t_2 timeframes for $t_1, t_2 \in \mathbb{N}, t_1 < t_2$. Three different results are possible:

1. The proposed scheduling framework reports `UNSAT` for t_1 and t_2. In this case we will continue the incremental approach with t'_1 and t'_2 where $t'_1, t'_2 \in \mathbb{N}, t'_1 < t'_2$ and $t'_1, t'_2 > t_2$.

2. The proposed scheduling framework reports `UNSAT` for t_1 and `SAT` for t_2. In this case the optimal solution is located between t_1 and t_2. Therefore, we

continue with t'_1 and t'_2 where $t'_1, t'_2 \in \mathbb{N}$ and $t_1 < t'_1 < t'_2 < t_2$.

3. Theoretically it possible that the proposed scheduling framework reports `SAT` for t_1 and t_2. In this case we will continue the incremental approach with t'_1 and t'_2 where $t'_1, t'_2 \in \mathbb{N}, t'_1 < t'_2$ and $t'_1, t'_2 < t_1$. However, if t_1 is chosen according to the lower bound this scenario cannot occur.

In the following part we will generalise this concept to n computation nodes. For this purpose we require n different numbers of timeframes which will be denoted $t_1 < t_2 < \cdots, < t_n$ and will be simultaneously checked for satisfiability. The computations t_1 to t_n will be performed in parallel. Therefore, if YICES2 returns `UNSAT` for a value $t_i \in \{t_1, t_2 \cdots t_n\}$ all runs for $t < t_i$ can be aborted immediately because there cannot be a solution requiring less timeframes. Vice versa if YICES2 returns `SAT` fora value $t_j \in \{t_1, t_2 \cdots t_n\}$ all all runs for $t > t_j$ can be terminated as they must return feasible, yet not optimal, solutions by construction.
We will suggest three different approaches to define sensible numbers of timeframes to start. The following approaches have been investigated:

1. For n computation nodes calculate schedules for $t, t+1, \cdots, t+n-1$ simultaneously.

2. For n computation nodes calculate schedules for $t, t+2, \cdots, t+2(n-1)$ simultaneously.

3. For n computation nodes calculate schedules for $t, t+n, t+2n, \cdots, t+n(n-1)$ simultaneously.

We will denote an optimal solution $\hat{t}$. The essential property of $\hat{t}$ can be described by the fact that YICES2 returns `SAT` for $\hat{t}$ and `UNSAT` for $\hat{t}-1$. Now we can formulate criteria for termination of the approaches described above. Approach 1 and 2 will terminate automatically as soon as an optimal solution is found. The third approach will firstly define the set $\{t', t'+1, \cdots, t'+n]$ where $\hat{t}$ is located for $t' \in \{t_1, t+n, t+2n, \cdots t+n(n-1)\}$ with YICES2 returning `UNSAT` for t' and `SAT` for $t'+n$. Once the set containing the optimal solution has been identified only one further execution of the framework is required checking $t', t'+1, \cdots t'+n-1$ for satisfiability simultaneously. All proposed strategies are also depicted in Figure 10.

Scenario	Nodes	SW	Jobs	Messages	Timeframes	Lower Bound	CPLEX	YICES2	YICES2 R2	Distributed R2
S1	18	4	5	6	9	3	0.16	1.18	72.97	1.41
S2	17	3	11	17	13	4	181.69	9.63	212.19	41.01
S3	26	5	11	16	21	7	9.70	567.22	4,707.00	237.34
S4	20	4	10	13	14	4	130,376.96	3.87	211.51	10.84
S5	12	3	9	10	12	3	4.88	5.14	55.82	1.32
S6	17	4	8	11	14	4	7.26	3.03	182.06	14.95
S7	15	3	8	8	12	4	3.39	0.42	34.69	0.32
S8	10	2	7	11	13	4	0.37	0.57	18.74	0.71
S9	12	2	6	8	10	3	0.20	0.61	12.83	0.28
S10	25	5	5	4	6	2	0.20	0.19	55.94	0.61
S11	25	5	6	7	14	4	1.71	1.07	178.15	3.89
S12	25	5	8	11	14	4	9.53	4.54	521.09	16.62
S13	24	4	10	12	12	4	3.89	1.74	199.57	8.45
S14	24	4	15	25	17	5	>2days	107.65	8,813.17	560.00
S15	24	4	19	26	19	6	>2days	46.67	1,427.97	352.84
S16	30	3	20	35	26	6	>2days	523.06	7,887.53	4,449.49
S17	27	3	24	41	32	7	>2days	24,140.88	117,721.69	105,864.95

Table 12: Complete overview of execution times for 17 scenarios comparing CPLEX- and YICES2-based schedulers on different platforms. **YICES2 R2** denotes execution time on Raspberry Pi 2. **Distributed R2** denotes fastest parallel execution approach. All execution times in seconds, number of endsystems similar to Table 10.

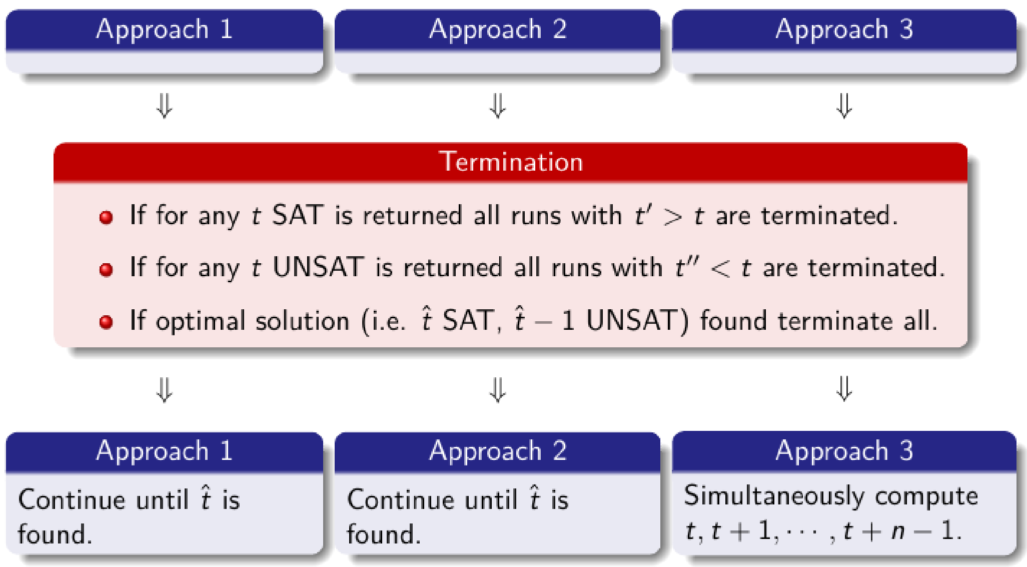

Approach 1
Approach 2
Approach 3
Termination
If for any t SAT is returned all runs with $t' > t$ are terminated.
If for any t UNSAT is returned all runs with $t'' < t$ are terminated.
If optimal solution (i.e. $\hat{t}$ SAT, $\hat{t}-1$ UNSAT) found terminate all.
Approach 1
Continue until $\hat{t}$ is found.
Approach 2
Continue until $\hat{t}$ is found.
Approach 3
Simultaneously compute $t, t+1, \cdots, t+n-1$.

Figure 10: Parallelisation of incremental scheduling

6.3 Results

For our experiments we choose $n = 6$ compute nodes as our MPSoC emulating platform consists of six Raspberry Pis connected via a single switch. The results thus are again obtained using six Raspberry Pi 2 Model B @ CPU 900MHz x 4, with 927MiB system memory. They are used to simulate the parallel execution of the proposed SMT-based scheduler on a system emulating an embedded system with restricted memory and limited computational properties.
The strategies to distribute problem instances are outlined in the previous subsection: We explicitly consider the first approach and simultaneously run six successive problems beginning with $t_0 = l_{\text{min}}$ many timeframes, increasing timeframes respectively. In this context l_{min} denotes the lower bound depicted in the seventh column of Table 12. All six runs can be aborted if the largest $t = t_0 + 5$ returns UNSAT as all instances with a lower number of timeframes must be UNSAT by construction. Then continue the reasoning process starting with $t_0 + 6, t_0 + 7, \ldots, t_0 + 11$. We continue in this fashion until optimal neighbours are found.
Table 12 serves as an overview illustrating the total experimental setup. In the final column the most successful (i.e. fastest) parallel approach is depicted. Thus the performance can be evaluated in comparison to the sequential execution scheme on the Raspberry Pi 2 and also on the same reference computer, we have applied to execute CPLEX.
In the following Table 13 the results for the three different parallel approaches are

outlined in more detail. The final column of Table 12 is broken to explicitly depict the performance of the three strategies outlines in Section 6.2.

Scenario	Approach 1	Approach 2	Approach 3
S1	3.17	2.43	1.41
S2	66.53	59.86	41.01
S3	307.49	399.16	237.34
S4	19.38	24.87	10.84
S5	2.19	1.32	1.32
S6	14.95	22.15	14.95
S7	1.27	1.27	0.32
S8	2.51	2.51	0.71
S9	0.99	0.99	0.28
S10	2.87	1.24	0.61
S11	6.2	10.55	3.89
S12	31.96	42.64	16.62
S13	17.44	20.75	8.45
S14	779.5	719.43	560
S15	413.67	586.29	352.84
S16	4618.32	4641.63	4449.49
S17	106510.89	106510.89	105864.95

Table 13: Scheduling performance of three different parallel strategies, all runtimes in seconds

Results show that runtime can be significantly reduced by applying parallelisation. As depicted in the last column of Table 12 this strategy results in a significant reduction of runtime enabling the SMT-based scheduler to compete with the results obtained incrementally using a high performance machine. Special attention should be dedicated to the results for benchmarks S5, S7 and S9. In these three scenarios the parallel approach executed on an MPSoC emulating target system outperforms the results obtained on the high performance machine regardless of applying YICES2 oder CPLEX. Again we refer to Table 10 for further details.

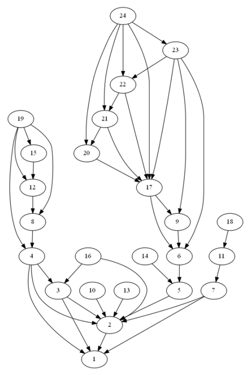


Figure 11: A logical model for 24 jobs

6.4 Dominator-based Partitioning

In the following a strategy using so-called *graph dominators* is discussed which will enable us to partition the logical model and hence compute optimal solutions for smaller sub-problems. These solutions will be combined into a global schedule which will no longer be optimal but feasible.

As we have seen the computation of optimal schedules is time-consuming as the scheduling problem is known to be $\mathcal{NP}$-complete [29]. Therefore, it is sensible approach to partition the scheduling problem into smaller sub-problems, which can then be scheduled in parallel before combining the resulting schedules into a global schedule for the system under investigation. We will discuss a concept based on graph dominators to partition the logical model. The partitions created will also be referred to as *clouds* or *splits*.

Dominators were already introduced in 1959 to examine flow diagrams [65] and an efficient algorithm was implemented in 1979 [66]. Dominators provide information about origin and end of re-converging paths and can be used to create possible partitions of the network under investigation. Dominators are widely used in the analysis of graphs and flowcharts for instance in logic synthesis, decomposition and industrial circuit designs [67]. Dominator trees can be calculated with a complexity of $O((V + E)log(V + E))$ [68], where V and E denote the numbers of vertices and edges in a directed flow graph respectively. The notations and definitions outlined in [67] are adapted in the following.

In a directed graph (in our case the logical model $G(V, E)$) a dominator $u \in V$ of a vertex $v \in V$ with respect to some vertex $w \in V$ is a vertex which is contained in every path starting from v to w. The set of dominators of some vertex $u \in V$ with respect to some vertex $w \in V$ is denoted $Doms(v, w)$. The immediate dominator z of a vertex $v \neq z$ with respect to some vertex r dominates v and itself and does

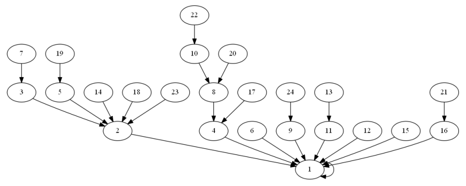


Figure 12: Dominator tree for the logical model depicted in Figure 11

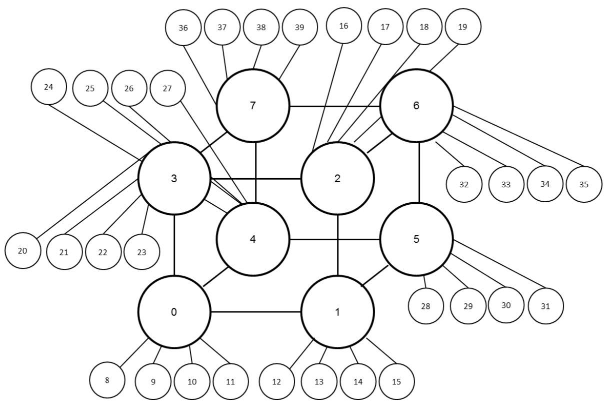


Figure 13: Model of three-dimensional Distributed System with 32 endsystems and 8 switches

not dominate any other dominator of v and is denoted by $z = idom(v, r)$. The edges $\{\langle idom(w, r), w\rangle | w \in V - \{r\}\}$ form a directed tree rooted at r which is called the *dominator tree*. As an example consider the dominator tree of the logical model depicted in Figure 11 which is illustrated in Figure 12. The complete transformation process is simulated in an illustrative example in the next section.
Dominators provide a general mechanism for identifying so called *re-converging paths* in graphs. Two or more paths are said to be re-convergent if their source and destination nodes are similar. If a vertex u is the origin of a re-converging path, then the immediate dominator of u is the earliest point at which such a path converges. Because the main aim when splitting the logical dependencies is minimising the communication between the emerging partitions, a partition containing v should also contain $Doms(v, w)$. A complete illustrative example for the graph in 15a is given as a conclusion at the end of the following section.
The traversal can be described by listing all nodes visited in this process. Those lists are not necessarily equal as there may exist different paths from source to destination. In order to avoid collisions of messages on a switch or an endsystem only a single message can be allocated to a switch or and endsystem in a given timeframe. Therefore, in every timeframe each node can be labelled with the respective message number or zero if the node is idle. A node n_1 is called idle in a given timeframe s if no message is allocated to n_1 in s . Considering this the following representation of the scheduling problem can be obtained:

$$\begin{array}{cccc}
n_1^0 & n_2^0 & n_3^0 & n_4^0 \cdots \\
n_1^1 & n_2^1 & n_3^1 & n_4^1 \cdots \\
\vdots & \vdots & \vdots & \vdots \\
n_1^t & n_2^t & n_3^t & n_4^t . \cdots
\end{array}$$

where n_i^s contains the message id of the message currently allocated to node n_i during timeframe s. If no message is allocated and the examined node is idle $n_i^s = 0$ in timeframe s.

Dominator-Orientated Decomposition of Logical Dependencies

In this section the partitioning techniques and the static compaction are described and the proposed dominator-orientated approach is demonstrated by the illustrative example from Figure 11.
First of all the dominator tree is computed for the logical dependencies under investigation using the Lengauer-Tarjan algorithm [68]. In order to do so the logical dependency graph G has to be modified by changing the directions of all its edges. Thus a graph G' with entry node job 1 is constructed for scenarios allowing

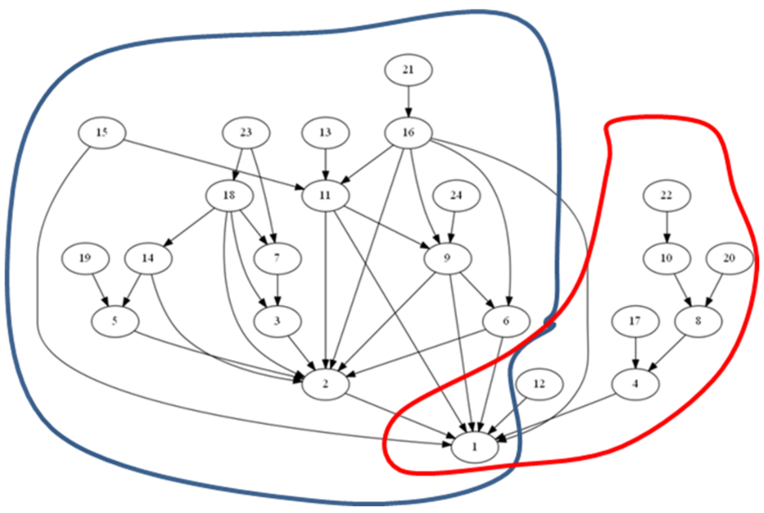


Figure 14: Two partitions for Figure 11

a meaningful application of Lengauer-Tarjan's algorithm. From the dominator tree information about jobs inducing each other can be deduced. Considering the dominator tree the logical model depicted in Figure 11 depicted one can see for instance that node 23 is dominated by node 2 and thus all paths beginning at 23 re-converge into node 2. Therefore, all messages (i.e. directed edges) ought to be contained in the same partition as it is aimed to split the logical dependencies into different clouds minimising the interaction between them. Because job 1 is always the sink this is never completely possible which is why the assignment of one job to different clouds is possible. Figure 14 shows how two different clouds can be constructed from Figure 11.

This dominator-based decomposition results in two clouds without any communication between them. Therefore, they can be scheduled independently and in parallel on the architecture depicted in Figure 13. It has to be ensured, however, that job 1 is allocated to the same node for both clouds as otherwise a non-resolvable contradiction would occur. Therefore, an additional constraint has to be implemented before computing schedules i.e. constantly allocating job 1 to endsystem 1. As the architecture is symmetric this will not have an impact on the optimality of the individual schedules for the two clouds.

Each schedule independently allocates jobs to possible endsystems. Therefore

before composing a global schedule, it has to be checked that endsystems are not overstaffed. If this is the case jobs have to be shifted for instance to idle endsystems attached to the same switch. If no idle resources can be provided, one can try to rotate the allocation of all jobs from the same schedule to a different switch as the network under investigation is symmetric. If these strategies prove to be successful and all conflicts can be resolved a feasible global schedule can be obtained by simply combining the two sub-schedules consecutively.
This approach can be refined by applying so-called static compaction: This means once all conflicts in the allocation of jobs have been resolved, the sub-schedules $s_1, s_2, \ldots, s_n$ are combined in a joint schedule side by side. Then the first time-frames is checked for collisions on vertices or edges. If no collisions are reported the analysis terminates and global schedule has been found. If a collision is identified the conflicting schedule s_i is shifted by one timeframe. Then the checking is repeated until no further collisions occur. The resulting schedule is then a feasible global solution.

Outlining the splitting algorithm

In this subsection we want to formalise the approach outlined in the previous chapter in order to propose an algorithm to split the logical dependencies automatically. Fur this purpose it can be assumed we have created the dominator tree T for a directed graph G' using the Lengauer-Tarjan algorithm. In this context G' denotes the graph the logical model G has been transformed into by changing the direction of all edges. Now let $\Omega := \{1, 2, \cdots, j\}$ denote the number of jobs (i.e. the nodes in the dominator tree) and let

$$\delta : \Omega := \{1, 2, \cdots, j\} \to \mathbb{N}$$

determine the number of nodes reachable from some vertex i in the dominator tree. Here reachability refers to the ability to get from one vertex to another within a graph. A vertex i' is defined to be *reachable* from i if it can there exists a sequence of adjacent vertices (i.e. a path) which starts with i and ends with i'.
By construction all vertices are reachable from the entry node. Thus job 1 is not considered in the following. Denote with $\hat{i}$ the first vertex in the dominator tree where $\delta(\hat{i})$ is maximal i.e. $\forall i \in \Omega : \ \delta(\hat{i}) \geq \delta(i)$. Now take $\hat{i}$ from the dependency graph and add it to the split S_1. Then traverse G' performing a depth first search (DFS) starting at $\hat{i}$ and append all vertices visited during this process to S_1. If the nodes contained in $S_1 \cup \{1\}$ and G' are identical, repeat the process with the next smaller or equal value for $\hat{i}$. Algorithmically this can be achieved by deleting $\hat{i}$ from the dominator tree T, re-assigning $\delta(i)$ to all nodes i remaining in T and thus

identifying a new vertex $\hat{\imath}$. If $S_1 \subsetneq G'$, i.e. $S_1 \cup \{1\}$ and G' are different add all nodes not included in S_1 to a second split S_2.
All nodes k with $\delta(k) > \delta(\hat{\imath})$ are added to both splits. This means that all jobs which were checked but not used as starting points for the proposed splitting heuristic are part of both partitions. In case that the first value for $\hat{\imath}$ already results in a valid distribution of jobs, job 1 is added to both splits.
Another goal of the proposed splitting technique is to distribute the workload as balanced as possible: If the number of messages in S_1 and S_2 are denoted by m_{S_1} and m_{S_2} respectively and without loss of generality let $m_{S_1} \geq m_{S_2}$ then repeat the splitting procedure with the next smaller or equal value for $\hat{\imath}$ if $\frac{m_{S_1}}{m_{S_2}} > 3$.
Therefore for valid splits the following restriction holds:

$$m_{S_1} \leq 3m_{S_2}.$$

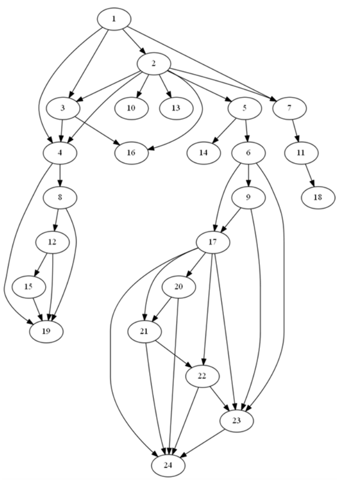
Finally it has to be ensured that no messages are sent from S_1 to S_2 or vice versa. If this is the case then the sending job will also be included in both partitions thus making communication between the splits obsolete. The problem size will however be enlarged because the sending and receiving jobs are scheduled redundantly in each partition.

An illustrative example

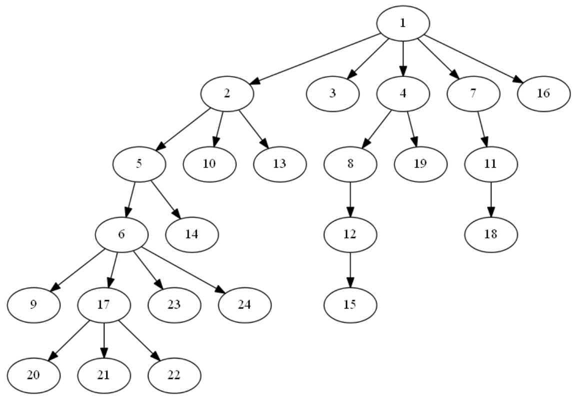
In order to illustrate Algorithm 2 the example depicted in Figure 7b is explained in detail. Each step is visualized in Figure 15.
First the dependency graph G is transformed in the corresponding graph G' by changing the directions of all edges which can be seen in Figure 15a. From G' the dominator Tree shown in Figure 15b is computed using the algorithm of Lengauer-Tarjan. By counting the number of children for each job different to job 1 in the dominator tree, $\hat{\imath} = 2$ can be identified as depicted in Figure 15c because $\delta(2) = 12$. Now $\hat{\imath} = 2$ is added to S_1 and the subgraph of G' rooted at $\hat{\imath} = 2$ is also copied into S_1. As $G' = S_1 \cup \{1\}$ the split is not valid and the proposed algorithm continues with $\hat{\imath} = 5$ as $\delta(5) = 9$ is the new maximum.
Again $\hat{\imath}$ is added to S_1 and all nodes reachable from node 5 $\hat{\imath}$ in G' are appended. This time $S_1 = \{5, 6, 9, 14, 17, 20, 21, 22, 23, 24\}$ and job 2 is also added as it has been previously checked for its splitting properties. Finally $\hat{\imath} = 2$ is also added to S_2 which is then completed by adding all the vertices from $G \setminus S_1$. As the ratio between S_1 and S_2 is almost $1:1$ i.e. $\frac{m_{S_1}}{m_{S_2}} = \frac{21}{20} \approx 1 \leq 3$ the algorithm terminates and schedules for each partition are computed.

Algorithm 2 Splitting the dependency graph

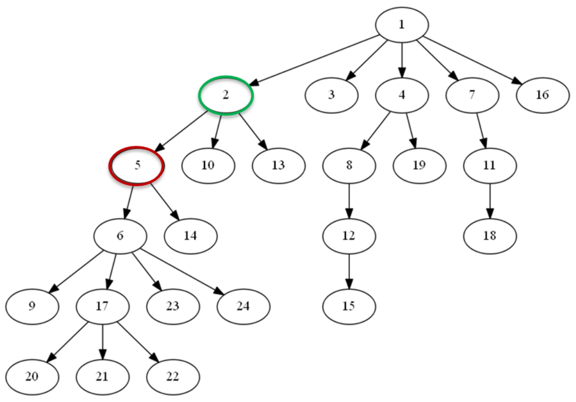
```
1:  procedure SPLIT($G$)                                  ▷ Logical Dependency Graph $G$
2:      compute $G'$                                                      ▷ Graph $G'$
3:      compute dominator Tree $T$
4:      while true do                                  ▷ Terminate if $S_1$ is complete
5:          if $T = \emptyset$ then
6:              break
7:          end if
8:          create empty split $S_1$
9:          $\forall i \in T$ count nodes reachable $\delta(i)$
10:         identify $\hat{i} == max(\delta(i))$
11:         DFS to find vertices reachable from $\hat{i}$
12:         add vertices to $S_1$
13:         if ($S_1 = G'$) or (ratio $> 3$) then remove $\hat{i}$ from $T$
14:         else
15:             break
16:         end if
17:     end while
18:     create $S_2$
19:     $S_2 = G' \setminus S_1$
20: end procedure
```



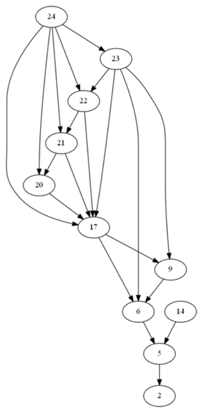
(a) The transformation of G into G'



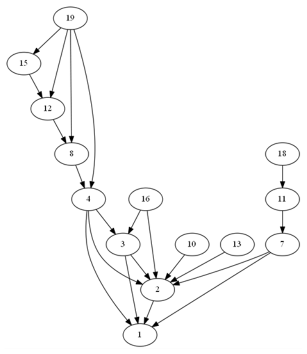
(b) Dominator Tree T



(c) Identifying $\hat{i}$ in T



(d) Split $S1$ containing all vertices reachable from job 5.



(e) Split $S2$ containing job 2 and all vertices in $G \setminus S1$.

Figure 15: Splitting the logical dependency graph G from Figure 7b using the proposed dominator-oriented approach

6.5 Results

This section provides results on seven benchmarks of varying complexity. The benchmarks were identified with respect to the results from Chapter 5.5. We have focused on the most time-consuming benchmarks in order to outline the effects of dominator-based partitioning.

Experiments were again executed using YICES2.3 running on an Intel(R) Xeon(R), CPU E5645, 2.40GHz, 64 GB RAM with the operating system Linux Mint Release 13 (maya) 64-bit. As the benchmarks have originally been chosen under complexity aspects in order to analyse the scalability and performance of the SMT-based scheduler, almost as many jobs as endsystems have to be allocated. This means there are hardly any idle endsystems, which prevents a successful application of static compaction due to conflicts, when allocating jobs simultaneously. In order to rectify this bottleneck, all experiments have been performed on an artificial architecture with 32 endsystems and 8 switches which is depicted in Figure 13. This architecture provides the necessary freedom for allocation and significantly improves results in runtime and static compaction as demonstrated in Table 14.

The table illustrates the performance of different sized scenarios, where ES, SW, M and TF denote the number of endsystems, switches, jobs and messsages respectively. The column labelled TF shows the number of timeframes needed for an optimal solution with necessary execution time depicted in the neighbouring column. In the following columns the performance after partitioning the logical dependencies is described. As the schedules for different clouds can be computed simultaneously only the slowest execution time is stated in the column labelled *Worst Execution Time*. Adding the number of timeframes necessary for each optimal cloud schedule results in the worst case number of timeframes (TF). The performance gain and the loss of optimality are quantified in the respective columns before finally the effect of static compaction is portrayed. Here S1S2 and S2S1 describe the different combinations of the two clouds $S1$ and $S2$, which can be connected either way.

In general it can be witnessed that the runtime to compute a feasible global schedule is significantly decreased at the expense of possibly losing optimality. Therefore, it might take longer to complete all jobs and receive all messages which may prove to be problematic if the scenarios have to be completed within a given number of timeframes. This may be the case for real-time periodic jobs.

The illustrative example from Figure 14 is denoted S21-2Clouds and accordingly S21-3D-2Clouds when executed on the architecture in Figure 13. As one can see the execution time is significantly reduced regardless of the underlying architecture. However, static compaction cannot be performed if the original physical

Scenario	Nodes	ES	SW	J	M	TF	Execution Time	Clouds	Execution Time Worst Case	Performance Gain	Worst Case TF	Optimality Loss	S1S2	S2S1
S2	17	14	3	11	17	13	9.22	2	1.13	**8.15**	26	**2.00**	20	19
S2-3D	40	32	8	11	17	13	57.68	2	15.18	**3.80**	25	**1.92**	17	15
S16	30	27	3	20	29	18	45.13	2	6.78	**6.65**	28	**1.56**	collision in endsystems	
S17	27	24	3	24	41	25	24,138.18	2	228.60	**105.59**	32	**1.28**	collision in endsystems	
S18	30	27	3	20	35	26	1,939.32	3	29.68	**65.35**	41	**1.58**	collision in endsystems	
S19	30	27	3	20	39	27	11,970.42	2	174.18	**68.72**	36	**1.33**	collision in endsystems	
S20	27	24	3	24	35	20	42,378.72	2	3.53	**11997.83**	32	**1.60**	collision in endsystems	
S16-3D	40	32	8	20	29	19	1,321.91	2	268.89	**4.92**	34	**1.79**	20	20
S17-3D	40	32	8	24	41	32	14,155.34	2	224.50	**63.05**	41	**1.28**	21	21
S18-3D	40	32	8	20	35	20	1,882.60	3	29.68	**63.43**	41	**2.05**	collision in endsystems	
S19-3D	40	32	8	20	39	23	5,133.92	2	1,744.79	**2.94**	36	**1.57**	34	31
S21-2Clouds	27	24	3	24	38	23	27,945.08	2	84.27	**331.63**	34	**1.48**	32	30
S21-3D-2Clouds	40	32	8	24	38	21	3,363.50	2	2703.50	**1.24**	34	**1.62**	32	29
S20-3Clouds	27	24	3	24	35	20	42,378.72	3	4.72	**8977.97**	46	**2.30**	collision in endsystems	
S20-3D-3Clouds	40	32	8	24	35	26	6,479.47	3	171.11	**37.87**	47	**1.81**	collision in endsystems	

Table 14: Results for seven scenarios illustrating the trade-off between runtime and quality of solution when splitting logical dependencies, 3D denotes 32 endsystem architecture, all execution times in seconds.

model from the previous chapter is taken as a basis, because there are not enough endsystems to resolve all conflicts arising during allocation. Therefore, the overall number of timeframes needed to complete all jobs is increased by more than 50% if all conflicts are resolved manually. On the other hand the experimental result shows a significant performance improvement with respect to runtime and achieved a speed-up of two magnitudes. The opposite effect can be witnessed when executing the second example scenario on a larger three-dimensional symmetric architecture with 32 endsystems as depicted in Figure 13. Here static compaction can be applied successfully and results in a loss of optimality of only 38%. Due to an increased number of different possible paths to be checked for optimality in the clouds, however, runtime can only be enhanced by about 25%. These effects will have to be analysed in future.

6.6 Summary

In this chapter we have demonstrated two different approaches to parallelise the scheduling problem in order to accelerate the computation of schedules. At first we exploited the incremental nature of the proposed SMT-based scheduling framework and checked different instances of the same problem for satisfiability given different upper limits for the number of timeframes available. We have been able to show that for the scenarios discussed in Chapter 5 we could significantly reduce execution times on an MPSoC emulating architecture to an extend that it could compete with a state-of-the-art CPLEX based scheduler. Therefore this technique can be considered for application on an MPSoC if memory and computational specifications permit the execution of large scheduling problems in parallel.
If the scenarios are enlarged, we have also investigated an approach to separate the logical dependencies and divide them into smaller problems. In order to do so we have applied a dominator-based approach which relies on the well-known technique to detect re-converging paths in a CFG. In this context we have created a setup which applies the Lengauer-Tarjan algorithm to the CFG representing the logical dependencies and formulated a strategy for an automatic division. We then have computed optimal solutions for each split which were finally merged into a global schedule under application of static compaction.
In order to implement static compaction sensibly under the avoidance of collisions on endsystems we have significantly enlarged the physical model and executed all logical scenarios on a symmetric three dimensional architecture consisting of 8 switches and 32 endsystems. We have shown that sacrificing optimality has a severe impact on the total execution time. Therefore it may be worthwhile to dispense an optimal solution if runtime is essential.

However, if we are willing to relax the optimality condition, it is necessary to compare the proposed scheduling framework to a scheduling software based on a state-of-the-art heuristic. Therefore we will introduce List Scheduling techniques in the next chapter and further evaluate the proposed scheduler in comparison to this concept.

7 Scheduling on Fault-Tolerant Architectures

7.1 Introduction

This chapter describes the fourth and final contribution of the thesis: An evaluation of our proposed optimal scheduling framework when comparing it to a state-of-the-art heuristic on a fault tolerant architecture. It is structured as follows:

The first subsection serves as an introduction into fault tolerance and presents common terms and concepts. This is followed by a demonstration of the proposed scheduling framework on a three-dimensional mesh which is subjected to randomly injected faults. In order to evaluate the performance, we will also use a popular heuristic called List Scheduling, which can be applied to schedule on-chip networks such as MPSoCs. We illustrate its functionality and we compare the gap in makespans of optimal solutions to those schedules obtained from heuristics. We will show that the overall number of timeframes required to perform all tasks can be significantly reduced when opting for an optimal scheduler. This effect is further amplified if the number of permanent failures is increased.

7.2 Introduction into Fault Tolerance Techniques

In this subsection we will introduce the basic terms and concepts of fault tolerance. We will follow [69] for notation and terminology. The key technique for handling failures is redundancy, which is also discussed. For more general information on fault tolerance in distributed systems we refer to [70].

Nowadays hardware is equipped with fault tolerance mechanisms and therefore the concepts are also applied to modern NoC and MPSoC such as the Phoenix NoC, a fault tolerant distributed on-chip architecture, which was presented for the first time in 2013 [71]. Reliability and availability have become increasingly important in today's computer dependent world. In many applications, where computers are used, malfunctions can be expensive or even disastrous, when we consider the telecommunication switching systems or the bank transaction systems for instance.

As embedded systems play a dominant role in safety-critical applications guaranteeing functionality and reliability of these architectures is vital and an indispensable part of every certification process. Therefore, fault-tolerant systems and protocols are required. They have the ability to tolerate faults by detecting failures, and isolate defect modules so that the rest of the system can operate correctly. Fault-tolerant mechanisms and techniques have also become of increasing interest to embedded systems. Four trends contribute to this:

1. Embedded systems are deployed under harsh conditions such as high temperatures over a wide range, dust, humidity and unstable power supply.
2. Embedded systems are operated without supervision of trained personnel.
3. Service costs are relative to hardware cost. Therefore, as components become cheaper, maintenance must not be expensive.
4. The most important aspect from our point of view is the significant increase of the size of modern NoC and MPSoC architectures. As systems become larger, there are more components that can fail. This means, to keep the reliability at an acceptable level, designs have to tolerate faults resulting from component failures.

When a system or module is designed, its behaviour is specified. If the observed behaviour differs from the specified behaviour, a so-called **failure** has occurred. A failure occurs because of an error caused by a **fault**. It is distinguished between **transient**, **intermittent** and **permanent** faults
Transient faults occur once and then disappear. If the operation is repeated, the fault goes away. A bird flying through the beam of a microwave transmitter may cause lost bits on some network. If the transmission times out and is retried, it will probably work the second time.
An intermittent fault occurs, vanishes and reappears randomly. A loose contact on a connector will often cause an intermittent fault. Intermittent faults are challenging because they are difficult to diagnose.
A permanent fault is one that continues to exist until the faulty component is repaired. Burnt-out chips, software bugs, and disk head crashes are examples of permanent faults.
In order to detect how serious a failure actually is a classification scheme was developed in 1993 [72]. Although originally designed for classical distributed systems (servers, networks) it can still be applied to modern MPSoC architectures. We will specify this scheme in Table 15.
If a system is fault tolerant, it can sustain its services even after the occurrence of failures. The key technique for masking faults is to use redundancy. Three different kinds are discussed in literature [73]: information redundancy, time redundancy and physical redundancy. Information redundancy means that extra bits are added to allow recovery from garbled bits. Time redundancy allows an action, i.e. sending a message, to be performed again.
We will focus on physical redundancy, which is a well-known technique for providing fault tolerance. It has been used in the design of fault-tolerant electronic

Type of failure	Description
Crash failure	A switch halts. An endsystem stops working.
Omission failure	A switch cannot handle messages. An endsystem can neither send nor receive messages.
Timing failure	A message does not travel between two nodes in the specified time interval.
Response failure	A switch does not respond after receiving a message. An endsystem cannot receive messages.
Arbitrary failure	Random messages are produced.

Table 15: Different types of failures.

circuits for years. Consider for instance the very popular Three Modular Redundancy (TMR) approach where each component is replicated three times and a voter device checks whether two or three inputs are the same. If this is true, the signal is assumed to be correct.
As the architectures we consider are consisting of a large number of switches, endsystems and different links between them, the system will not usually be used to full capacity. Therefore, on the one hand we are able to re-allocate jobs from faulty components to endsystems free from defect. On the other hand we can compensate for broken switches and links by re-routing the messages in the system. Our model considers all faults to be permanent and we assume all failures are identified correctly by the system. In the following subsection we will demonstrate how recovery from faults can be achieved by re-computing the scheduling problem.

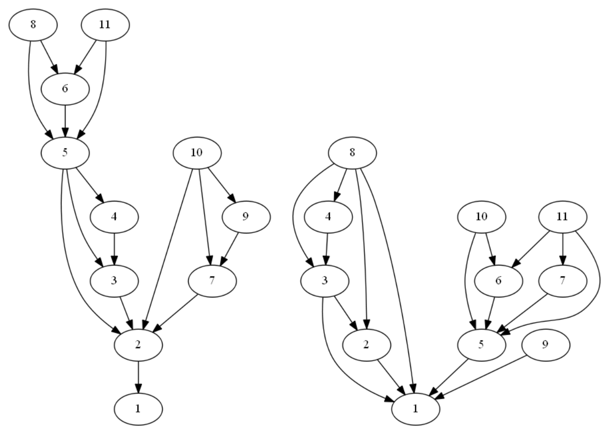
7.3 Re-scheduling after Failure

Due to the large number of computational units provided by an MPSoC architecture fault tolerance can be employed using redundancy techniques as described in the previous section. Usually the system is not used to full capacity and therefore jobs can be allocated to other endsystems if certain components become faulty. If an error is detected, the proposed optimal scheduler will re-schedule the modified system and compute a new optimal schedule providing the minimal makespan with respect to the new topology.
In an initial step an optimal schedule with respect to the makespan is computed considering the MPSoC is operating without errors and all components are faultless. In order to quantify the makespan we will again apply the concept of *timeframes*. Just as before all messages require one timeframe to pass from one node to another.

In this context a node may either be a switch or an endsystem.
In order to guarantee the optimality of the solution an incremental approach has been chosen where it is evaluated whether messages can be received within a given number of timeframes t. If this is not the case t is incremented by 1 and the reasoning process is repeated until the minimal value for t, denoted as t_{min} has been determined.
We assume that multiple combinations of the following permanent faults are can occur randomly:

1. Failure of an arbitrary node (i.e. switch or endsystem).
2. Bidirectional breakdown of a link.
3. Multiple combinations of both.

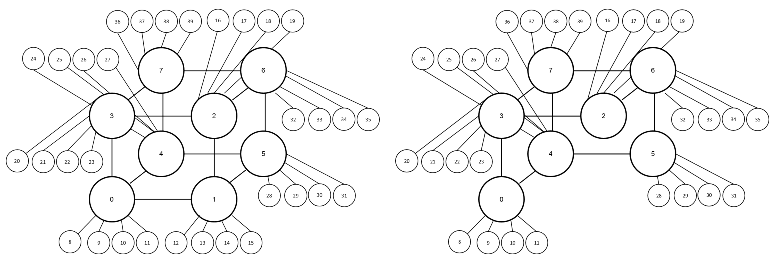
By assumption faults are detected correctly. The re-scheduling process can then be initiated using t_{min} as a starting value. By construction of the incremental approach described in Subsection 5.3 a solution requiring a lower number of timeframes cannot exist. This means the satisfiability checks for $t < t_{min}$ become obsolete which will significantly reduce runtime. As depicted in Table 16 the runtime to compute an optimal schedule for the problem is significantly reduced if components are faulty. For instance if a switch fails the number of possibilities to allocate jobs is immediately reduced by the number of endsystems attached to the faulty component. These two aspects lead to a significant reduction of runtime as on the one hand the problem becomes smaller and on the other hand the evaluation for $t < t_{\text{min}}$ becomes obsolete. Thus overall computation time can be significantly reduced enabling the application of the scheduler at runtime if all pre-computed communication schedules fail. This can prolong the lifetime of the system under consideration.
As the optimal scheduling problem is $\mathcal{NP}$-complete we cannot guarantee that we will be able to reduce the runtime of the scheduling process if the size of the problem is reduced. There we initially evaluate two medium-sized scenarios from the previous section illustrated in Figure 16. We assume that components fail randomly. The results are depicted in Table 16.
The jobs were optimally scheduled on the three-dimensional mesh found in Figure 17a obeying their logical dependencies depicted in 16. Firstly the system under consideration is supposed to perform faultlessly. Then the model is changed t to reflect the output of an error detection service. If an error is detected, we reallocate all jobs to endsystems free of fault and ensure that messages are only sent via links and switches still in operation. This ensures that the new solution is still optimal with respect to the components still in operation.



(a) First example (b) Second example

Figure 16: Logical models for examples in Table 16

Technically the process described can be illustrated by removing the faulty components from the physical model thus creating a new scenario. This process is illustrated in Figure 17.



(a) Physical model before failure (b) Physical model after failure

Figure 17: Transformation after permanent error on switch 1

7.4 List Scheduling

One of the main advantages of an optimal scheduler is that it always provides a schedule with the minimal makespan even if components in the system under consideration become faulty. In order to evaluate this effect, we will compare the performance of the proposed scheduling framework to one of the most popular heuristic approaches. When applying heuristics we will show that the makespan will increase significantly if the number of faulty components within the system is increased. Vice versa the makespan is reduced when applying an optimal scheduler.

Scenario	Total Nodes	Endsystems	Switches	Jobs	Messages	Lower Bound	Timeframes	Execution Time
no fault	40	32	8	11	16	7	21	45.3587
1 SW fault	35	28	7	11	16	21	21	13.6809
2 SW fault	30	24	6	11	16	21	21	7.6125
3 SW fault	25	20	5	11	16	21	21	3.8082
4 SW fault	20	16	4	11	16	21	21	1.4321
5 SW fault	15	12	3	11	16	21	23	9.7846
no fault	40	32	8	11	17	5	21	252.6438
1 SW fault	35	28	7	11	17	21	21	39.2585
2 SW fault	30	24	6	11	17	21	21	20.5533
3 SW fault	25	20	5	11	17	21	21	9.1286
4 SW fault	20	16	4	11	17	21	21	4.8643
1 ES fault	39	31	8	11	17	21	21	98.1221
2 ES fault	38	30	8	11	17	21	21	97.3426
3 ES fault	37	29	8	11	17	21	21	97.3621

Table 16: Re-scheduling at runtime considering random number of faulty endsystems and switches. Logical dependencies for the two scenarios are depicted in Figure 17.

We will focus on List Scheduling (LS), the dominant scheduling heuristic. LS heuristics use a sorted priority list, containing the jobs ready to be scheduled, while respecting the precedence constraints. A job is ready if all the predecessor tasks have finished executing and all the incoming messages are received.
LS generates the schedule by successively scheduling each job and message onto an endsystem. The start time in the schedule table is the earliest time when the resource is available for the respective job (or message). If more resources than jobs are available every job is assigned to the idle endsystem with the lowest index. Therefore, the allocation of jobs has a direct influence on communication cost.
Adapting this heuristic to our proposed time-discrete model the following adjustments are made:

- A priority list is obsolete as all jobs have the same priority. Therefore, we will simply list them in ascending order. This means for $i \neq i'$ job j_i is allocated before job $j_{i'}$ if $i < i'$ and vice versa.
- Jobs will be allocated to an endsystem if all incoming messages are received. If no messages have to be received (i.e. job is source) the job will be allocated immediately to the idle endsystems with lowest index.
- Once a job has been assigned to an endsystem this endsystem is no longer considered idle an thus not available for future allocations.

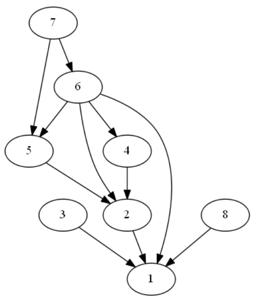


Figure 18: Logical dependencies

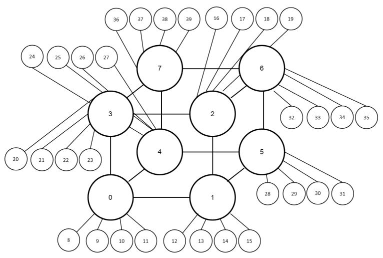


Figure 19: 3D Mesh

As an example we consider the logical model depicted in Figure 18 and execute it on the 3D-Mesh represented in Figure 19. Here we have to schedule 8 jobs $j_1 \cdots j_8$ onto 32 endsystems.
It is our goal to assign the job variables $j_1 \cdots j_8$ with the number of the endsystem the job is allocated to. We can determine the following:

$$j_3 = 8, j_7 = 9, j_8 = 10$$

because jobs $3, 7$ and 8 are source jobs and as such they are the only jobs that can be allocated in timeframe 1. Now the next jobs, which can be assigned are j_6 and j_5 because they do not have to wait for any other jobs to be scheduled. Opposed to this j_1 cannot be assigned because it still has to wait for j_2 which has not yet been allocated.
We continue in the proposed fashion and therefore assign

$$j_5 = 11, j_6 = 12.$$

Of all the jobs not yet allocated only j_4 does not depend on any other non-scheduled jobs. Therefore, it can be stated

$$j_4 = 13$$

and subsequently

$$j_2 = 14$$

and finally

$$j_1 = 15.$$

In conclusion the job variables are assigned according to LS as follows:

$$j_1 = 15, j_2 = 14, j_3 = 8, j_4 = 13, j_5 = 11, j_6 = 12, j_7 = 9, j_8 = 10. \qquad (23)$$

Messages are scheduled in the following way: Whenever a job is allocated it is checked whether it is a destination of a message. If this is the case, the messages that are received by this node are scheduled so that they are received in the earliest possible timeframe obeying the logical dependencies and the conflict-free traversal. If more than one message is received, the message with the smallest destination job is scheduled first. To illustrate this consider again the example above. We will identify the messages with respect to the original SMT problem sheet:

- $M_1 : j_2 \rightarrow j_1$
- $M_2 : j_3 \rightarrow j_1$
- $M_3 : j_4 \rightarrow j_2$
- $M_4 : j_5 \rightarrow j_2$
- $M_5 : j_6 \rightarrow j_1$
- $M_6 : j_6 \rightarrow j_2$
- $M_7 : j_6 \rightarrow j_4$
- $M_8 : j_6 \rightarrow j_5$
- $M_9 : j_7 \rightarrow j_5$
- $M_{10} : j_7 \rightarrow j_6$
- $M_{11} : j_8 \rightarrow j_1$

The three source jobs j_3, j_7 and j_8 can be allocated without consideration of any messages. When j_5 is allocated the following messages have to be considered:

- $M_9 : j_7 \rightarrow j_5$

- $M_8 : j_6 \rightarrow j_5$

M_8 cannot be scheduled because j_6 has not yet been allocated. M_9 however can be scheduled because it corresponding source and destination jobs have already been scheduled to $j_7 = 9$ and $j_5 = 11$. In order to travel from node 9 to node 11 switch 0 has to be passed. As no further dependencies have to be obeyed this can be scheduled between timeframes 1 and 3. We continue in the same way and allocate j_6. This means we can add

- $M_{10} : j_7 \rightarrow j_6,$
- $M_8 : j_6 \rightarrow j_5$

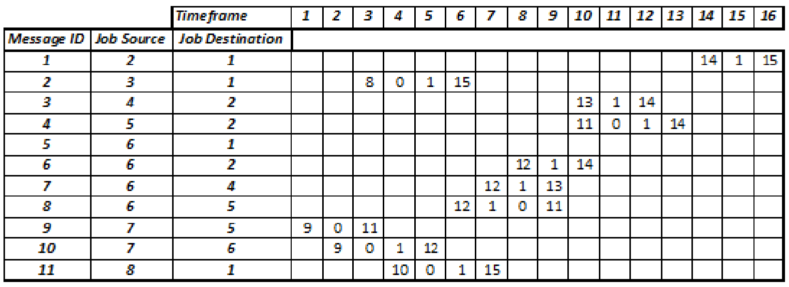
to the schedule. As M_{10} has to be scheduled before M_8 (logical dependencies) M_{10} starts in timeframe 2 and finishes in timeframe 5 as it has to pass two switches 0 and 1. Then M_8 is scheduled subsequently finishing in timeframe 9. In the next step j_4 is allocated an thus M_7 can also be scheduled. Then j_2 is allocated receiving messages

- $M_3 : j_4 \rightarrow j_2$
- $M_4 : j_5 \rightarrow j_2$
- $M_6 : j_6 \rightarrow j_2.$

At first M_3 is scheduled due to the lowest index of the sending job needing only three timeframes and finishing in timeframe 12. We continue with M_4 and M_6 before finally allocating j_1. Now the final remaining messages M_1, M_2 and M_{11} must be scheduled. Conveniently only the message to be scheduled first i.e. M_1 has to obey any dependencies and is scheduled following the arrival of M_4. M_2 and M_{11} are independent and are scheduled such that they finish in the earliest timeframe possible.
The proposed list-scheduling approach results in a schedule requiring 16 timeframes. The proposed scheduling framework finds a solution in 15 timeframes when allocating the jobs to endsystems as follows:

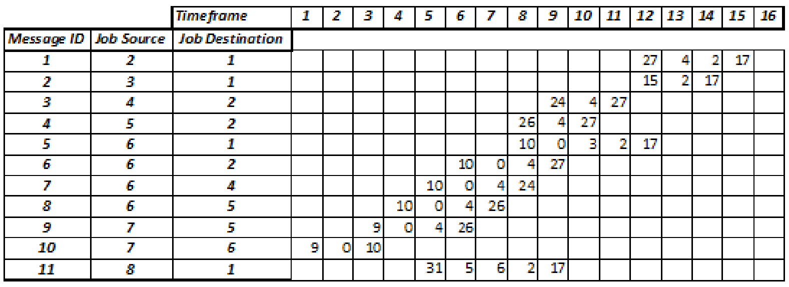
$$j_1 = 17, j_2 = 27, j_3 = 15, j_4 = 24, j_5 = 26, j_6 = 10, j_7 = 9, j_8 = 31. \qquad (24)$$

The different schedules arising from the assignments of the job variables in (23) and (24) are depicted in Figure 20. For every message the tables illustrate which endsystem or switch is visited in each timeframe.
The gap between the optimal solution and the schedule based on LS is one timeframe and thus almost negligible. However, as we will demonstrate in the next section the gap increases significantly if faults are injected into the 3D-mesh.



		Timeframe	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Message ID	Job Source	Job Destination																
1	2	1														14	1	15
2	3	1			8	0	1	15										
3	4	2										13	1	14				
4	5	2										11	0	1	14			
5	6	1																
6	6	2								12	1	14						
7	6	4							12	1	13							
8	6	5						12	1	0	11							
9	7	5	9	0	11													
10	7	6		9	0	1	12											
11	8	1				10	0	1	15									

(a) Schedule based on LS requires 16 timeframes.



		Timeframe	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Message ID	Job Source	Job Destination																
1	2	1												27	4	2	17	
2	3	1												15	2	17		
3	4	2									24	4	27					
4	5	2								26	4	27						
5	6	1								10	0	3	2	17				
6	6	2						10	0	4	27							
7	6	4					10	0	4	24								
8	6	5				10	0	4	26									
9	7	5			9	0	4	26										
10	7	6	9	0	10													
11	8	1					31	5	6	2	17							

(b) Optimal schedule requires 15 timeframes.

Figure 20: Different schedules for the example.

7.5 Results

So far we have seen that the proposed scheduling framework is performing better than state-of-the-art schedulers when we compare the computation of optimal solutions i.e. for minimal transmission times. However usually heuristic-based approaches are sufficient for most applications because they are often able to compute feasible solutions. Therefore, depending on the properties of the architecture under consideration, an optimal solution with respect to a minimal makespan may not be necessary, because all timing constraints can be met with a feasible solution which makes the computational expenditure for an optimal solution redundant. As we have seen with LS in the previous section heuristic approaches are very fast and opposed to the optimal scheduling problem not in the class of $\mathcal{NP}$-complete problems. Feasible solutions may however not be sufficient as their makespan might not meet certain deadline constraints. Therefore it is important to clearly distinguish between a feasible solution, which may be regarded as optimal because it fulfils all timing constraints, and an optimal schedule with the respect to a minimal makespan. The latter schedule guarantees that there cannot exist a schedule with a smaller makespan. There are good reasons to insist on an optimal solution with respect to the minimal makespan as we will outline in detail in this section. We will illustrate how the timeplans computed with a scheduler based on the LS-heuristic introduced in the previous section will differ from optimal solutions. We will show that this gap is actually growing significantly if the number of faulty components within the system under consideration is increased. This effect is further intensified if faults occur more frequently in the same area of the mesh (i.e. failure of neighbouring switches). An accumulation of faults in the same region of a distributed system is a realistic assumption and has for instance been described in a fault hypothesis for integrated architectures [74].

We will randomly declare components to be faulty. By assumption faults are detected correctly. The re-scheduling process is then be initiated using $t_{\min}$ as described in Section 7.2. In addition we compute a schedule applying an LS-based heuristical scheduler from Section 7.4. We investigate on the one hand how the total number of timeframes increases when the number of faults on the architecture under consideration is increased and also analyse how this is linked to connectivity of the components that become faulty. On the other hand we will show that we can significantly reduce the total number of timeframes required to complete the transmission of all messages if the proposed scheduling framework is deployed. Therefore, shorter deadlines can still be met even if large parts of the system fail. This will eventually prolong the lifetime of the systems under consideration considerably.

As before we have executed the experiments with respect to a symmetric, three-dimensional mesh consisting of eight switches each connected to two other switches and 4 endsystems. Hence in a scenario without faults we basically consider a cube of switches as depicted in Figure 19. For every fault that occurs we consider a new architecture and remodel the problem respectively just as we did in the example depicted in Figure 20.

In the following Table 17 demonstrates explicitly the faults we randomly considered. The first columns refer to the logical model executed on the platform. The final column describes the origin of the faults and their locations. Evaluation of the performance with respect to runtime and minimal makespan are provided in Table 18 and 19 using the short notation introduced in the second column of the following table:

Obviously the heuristic based approach significantly outperforms the proposed optimal scheduling framework when comparing computation times. This is due to the fact that LS is immediately allocates a job once a spare endsystem is detected. The first schedule acquired in this manner is then given as a result without checking for optimality. Therefore, other schedules with a smaller makespan may exist which is practically always the case unless the scenarios are tiny. This is also the most striking difference to the $\mathcal{NP}$-completeness of the optimal scheduling

Scenario	Number	Total Nodes	Endsystems	Switches	Jobs	Messages	Description of randomly injected faults
S1	1	40	32	8	6	8	no fault
S3	2	40	32	8	11	16	no fault
S3	2a	35	28	7	11	16	1 faulty switch
S3	2b	30	24	6	11	16	2 faulty switches (nn)
S5	3	40	32	8	11	17	no fault
S5	3a	35	28	7	11	17	1 faulty switch
S5	3b	30	24	6	11	17	1 faulty switch
S16	4	40	32	8	19	26	no fault
S16	4a	35	28	7	19	26	1 faulty switch
S16	4b	30	24	6	19	26	2 faulty neighboring switches
S16	4c	32	24	8	19	26	failure of every fourth ES
S16	4d	32	24	8	19	26	additionally links between switches 0/1 and 3/4
S16	4e	32	24	8	19	26	additionally links between switches 0/1, 2/3, 2,7, 3/4
S16	4f	29	21	8	19	26	failure of every third ES
S16	4g	29	21	8	19	26	additionally links between switches 0/1, 2/3
S16	4h	29	21	8	19	26	additionally links between switches 0/1, 2/3, 2,7, 3/4
S20	5	40	32	8	20	35	no fault
S20	5a	35	28	7	20	35	1 faulty switch
S20	5b	30	24	6	20	35	1 faulty switch
S20	5c	32	24	8	20	35	failure of every fourth ES
S20	5d	32	24	8	20	35	additionally links between switches 0/1, 3/4
S20	5e	32	24	8	20	35	additionally links between switches 0/1, 2/3, 2,7, 3/4
S20	5f	29	21	8	20	35	additionally links between switches 0/1, 2/3, 2,7, 3/4
S20	5g	29	21	8	20	35	failure of every third ES
S20	5h	29	21	8	20	35	additionally links between switches 0/1, 3/4
S17	6	40	32	8	24	41	no fault
S17	6a	40	32	8	24	41	ES 8-15 and switches 0,1
S17	6b	35	28	7	24	41	1 faulty switch
S17	6c	30	24	6	24	41	2 faulty neighboring switches
S17	6d	30	24	6	24	41	2 switches (nn)
S17	6e	32	24	8	24	41	failure of every fourth ES
S17	6f	32	24	8	24	41	additionally links between switches 0/1, 2,7
S17	6g	32	24	8	24	41	additionally links between switches 0/1, 2/3, 2,7, 3/4
S19	7	40	32	8	20	29	no fault
S19	7a	39	31	8	20	29	1 faulty ES
S19	7b	37	29	8	20	29	2 faulty ES
S19	7c	36	28	8	20	29	3 faulty ES
S19	7d	35	28	7	20	29	1 faulty switch
S19	7e	30	24	6	20	29	2 faulty neighbouring switches
S19	7f	30	24	6	20	29	2 faulty switches (nn)
S19	7g	25	20	5	20	29	3 faulty switches
S19	7h	32	24	8	20	29	failure of every fourth ES
S19	7i	32	24	8	20	29	additionally links between switches 2,7, 3/4
S19	7j	32	24	8	20	29	additionally links between switches 0/1, 2/3, 2,7, 3/4
S19	7k	29	21	8	20	29	failure of every third ES
S19	7l	29	21	8	20	29	additionally links between switches 0/1, 2/3,
S19	7m	29	21	8	20	29	additionally links between switches 0/1, 2/3, 2,7, 3/4

Table 17: Classification of random faults which occurred. (nn) denotes non-neighbouring switches, i.e. no link between them, in the three-dimensional mesh under consideration.

Number	**Jobs**	**Messages**	t_o **Timeframes YICES+**	t_h **Timeframes LS**	**Proportion**	**g**	**Runtime Y+ in s**	**Runtime Heuristic in s**	**Proportion**
1	6	8	11	11	1.00	0	0.68	0.01	136.00
2	11	16	18	25	1.39	7	45.36	0.03	1.814.35
2a	11	16	18	25	1.39	7	13.6809	0.04	361.93
2b	11	16	18	26	1.44	8	7.6125	0.03	254.85
2c	11	17	18	21	1.17	3	252.64	0.08	3.158.05
2d	11	17	18	21	1.17	3	39.2585	0.08	506.56
2e	11	17	18	23	1.28	5	20.5533	0.08	253.37
3	19	26	19	31	1.63	12	1321.91	0.84	1.565.32
3a	19	26	19	32	1.68	13	154.54	0.67	231.78
3b	19	26	19	32	1.68	13	74.56	0.79	94.58
3c	19	26	20	37	1.85	17	1028.48	0.65	1.582.27
3d	19	26	20	39	1.95	19	100.25	0.70	143.21
3e	19	26	20	39	1.95	19	96.70	0.62	155.97
3f	19	26	20	37	1.85	17	121.65	0.65	187.15
3g	19	26	22	45	2.05	23	1359.44	0.79	1.720.81
3h	19	26	22	51	2.32	29	86.50	0.68	127.21
4	20	35	21	39	1.86	18	6370.00	0.95	6.705.26
4a	20	35	21	39	1.86	18	1323.35	0.98	1.350.36
4b	20	35	21	40	1.90	19	410.68	0.85	483.15
4c	20	35	22	45	2.05	23	2025.33	1.01	2.005.28
4d	20	35	22	43	1.95	21	541.26	0.80	676.57
4e	20	35	23	48	2.09	25	2002.25	0.62	3.229.44
4f	20	35	25	44	1.76	19	2002.25	0.62	3.229.44
4g	20	35	25	44	1.76	19	3412.12	0.77	4.431.32
4h	20	35	26	52	2.00	26	9221.09	0.87	10.598.95

Table 18: Part 1: Comparing Runtime and Gap g between optimal and heuristic solution after injecting different faults into the systems

Number	**Jobs**	**Messages**	t_o **Timeframes YICES+**	t_h **Timeframes LS**	**Proportion**	**g**	**Runtime Y+ in s**	**Runtime Heuristic in s**	**Proportion**
5	24	41	33	47	1.42	24	14155.34	1.25	11.333.34
5a	24	41	33	43	1.30	10	14155.34	1.25	11.333.34
5b	24	41	33	48	1.45	15	1784.42	1.23	1.446.51
5c	24	41	34	47	1.38	13	4076.52	1.24	3.275.89
5e	24	41	34	42	1.24	8	4076.52	1.34	3.034.70
5f	24	41	36	51	1.42	15	18229.36	1.45	12.571.97
5g	24	41	38	63	1.66	25	6157.23	1.45	4.246.36
5h	24	41	40	79	1.98	39	13708.53	1.48	9.262.52
6	20	29	19	34	1.79	15	977.49	0.21	4.599.96
6a	20	29	19	34	1.79	15	364.0628	0.20	1.820.31
6b	20	29	19	34	1.79	15	361.2323	0.20	1.806.16
6c	20	29	19	34	1.79	15	316.9943	0.20	1.584.97
6d	20	29	19	34	1.79	15	204.48	0.16	1.315.81
6e	20	29	19	34	1.79	15	97.88	0.13	780.53
6f	20	29	19	34	1.79	15	97.88	0.13	780.53
6g	20	29	19	38	2.00	19	52.83	0.15	352.21
6h	20	29	22	43	1.95	21	824.17	0.21	3.924.61
6i	20	29	25	48	1.92	23	2373.00	0.27	8.788.89
6j	20	29	26	53	2.04	27	1204.38	0.27	4.460.67
6k	20	29	24	47	1.96	23	1145.84	0.25	4.583.37
6l	20	29	27	57	2.11	30	1187.20	0.25	4.748.80
6m	20	29	29	63	2.17	34	4158.24	0.25	16.632.95

Table 19: Part 2: Comparing Runtime and Gap g between optimal and heuristic solution after injecting different faults into the systems

problem. As the number of possible schedules grows exponentially with respect to the number of endsystems and links the expenditure of an optimal scheduler is simultaneously increased.
In order to evaluate the quality of the solution we will examine the gap g between the number of timeframes required in an optimal solution as opposed to the makespan of an LS schedule. We define g as

$$g := t_o - t_h, \tag{25}$$

where t_o denotes the number of timeframes of an optimal solution and t_h the number of timeframes needed in an LS schedule respectively.
We have again chosen the six largest benchmarks from Chapter 5 to demonstrate the performance of our proposed scheduling framework. We have injected different faults in order to examine the behaviour of the architecture under different failure hypotheses. In total we have evaluated 50 scenarios and have been able to establish a connection between the number of timeframes required and the number of faults injected: The quality of the optimal solution is improved the more faults are injected into the architecture.
Consider for instance example number 5: Here we witness how g is growing from 14 in the no failure case to 39 if we fail one in three endsystems a additionally provoke a further accumulation of link failures between the neighbouring switches $0, 1, 2$ and 4 g increases to 39. However we also recognize that applying LS can also narrow the gap - see $5a$ - because the failure may coincidentally force LS to allocate jobs more efficiently. In this scenario the solutions differ relatively between 24% in $5e$ and 98% in $5h$.
In all examples g increases in absolute and relative terms in the long run, therefore especially for architectures applied in safety-critical environments it may prove sensible to compute an optimal solution if stringent timing constraints have to be met. On the other hand LS is much faster than optimal scheduling which makes it an attractive alternative if global timing constraints are relaxed.

7.6 Summary

Due to the growing demand of embedded systems in safety critical areas fault tolerance is an important prerequisite for NoC and MPSoC architectures. In this chapter we have evaluated our proposed scheduling framework assuming the underlying architecture supports fault-tolerance through active redundancy and under the assumption the faults are correctly diagnosed.
We have shown that the proposed scheduling framework significantly outperforms a scheduler based on a heuristic when the makespan is compared. Although feasi-

ble solutions can be computed significantly faster using a heuristic approach the makespan of the schedule computed with the proposed scheduling framework is significantly shorter. Depending on the number of faults injected and the containment area the makespan obtained by LS can be 250 % of the optimal solution. Considering stringent global timing constraints this can become a problem because deadlines are not met. At worst this could result in an immediate shut-down of the entire component and overall failure.
Additionally we have also seen that computing an optimal schedule is accelerated if faults are injected into the system. This effect has two reasons: On the one hand the problem shrinks because faulty endsystems and broken switches decrease search space for an optimal schedule. On the other hand an adequate lower bound for the incremental approach has already been computed for the defect-free scenario, because by construction a solution requiring less timeframes must not exist even if components are deleted.
In conclusion it may be sensible to deploy the proposed scheduling framework on modern NoC and MPSoC architectures especially if they have to meet tight deadlines. Heuristics will never be able to guarantee optimal schedules with respect to a minimal makespan. Therefore, an optimal solution may increase the lifetime of a system because it can still meet its requirements even if components are subject to permanent faults. This can make the proposed scheduling framework especially attractive for embedded systems designed for long time services for instance in aeronautical applications.

8 Conclusion

This chapter concludes the thesis and discusses open problems and future work.

8.1 Summary

In this thesis we have evaluated how formal verification tools such as SAT- and SMT-solvers can be applied to $\mathcal{NP}$-complete scheduling problems on novel NoC and MPSoC architectures. For this purpose we have established a time discrete model, which allowed us to formulate the scheduling problem as an optimization problem.
We started our experiments for simple architectures and small problem sizes limiting the complexity by the constant allocation of jobs to endsystems. We introduced one-hot-encoding in order to transfer the optimization problem to a pseudo-Boolean optimisation problem. This enabled the use of MiniSat+, an efficient, SAT-based optimising tool, to compute optimal schedules with respect to a minimal makespan.
Hence we were able to deduce that our approach was on the one hand more efficient in computing optimal solutions than state-of-the-art MILP solvers and on the other hand significantly reduced the amount of memory needed to compute optimal solutions. Therefore, we elaborated the optimal scheduler enhancing its functionality by allowing an arbitrary allocation of jobs to endsystems. This significantly increased complexity because the search space for possible solutions was enlarged. Therefore, we dropped the concept of one-hot-encoding and also allowed integer variables to formulate the optimization problem. As a consequence we switched from SAT solving to SMT solving and deployed the state-of-the-art SMT solver YICES2 in order to compute optimal schedules. As optimization is not a feature of YICES2 we choose an incremental approach to compute the minimal number of timeframes need to complete all jobs by sending and receiving all messages between them.
Again we were able to show a reduction in runtime and memory consumption when comparing our SMT-based tool to state-of-the-art MILP scheduler. Also we were able to port the proposed framework onto an MPSoC emulating multicore architecture based on Raspberry Pi 2. However due to restricted computation and memory capacities the calculation of optimal schedules was decelerated by at least one magnitude. In order to compensate for this defect we investigated different strategies to parallelise the incremental approach formulated. As a result we could show that using different parallelisation techniques we could again at least match or even outperform state-of-the-art MILP solvers which cannot be executed on

MPSoCs themselves.
Being able to execute our scheduling framework on the architecture under investigation itself enables us to investigate whether we can perform online scheduling. As opposed to scheduling during the design process this increases the degree of flexibility to react to an unforeseen error occurring within the system. Faulty components may lead to a failure of the entire architecture, because no feasible schedule stored in a TRM may be applicable to mask the error. Online scheduling with our proposed scheduling framework will always find an optimal schedule if it exists within a given interval. Due to the complexity of this $\mathcal{NP}$-complete problem this may become very time consuming.
However, under certain circumstances feasible solutions are sufficient. Therefore, we also investigated how the performance of the scheduling framework can be enhanced with respect to computation time, if we relax the requirement on the optimality of the makespan. We evaluated an approach dividing the problem based on the dominator tree and showed how we could merge locally optimal solutions into an global feasible solutions and analysed the trade-off between the sacrifice of optimality and the reduction of runtime.
Finally we evaluated the proposed scheduling framework to a scheduler based on the popular LS heuristic. Obviously LS scheduling is much faster in producing feasible schedules. But as the solutions are not optimal they may no meet deadline requirements. This effect is amplified if we consider fault tolerant architectures and increase the number of faulty components. We were able to show that even though multiple errors occurred the makespan computed under failure was still close to the minimum of the original schedule when the proposed optimal scheduling framework was applied. Opposed to this LS heuristics computed significantly larger makespans increasing the probability to fail deadline requirements.
It can be concluded that the application of formal verification tools such as SAT and SMT can be a sensible enhancement for scheduling jobs to novel NoC and MPSoC architectures especially if they are used in safety critical embedded systems which require a high degree of reliability and may have to be re-scheduled at runtime. Although computing optimal schedules is $\mathcal{NP}$-complete the use of the proposed scheduling framework computes results within reasonable runtime for complex problems especially when comparing it to state-of-the-art MILP schedulers.

8.2 Open Problems and Future Work

Scheduling will remain a prominent research topic. Special emphasis will be given to dynamic, online scheduling as safety critical embedded systems will be even more dominant in future.

We can extend our research by investigating more sophisticated ways partition the scheduling problem to further parallelise the computation of schedules exploiting the large number of endsystems in modern MPSoC architectures. Also it can be considered to evaluate a portfolio scheduler which employs SAT, SMT and heuristic based approaches in parallel to compute feasible schedules within a given number of timeframes. It terminates if either SAT or SMT returns an optimal solution or a heuristic based method found a schedule within less than a given number of timeframes.
Furthermore our work can also be continued for instance by investigating further optimization criteria such as the generation of resource-efficient schedules by minimising the energy consumption of the architecture under consideration or by extending life expectancy of the systems by equally balancing jobs to endsystems. Another interesting perspective would be to extend the presented approach by the use of different criticality levels. One could image a system configuration comprising of several functions that has different levels of criticality (e.g. safety-critical functions and non safety-critical functions) to which a two step optimisation can be applied which firstly emphasises on safety critical functions before allocating non-safety-critical jobs to idle endsystems. The approach could be adapted to prefer tasks or messages that are safety-critical in order to generate a schedule that is optimal with respect to criticality. Calculating schedules that guarantee quantifiable safety-properties are conceivable as well.

A Appendix

A.1 List of Figures

List of Figures

A.2 List of Tables

List of Tables

A.3 List of Abbreviations

ATPG	Automatic Test Pattern Generation
BCP	Boolean Constrain Propagation
BFS	Breadth First Search
CAV	Computer Aided Verification
CDCL	Conflict Directed Clause Learning algorithms
CFG	Control Flow Graph
CNF	Conjunctive Normal Form
CTL	Computational Tree Logic
DAG	Directed Acyclic Graph
DFS	Depth First Search
EDA	Electronic Design Automation
LP	Linear Programming
LS	List Scheduling
MILP	Mixed Integer Linear Programming
NP	nondeterministic, polynomial time
NoC	Network-on-a-Chip
MPSoC	Multi-Processor-System-on-a-Chip
SAT	Boolean Satisfiability
SMT	Satisfiability Modulo Theories
SoC	System-on-a-Chip
TMR	Three Modular Redundancy
TRM	Trusted Resource Manager
TSP	Travelling Salesman Problem
UNSAT	unsatisfiable
VLSI	Very Large Scale Integration

A.4 References

References

[1] C. Schöler, R. Krenz-Bååth, and R. Obermaisser, “A Novel Formal Verification Framework for Future MPSoC Architectures,” in *Proc. on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN/ETS) Workshop, ISBN*, 2015, pp. 48–51.

[2] C. Schöler, R. Krenz-Bååth, A. Murshed, and R. Obermaisser, “Optimal SAT-based scheduler for time-triggered networks-on-a-chip,” pp. 1–6, 2015.

[3] ——, “Optimal SAT-based scheduler for Time-Triggered Networks-on-a-Chip,” in *11th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2016, pp. 1–8.

[4] C. Schöler, R. Krenz-Bååth, and R. Obermaisser, “A Dominator-Based Partitioning for Efficient Scheduling in Time-Triggered NoCs,” in *Proceedings of the Work in progress Session held in connection with DSD 2016*. Johannes Kepler University Linz, 2016, pp. 1–2.

[5] C. Schöler, “Novel Scheduling Strategies for future NoC and MPSoC Architectures.” in *Proceedings of 24th IFIP/IEEE International Conference on Very Large Scale Integration*. PhD Forum, 2016.

[6] H. Kopetz and R. Obermaisser, “Temporal composability,” *Computing & Control Engineering Journal*, vol. 13, pp. 156–162, Aug. 2002.

[7] R. Obermaisser, *Time-triggered communication*. Taylor & Francis, 2011.

[8] K. Singh, M. Alam, and S. Sharma, “A survey of static scheduling algorithm for distributed computing system,” *International Journal of Computer Applications*, 2015.

[9] H. Kopetz, *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Springer, 2011.

[10] H. Kopetz, R. Obermaisser, C. El Salloum, and B. Huber, “Automotive software development for a multi-core system-on-a-chip,” in *Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems*. IEEE Computer Society, 2007, p. 2.

[11] M. Schoeberl, "A time-triggered network-on-chip," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on.* IEEE, 2007, pp. 377–382.

[12] "Time-triggered protocol TTP/C – High-Level Specification Document Protocol Version 1.1," TTTech, Tech. Rep., 2003.

[13] *FlexRay Communications System Protocol Specification Version 2.1*, FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG., May 2005.

[14] A. I. T. Company, "White paper: SAE AS6802 Deterministic Ethernet Network Solution," Tech. Rep., Mar. 2011.

[15] R. Obermaisser and C. Paukovits, "A cross-domain multiprocessor system-on-a-chip for embedded real-time systems," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 548–567, Nov 2010.

[16] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 306–311. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837353

[17] W. Wolf, "The future of multiprocessor systems-on-chips," in *Design Automation Conference, 2004. Proceedings. 41st.* IEEE, 2004, pp. 681–685.

[18] L. Torres, P. Benoit, G. Sassatelli, M. Robert, F. Clermidy, and D. Puschini, "An introduction to multi-core system on chip–trends and challenges," in *Multiprocessor System-on-Chip.* Springer, 2011, pp. 1–21.

[19] E. Fernandez-Alonso, D. Castells-Rufas, J. Joven, and J. Carrabina, "Survey of noc and programming models proposals for mpsoc," *International Journal of Computer Science Issues*, vol. 9, no. 2, pp. 22–32, 2012.

[20] H. Kopetz, *Real-time systems – Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

[21] R. Obermaisser and D. Weber, "Architectures for mixed-criticality systems based on networked multi-core chips," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Sept 2014, pp. 1–10.

[22] S. Borkar and A. A. Chien, “The future of microprocessors,” *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, 2011.

[23] A. Jerraya and W. Wolf, *Multiprocessor systems-on-chips*. Elsevier, 2004.

[24] J. K. Lenstra and A. Kan, “Complexity of vehicle routing and scheduling problems,” *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[25] W. Steiner and B. Dutertre, “SMT-Based formal verification of a TTEthernet synchronization function,” in *Formal Methods for Industrial Critical Systems*. Springer, 2010, pp. 148–163.

[26] H. Kopetz, “Time-triggered real-time computing,” *IFAC Proceedings Volumes*, no. 1, 2002.

[27] R. Obermaisser and O. Höftberger, “Fault containment in a reconfigurable multi-processor system-on-a-chip,” in *2011 IEEE International Symposium on Industrial Electronics*. IEEE, 2011, pp. 1561–1568.

[28] IBM, “Cplex Optimization Studio,” *URL: http://www-01. ibm. com/software/commerce/optimization/cplex-optimizer*, 2014.

[29] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 1971, pp. 151–158.

[30] L. Zhang and S. Malik, “Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications,” in *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*. IEEE Computer Society, 2003, p. 10880.

[31] Y. Hamadi, S. Jabbour, and L. Sais, “ManySAT: a parallel SAT solver,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 6, pp. 245–262, 2009.

[32] L. De Moura and N. Bjørner, “Satisfiability modulo theories: introduction and applications,” *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[33] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, “The mathSAT5 SMT solver,” in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2013, pp. 93–107.

[34] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE, 2010, pp. 375–384.

[35] F. Pozo, G. Rodriguez-Navas, H. Hansson, and W. Steiner, "SMT-based synthesis of TTEthernet schedules: A performance study," in *Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on*. IEEE, 2015, pp. 1–4.

[36] B. Barney *et al.*, *Introduction to parallel computing*, [Online; accessed 16-January-2017].

[37] M. Wolf, *High-performance embedded computing: applications in cyber-physical systems and mobile computing*. Newnes, 2014.

[38] K. Asanovic, Bodik *et al.*, "The landscape of parallel computing research: A view from Berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[39] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and applications of satisfiability testing*. Springer, 2004, pp. 502–518.

[40] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Theory and Applications of Satisfiability Testing*. Springer, 2005, pp. 61–75.

[41] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: a Parallel SAT Solver." *JSAT*, vol. 6, no. 4, pp. 245–262, 2009.

[42] S. Hölldobler, N. Manthey, V. H. Nguyen, J. Stecklina, and P. Steinke, "A Short Overview on Modern Parallel SAT-Solvers," in *Proceedings of the International Conference on Advanced Computer Science and Information Systems*, 2011, pp. 201–206.

[43] A. V. Aho and J. D. Ullman, *Foundations of computer science*. Computer Science Press, Inc., 1992.

[44] R. E. Simpson, *Introductory electronics for scientists and engineers*. Allyn & Bacon, 1974.

[45] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[46] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

[47] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "The traveling salesman problem (2006)," ISBN 0-691-12993-2, Tech. Rep.

[48] J. K. Lenstra and A. R. Kan, "Some simple applications of the travelling salesman problem," *Journal of the Operational Research Society*, vol. 26, no. 4, pp. 717–733, 1975.

[49] L. Zhang and S. Malik, "The quest for efficient Boolean Satisfiability Solvers," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 17–36.

[50] M. Ganai and A. Gupta, *SAT-based scalable formal verification solutions*. Springer, 2007.

[51] B. Dutertre and L. De Moura, "The Yices SMT solver," *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, vol. 2, no. 2, 2006.

[52] B. Dutertre, "Yices 2.2," in *Computer-Aided Verification (CAV'2014)*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, July 2014, pp. 737–744.

[53] C. W. Barrett, D. L. Dill, and A. Stump, "Checking satisfiability of first-order formulas by incremental translation to sat," in *Computer Aided Verification*. Springer, 2002, pp. 236–249.

[54] C. Schöler, R. Krenz-Baath, and R. Obermaisser, "A Novel Formal Verification Framework for Future MPSoC Architectures," in *Proc. on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN/ETS) Workshop*, 2015, pp. 48–51.

[55] N. Eén and N. Sorensson, "Translating pseudo-Boolean constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.

[56] J. Leskovec, "Stanford Network Analysis Package(SNAP)." http://snap.stanford.edu/, [Online; accessed 16-November-2015].

[57] B. Dutertre, "Yices 2.2," in *International Conference on Computer Aided Verification*. Springer, 2014, pp. 737–744.

[58] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[59] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*. ios press, 2009, vol. 185.

[60] A. Fellner, P. Fontaine, G. Hofferek, and B. W. Paleo, "NP-completeness of small conflict set generation for congruence closure," p. 8, 2015.

[61] B. Jobstmann and K. Leino, "Verification, Model Checking, and Abstract Interpretation: 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings," 2015. [Online]. Available: https://books.google.de/books?id=GYxNCwAAQBAJ

[62] D. R. Cok, D. Déharbe, and T. Weber, "The 2016 SMT Competition," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 207–242, 2016.

[63] C. Barrett, P. Fontaine, and C. Tinelli, "The SMT-LIB Version 2.5," 2010.

[64] W. Blair and N. Ghalili, "When Z3 Met Yices."

[65] R. T. Prosser, "Applications of boolean matrices to the analysis of flow diagrams," in *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. ACM, 1959, pp. 133–138.

[66] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 1, no. 1, pp. 121–141, 1979.

[67] R. Krenz-Bååth, A. Glowatz, and J. Schloeffel, "Computation and application of absolute dominators in industrial designs," in *Test Symposium, 2007. ETS'07. 12th IEEE European*. IEEE, 2007, pp. 137–144.

[68] L. Georgiadis, R. E. Tarjan, and R. F. F. Werneck, "Finding dominators in practice." *J. Graph Algorithms Appl.*, vol. 10, no. 1, pp. 69–94, 2006.

[69] H. Kopetz and P. Verissimo, "Real time and dependability concepts," in *Distributed systems (2nd Ed.)*. ACM Press/Addison-Wesley Publishing Co., 1993, pp. 411–446.

[70] P. Jalote, *Fault tolerance in distributed systems*. Prentice-Hall, Inc., 1994.

[71] C. Marcon, A. Amory, T. Webber, T. Volpato, and L. B. Poehls, “Phoenix NoC: A distributed fault tolerant architecture,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 2013, pp. 7–12.

[72] V. Hadzilacos and S. Toueg, “Distributed systems,” *MULLENDER, S.(Ed.)*, vol. 2, pp. 97–145, 1993.

[73] B. W. Johnson, “An introduction to the design and analysis of fault-tolerant systems,” pp. 1–84, 1995.

[74] H. Kopetz, C. El Salloum, B. Huber, R. Obermaisser, and C. Paukovits, “Composability in the time-triggered system-on-chip architecture,” in *2008 IEEE international SOC conference*. IEEE, 2008, pp. 87–90.