TUM School of Computation, Information and Technology
Technische Universität München

Protection Switching for Efficient Fail-Operational and Hard Real-Time Communication in Network on Chip

Max Wenzel Niklas Koenen

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:

Prof. Dr.-Ing. Ralf Brederlow

Prüfer der Dissertation:

1. Prof. Dr. sc. techn. Andreas Herkersdorf
2. Prof. Dr.-Ing. Georg Sigl

Die Dissertation wurde am 03.02.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 28.07.2023 angenommen.

Acknowledgements

This thesis is the result of my research at the Chair of Integrated Systems (LIS) at the Technical University of Munich. Many people contributed to it one way or another for which I am immensely grateful.

First and foremost I want to thank my "Doktorvater" Prof. Dr. Andreas Herkersdorf for the opportunity of pursuing my PhD under his guidance, his valuable and constructive feedback on my research in general and my publications in particular, the productive discussions, the opportunity to pursue my own ideas, the chance to attend multiple conferences to broaden my horizon, and generally the enjoyable atmosphere at the chair.

I also thank Prof. Dr. Georg Sigl for acting as second examiner on the doctoral examination commission of this thesis.

Furthermore, I want to thank Dr. Thomas Wild for supporting me in many ways, the informal discussions in the hallway, kitchen, office, or train, for going out of his way to free up some time when I had questions, and for his invaluable feedback on many written pages.

Special thanks go to Dr. Nguyen Anh Vu Doan for the great collaboration on multiple topics, his valuable insights, for his contribution to making the chair a fun place to work, the occasional beer or whiskey late at night, and for verbally kicking my butt every now and then. I want to thank Dr. Philipp Wagner for encouraging me to pursue a PhD at LIS and for helping me a great deal in many ways during my early years at the chair. Furthermore, I want to thank Michael Vonbun for his support over the years, the many discussions on NoCs, two fantastic conferences, and, of course, "tikznoc".

Without my colleagues at the chair none of this work would have been much fun. Thank you all for the many discussions—both technical and totally nonsensical—the coffee meetings, the beers and music late at night, and, of course, Tokami Tuesday and Dello Chef Friday. Special thanks also to the administrative staff at LIS that keep the chair running: Doris, Gabi, and particularly Verena (sorry for the various administrative nightmares I dropped on you over the years).

I also thank my students that I supervised over the years and whose work contributed to this thesis and, particularly, the implementation of the ARAMiS II demonstrator system.

Last but not least I want to thank my friends and family that accompanied me on my way. My parents who supported me over all these years and never gave up on me. And my friends who helped me keep my sanity during times of hardship—be it by means of metal music, a bottle of "Black Rock", or *just* an open ear—and who dragged me to a beergarden or a bar when they knew I needed a break.

To all of you: thank you!

Abstract

Multi- and even Manycore Processor System on Chips (MPSoCs) are becoming increasingly prevalent in embedded systems. However, despite their technical and economic advantages they are only hesitantly adopted in safety-critical embedded application domains such as avionics and automotive. Safety-critical applications typically have fault-tolerance and (hard) real-time requirements which are difficult to guarantee on an MPSoC that runs multiple applications. A key issue is the potential interference of different applications on shared resources such as memory or the on-chip interconnect. Isolation of different applications is especially challenging in mixed-critical systems that run both safety-critical and Best Effort (BE) applications.

This thesis contributes to the ongoing research of safely using MPSoCs in safety- and mixed-critical systems by developing both the concept for and architecture of a Network on Chip (NoC) that provides both fault-tolerance and hard real-time guarantees to safety-critical communication while also implementing isolation between different critical traffic streams and critical and BE communication. The basis is a hybrid NoC that uses Time-Division Multiplexing (TDM) for critical, and packet switching for BE traffic. Fault-tolerance of the critical communication is implemented by adopting protection switching to NoCs. Two disjoint paths are reserved for each critical communication channel. In case one path is affected by a fault, the other path is still fully operational. Three different protection switching versions are compared—1:n, 1:1, and 1+1 protection—and their respective advantages and disadvantages are evaluated and discussed. The proposed NoC architecture is implemented in hardware on an FPGA and the synthesis results are compared to related work. A comparison shows that the hybrid NoC router with protection switching uses up to 48% fewer LUTs and 46% fewer registers than a state-of-the-art packet switched router that provides throughput guarantees (but not fault tolerance) in a mixed-critical system. Furthermore, two systems using the hybrid NoC are implemented: an evaluation system with 8x8 nodes to evaluate the performance of the NoC for different traffic scenarios, and a demonstrator system with 4x4 nodes to showcase the capabilities of protection switching in NoC with an interactive system.

In addition to the concept and architecture of the hybrid NoC with protection switching, mapping strategies that consider protection switching and—for a given set of critical applications—increase the available bandwidth for the BE applications in order to use the NoC resources most efficiently and maximize the overall achievable network utilization are developed and evaluated. The different mapping strategies are evaluated in hardware with the evaluation system implemented on FPGA. The results show that evenly spreading out the critical applications across the entire system and evenly spreading out the critical traffic across all links is most beneficial for the BE communication.

It is formally proven that—between the three protection switching versions compared—1+1 protection has the lowest worst-case latency. Additionally, 1+1 protection requires a lower hardware overhead than both 1:n and 1:1 protection. It is shown that 1+1 protection has a worse adverse effect on the BE traffic performance than both 1:n and 1:1 protection have but that this effect can be mitigated and often even be nullified by good mapping strategies. In comparison to related work even 1+1 protection achieves very competitive traffic injection rates of up to $\sim$ 23%–31% in an 8x8 NoC, depending on the traffic scenario. It is concluded, that 1+1 protection is the most suited protection switching version to be used in mixed-critical MPSoCs.

To the best of my knowledge, the work presented in this thesis is the first one that considers a NoC that provides both fault-tolerance and hard real-time communication to critical applications in a mixed-critical MPSoC with a holistic approach.

Contents

List of Figures

List of Tables

Acronyms

ADAS	Advanced Driver-Assistance System.
BE	Best Effort.
BIST	Built-In Self-Test.
BRAM	Block RAM.
CPU	Central Processing Unit.
DI	Debug Interconnect.
DMA	Direct Memory Access.
DSP	Digital Signal Processor.
FDM	Fault Detection Module.
FG	Flit Generator.
FIFO	First In, First Out (buffer).
FIM	Fault Injection Module.
FMEA	Failure Mode and Effects Analysis.
FPGA	Field-Programmable Gate Array.
FSM	Finite State Machine.
GALS	Globally Asynchronous Locally Synchronous.
GS	Guaranteed Service.
GT	Guaranteed Throughput.
GUI	Graphical User Interface.
HDL	Hardware Description Language.
IRQ	Interrupt Request.
ISR	Interrupt Service Routine.
LFSR	Linear-Feedback Shift Register.
LSB	Least Significant Bit.
LUT	Look-Up Table.
MPSoC	Manycore Processor System on Chip.
MSB	Most Significant Bit.

NCM	NoC Control Module.
NI	Network Interface.
NoC	Network on Chip.
NSGA	Non-dominated Sorting Genetic Algorithm.
OpTiMSoC	Open Tiled Manycore System-on-Chip.
OSD	Open SoC Debug.
QoS	Quality of Service.
RNG	Random Number Generator.
SDM	Spatial-Division Multiplexing.
SET	Single Event Transient.
SEU	Single Event Upset.
SoC	System on Chip.
SSE	Solution Space Exploration.
TDM	Time-Division Multiplexing.
TGM	Traffic Generator Module.
TMR	Triple Modular Redundancy.
UART	Universal Asynchronous Receiver Transmitter.
USB	Universal Serial Bus.
VC	Virtual Channel.

1 Introduction

The year is 2025. You are on a plane, on your way to a conference in Copenhagen, Denmark. The flight is a bit rough. Nothing too serious. For the third time in a row the screens of the in-flight entertainment system go black. You sigh and take out your laptop to review your slides. After making some minor changes the program freezes. "Have you tried turning it off an on again?" says the man next to you. At the same time, cosmic rays cause a bit-flip in the flight computer calculating the plane's altitude. Nothing happens.
The plane lands safely. You board the fully automated metro into the city and decide to listen to some music. At some point the music breaks up. Bad connection. "Not my day", you think.
After a smooth ride you exit the train and call a taxi to bring you to your hotel. One of the new taxis. Fully automated. It starts to rain and the visibility is getting worse. A fast cyclist approaches from the right. The car breaks just in time. You curse at the reckless cyclist but the car safely brings you to your destination without anyone being harmed.

We live in a fast-paced world that is quickly changing, surrounded by a multitude of embedded systems doing their work. Most of the time we do not even notice them. Unless something goes wrong, in which case we are annoyed. Or harmed. Or dead.

One of the fastest changing industries, regarding the use of electronics, is the automotive industry. Over the last decades, the amount of electronic devices in cars has drastically increased and is expected to account for $\sim 50\%$ of the total cost of a new car in the year 2030—as shown in Figure 1.1 [1]. This trend is mostly driven by the increasing amount of Advanced Driver-Assistance System (ADAS) in today's and future cars and will be pushed further by the current motion towards autonomous driving. It is expected that by the year 2030 more than 80 million cars will have adopted at least Level 1 autonomous driving, which refers to driver support systems such as adaptive cruise control or lane-following assistance (cf. Figure 1.2) [2].

While a failure of an assistance system is merely an inconvenience for the driver—who must always be able to take over control—a failure in a system that is currently in full control of the car (i.e., Level 3 and above) can have dire consequences. Just as the flight-computer, or other essential systems of modern airplanes or autonomous trains, these systems must be hardened to tolerate faults—be they originated in hardware or software—and protected against malicious intent. These requirements are not just "arbitrarily" set by researchers, politicians, or the industry, but are requested by consumers themselves. In a study in Europe from 2015 most participants expected "maximal safety" from future cars (95.4% "fully" or "mostly" agreed), more than anything else ("maximal safety" was followed by "affordable mobility" with 90.5%, "comfort" with 89.9%, and

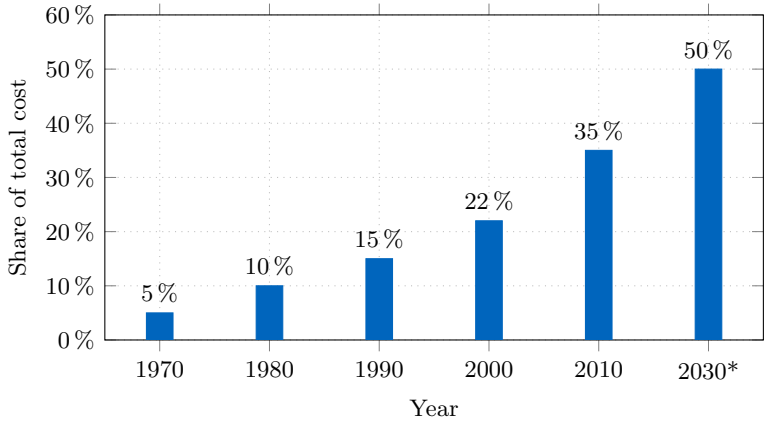


Figure 1.1: Automotive electronics cost as a percentage of total car cost worldwide from 1970 to 2030 (* forecast) [1]

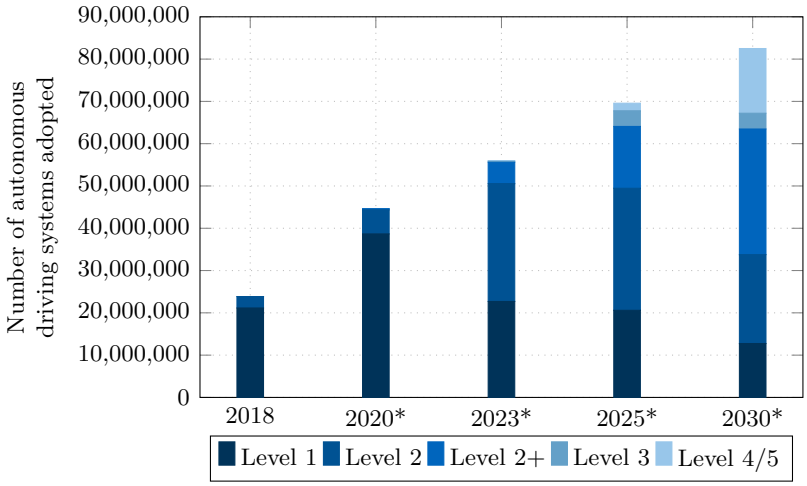


Figure 1.2: Global market size of ADAS & autonomous driving systems (* forecast) [2]

"environmental sustainability" with 88.1%) [4]. The consequence is that future cars will not only require more computational power in general but that they will also contain more safety-critical systems which must not fail or, at the very least, must handle an occurring fault in a way that ensures the safe halt of the vehicle.

For decades, Moore's law [5] was the main driver of increased computing performance with the number of transistors per chip area doubling roughly every 18 months. In combination with Dennard scaling [6] this meant an exponential growth of both overall performance and performance per Watt and enabled ever increasing processor clock speeds. Around the year 2005, however, Dennard scaling broke down and it was no longer feasible to further increase the clock speed of processors for mainstream computing. Instead, performance increase was achieved by moving towards multicore systems, which had a fundamental impact on the way software is developed in order to benefit from

the performance gains [7]. Not all applications can utilize the additional computing resources equally well and the possible speedup is ultimately limited by Amdahl's law [8]. Furthermore, the failure of Dennard scaling lead to the occurrence of "Dark Silicon", meaning not all transistors of a chip can simultaneously be powered at operating voltage for a given thermal design power constraint [9]. One way this was addressed was by going from homogeneous multicore systems to heterogeneous ones. By adding different kinds of processing elements and specialized hardware accelerators, applications can be executed more efficiently but, again, this affects how software must be implemented [10]. One recent example for such a system in mainstream computing is Apple's M1 System on Chip (SoC) which provides four high-performance cores and four high-efficiency cores [11].

A similar trend as the one described in mainstream computing has occurred in embedded systems—albeit slightly delayed—and lead to a transition from "simple" SoCs to Manycore Processor System on Chips (MPSoCs) that house tens and, in the near future, potentially up to hundreds of individual processing elements. Such MPSoCs offer great performance and can meet the ever increasing need for computational power in the automotive industry, as described above. Furthermore, they allow to implement different applications on a single device which helps to limit the hardware cost and power consumption of the system. However, using MPSoCs in safety-critical environments is problematic for various reasons. The question of *if* and *how* MPSoCs could be utilized in safety-critical applications in the automotive, railway, avionics, and industry domain lead to the ARAMiS research project [12]. The work presented in this thesis is supported by the follow-up project ARAMiS II.

A safety-critical system, as the name suggests, must ideally be safe to operate at any given time no matter the current state or input of the system. What exactly being *safe* to operate means typically depends on the environment where a system is deployed. At the very least it means that significant harm to people or the environment is prevented by any means necessary (or reasonable). This means that both hardware and software of such systems must be hardened against faults. Furthermore, such systems must be extensively tested in order to make sure that invalid states either never occur or are handled appropriately (basically, making them valid states). Although guidelines for the implementation of these systems exists—such as the ISO 26262 regulating functional safety for road vehicles [13]—this is a tedious task even for small systems. Because of this, the typical present approach of implementing such systems is to keep them relatively simple and static—e.g., by avoiding dynamic memory allocation or task mapping—and completely isolated from other applications, which also means using dedicated SoCs.

When multiple safety-critical applications are implemented on a single MPSoC there is a new problem: the potential unintended interaction or interference between different applications, e.g., because they compete for access to certain resources such as a memory interface to an external DDR memory. Even if all applications have their dedicated address space, competing access can lead to contention which causes congestion and changes the timing behavior of the applications. Since the potential interference of all applications must be analyzed and tested this means that verification and testing becomes much more complex. The development becomes evermore challenging with

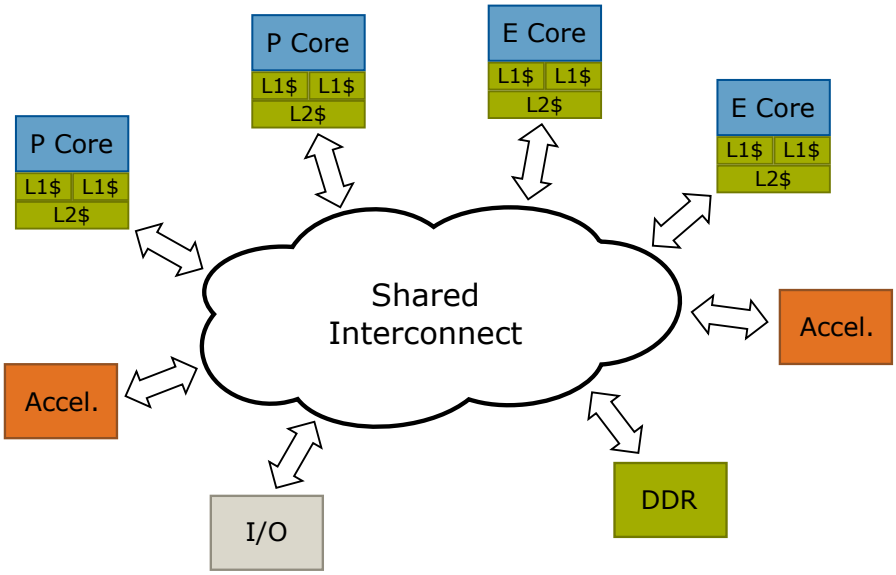


Figure 1.3: Different parts of an MPSoC communicating via a shared interconnect

growing number of applications and, even more so, if non-critical applications are also implemented on the same system resulting in a mixed-critical MPSoC.

1.1 Problem Statement

Ideally, the developers of each individual applications—safety-critical or not—should be able to develop their application completely independent from the others. This means that measures must be taken to implement appropriate isolation between different applications and ensure that each application has access to the resources it needs (or is assigned to). Regarding the processing resources it seems reasonable to implement isolation by statically assigning each application—or at least the safety-critical ones—to a fixed set of cores and/or accelerators. This approach will be assumed throughout this thesis unless stated otherwise. A possible way of implementing isolation in I/O interfaces is, e.g., I/O virtualization. A similar approach is possible for an interface to an external DDR memory. Another approach for a limited amount of applications would be to use multiple such interfaces, one exclusive for each safety-critical application. One of the open challenges, however, is isolation in the shared on-chip interconnect of the MPSoC (cf. Figure 1.3).

For decades, a single shared bus was used as a simple and convenient way of connecting the different components of a computing system to the Central Processing Unit (CPU). Similarly, this approach is commonly used in SoCs which contain a limited amount of processing and other elements. The downside of a single shared bus, however, is that the available communication bandwidth does not scale up with increasing amount of bus masters—i.e., the number of CPUs or other processing elements in general—whereas the

required bandwidth does increase, meaning the bus quickly becomes a performance bottleneck. Increasing cache sizes can mitigate but not avoid this problem. Early approaches to address this bottleneck include hierarchical buses and crossbars which both alleviate the problem for a limited amount of processing elements but do not scale well to the expected amount of tens or even hundreds of cores in future MPSoCs.

An approach where the total bandwidth does scale well with the number of processing elements is the Network on Chip (NoC). In a NoC each network node is only directly connected to a limited amount of neighboring nodes. Messages to other than neighboring nodes are sent over multiple *hops* to their destination. The number of connections—and, thereby, the total available bandwidth—grows with the number of nodes which makes NoCs the ideal on-chip interconnect for large-scale MPSoCs. In order to be utilized in a safety-critical or mixed-critical system, however, the NoC must ensure isolation between communication of different applications. Furthermore, service guarantees such as a maximum latency and a minimum bandwidth must be provided, at least to the safety-critical communication, and lastly, the critical communication must be fault-tolerant.

Different approaches and implementations have been proposed for NoCs in general and to address the challenges of traffic isolation, service guarantees, and fault-tolerance in particular. However, to the best of the author's knowledge, no approach addressing all three of these challenges combined has been proposed so far.

1.2 Contributions

The goal of this work is to develop a NoC that provides efficient on-chip communication in safety-critical and mixed-critical MPSoCs. Specifically, the thesis describes three contributions:

- The development of a concept for a NoC that provides traffic isolation, hard real-time guarantees, and fault-tolerance to communication of safety-critical applications in mixed-critical systems. Traffic isolation and hard real-time guarantees are achieved by using Time-Division Multiplexing (TDM) for critical communication and packet switching for all other communication. Fault tolerance is achieved by introducing protection switching to NoCs.

- The design of the architecture of the hybrid NoC described above, as well as the design of two systems using the hybrid NoC: an evaluation system enabling the evaluation of protection switching in NoC in hardware, and a demonstrator system to showcase the traffic isolation and fault-tolerance of the NoC with an interactive system.

- The development—and particularly the evaluation—of efficient mapping strategies that maximize the communication bandwidth available to the applications in an MPSoC using the proposed NoC. These mapping strategies enable a more efficient use of the network resources, thereby increasing the maintainable flit injection rate and achievable NoC utilization.

Three different protection switching versions are considered for the concept of the hybrid NoC and are compared regarding their expected hardware and power overhead, their respective worst-case latency in case of a fault occurring in the NoC, and their effect on the packet switched traffic. The concept is first evaluated with cycle accurate simulations—with the results first published in [14]—and later with an evaluation system implemented in hardware. The proposed mapping strategies—first presented in [15] and [3]—are used to create task and channel mappings for different traffic scenarios and TDM slot table sizes in an 8x8 NoC. These mappings are then evaluated and compared to each other by running tests in hardware and monitoring the latencies of the packet switched traffic as well as the flit injection rate at which the NoC saturates. The results were published in [16] and [17]. The results presented in this thesis show that:

- The proposed hybrid TDM and packet switched NoC using protection switching is a powerful option for on-chip communication in mixed-critical MPSoCs.
- Despite being hybrid, the proposed NoC router is smaller than a comparable state-of-the-art packet switched router while allowing for competitive flit injection rates before the NoC saturates.
- The strategy with which a given set of critical applications and their communication channels is mapped to a system that uses the proposed NoC has a significant effect on the overall achievable flit injection rate and network utilization. A good mapping strategy can, in the best case, almost entirely cancel out the adverse effect the TDM traffic has on the packet switched traffic.

The work presented in this thesis is supported by the German BMBF research project ARAMiS II with funding ID 01 IS 16025.

1.3 Outline

The remainder of this thesis is structured as follows. The necessary NoC concepts and basics that the contributions of this thesis are based on as well as the current state of the art are described in Chapter 2. The main contribution of this thesis—i.e., the concept for the NoC providing traffic isolation, hard real-time guarantees, and fault-tolerance to safety-critical communication—is presented in Chapter 3. This chapter also provides a formal analysis of the worst-case latencies of the different protection switching versions as well as the development of mapping strategies designed to improve the NoC utilization. Chapter 4 presents the design of the architecture of the hybrid NoC as well as two different systems using the NoC and their hardware implementation. Chapter 5 discusses the Field-Programmable Gate Array (FPGA)-based evaluation of the mapping strategies proposed in Chapter 3. Lastly, Chapter 6 concludes the thesis and gives an outlook to future work.

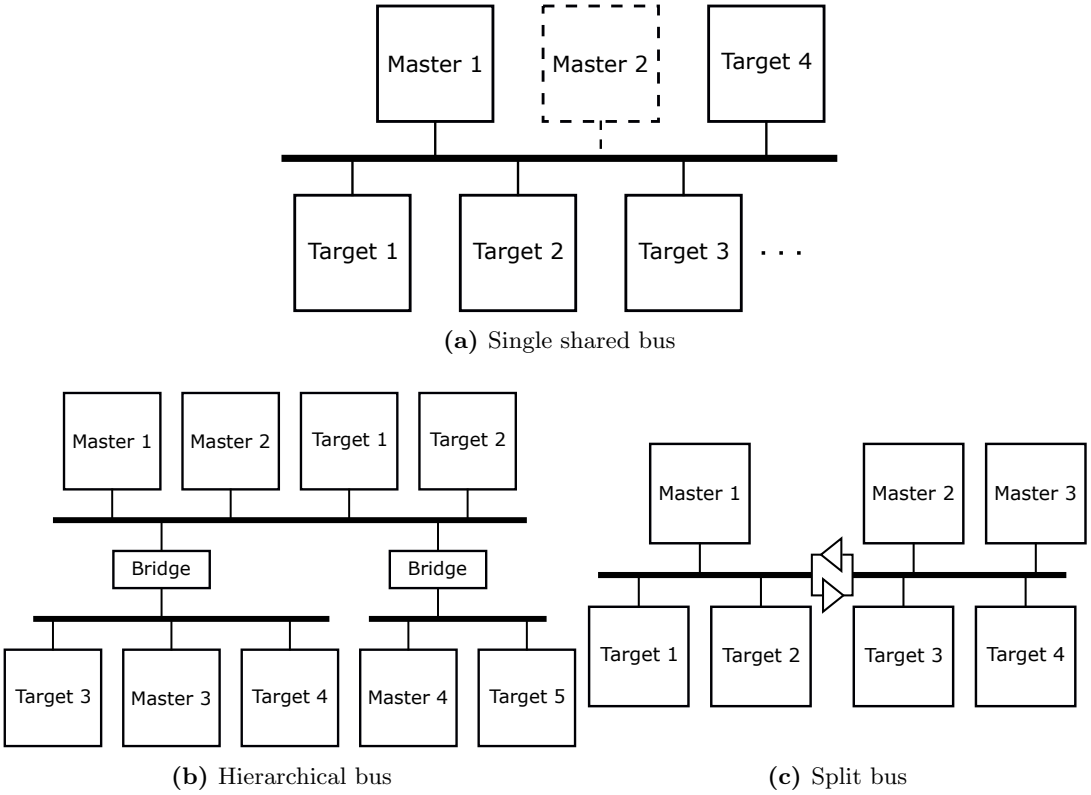
2 Background and Related Work

The main contributions of this thesis are the concept and architecture of a fault-tolerant NoC that provides hard real-time guarantees to critical traffic in an MPSoC as well as the development and evaluation of task and channel mapping strategies with the goal of increasing the available network bandwidth for Best Effort (BE) traffic in order to maximize the achievable total network utilization. Before the subsequent chapters describe and discuss the details of said contributions, this chapter covers the necessary basics of both on-chip communication in general and NoC in particular, including a description of different types of NoCs. Afterwards, an overview over related work is given, divided into three different topics: guaranteed service in NoCs, fault-tolerance in NoCs, and task and channel mapping in NoCs. Lastly, a brief discussion on functional safety is presented.

2.1 On-Chip Interconnect Basics

For decades, single shared buses have been the de facto standard for computer communication, both off-chip and on-chip, and even today they are the basic building block that is used for on-chip communication in many SoCs and MPSoCs [18]. A shared bus is often favored due to its simplicity. It can either be implemented with a single wire as a serial bus or, more typically, with a set of wires as a parallel bus. A shared bus is typically subdivided into individual address and data buses—possibly with individual wires for both read and write transfers—and different configuration signals. Depending on the bus protocol and the widths of the individual address and data buses, a shared bus can easily consist of several hundred individual wires. A single shared bus—shown in Figure 2.1a—is an efficient way of connecting modules in systems with only few initiators or *bus masters* (ideally just one) and multiple target modules (i.e., *bus slaves*) [19]. However, a single shared bus does not scale well and quickly becomes a performance bottleneck in systems with multiple bus masters. Furthermore, increased chip sizes result in larger wire length which limits the clock speed at which the bus can operate.

Driven by multiple factors—most notably technology scaling, the breakdown of Dennard scaling, and the "power wall"—most computing systems, both desktop and embedded, have moved from single processor systems to multi processor systems in the last two decades [18]. This development made it necessary to find new approaches for on-chip communication that are better suited to handle the increased communication requirements. A logical first approach is to use multiple buses instead of a single shared bus which lead to, e.g., the hierarchical bus and the split bus, shown in figures 2.1b and 2.1c respectively. Both topologies allow for multiple bus transactions in parallel as long as a transaction only involves the local bus segment. The hierarchical bus with bus



(a) Single shared bus

(b) Hierarchical bus

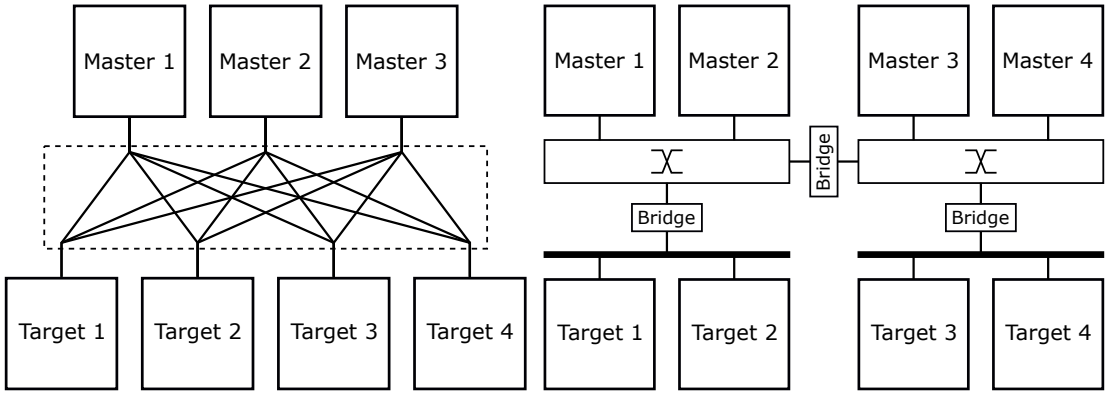
(c) Split bus

Figure 2.1: Shared buses

bridges is a more complex approach. The bridges can provide buffers for the interaction between different bus segments as well as—if necessary—clock domain crossing which allows different buses to operate at different clock speeds or bus-width conversions in case the bridged buses use different widths. The split bus, on the other hand, is typically implemented with a simple tri-state buffer between the bus segments which requires less hardware but does also not provide as much flexibility as a bridge [20, 21].

Systems that require a high amount of parallel transfers can use a full bus crossbar (or bus matrix) as depicted in Figure 2.2a. A full bus crossbar—also called a point-to-point bus—is essentially an individual connection from each master node to each target node. It is easy to see that this comes with large hardware overhead—both in terms of wiring and logic—as each target node needs arbitration logic in case several master nodes want to initiate a data transfer at the same time. Variations are sometimes used in order to decrease the hardware overhead, e.g., by using a shared address bus but a bus matrix for the data bus, or by only implementing those connections that are needed in a deployed system—if known beforehand—which results in partial bus crossbar.

A different well known topology is the ring bus which consists of a number of unidirectional point-to-point connections that are arranged in a circular manner, often with an additional ring bus in opposite direction to decrease the maximum distance between



(a) Full bus crossbar **(b)** Interconnect using crossbars and shared buses

Figure 2.2: Buses allowing for a high amount of parallel transfers

any two nodes. Messages are forwarded through one or more of these connections until reaching their designated destination.

An approach that is commonly used in modern SoCs is a hybrid topology where some components are grouped together and connected to a shared bus segment or a bus crossbar. This allows more flexible designs and makes it easier to implement logical clusters and different clock domains. An example for such an on-chip interconnect composed of multiple buses and crossbars is depicted in Figure 2.2b.

A bus is traditionally a broadcast medium with all connected components listening but only the designated modules reacting to a transaction. Both the bus crossbar and the ring bus are already a clear step away from this paradigm since they both only consist of point-to-point connections. Over the years, many different on-chip interconnect topologies emerged using a mix of shared buses and point-to-point connections, often individually tailored to specific use cases. The move towards NoC is essentially a gradual evolution of the bus-based on-chip interconnects rather than a dramatic and sudden paradigm shift [19]. Many of the emerged interconnects can already be classified as NoCs and—especially in case of large SoCs with multiple shared buses and bus crossbars—the terms "on-chip interconnect" and "NoC" are often used interchangeably.

As previously mentioned, the reason for the evolution of on-chip interconnects lays in the ever increasing number of processing elements and the resulting demand for more transaction parallelism. But even a full bus crossbar or ring bus only scales to a certain point at which the hardware cost (in case of the former) or the communication latencies (in case of the latter) become too large. One of the first works proposing on-chip interconnection networks with a modular design was presented in [22] which laid the foundation for what is now referred to as mesh NoC: an on-chip interconnect that, depending on the topology used, scales approximately linearly with the number of nodes in a system. The following sections will give an introduction to the mesh NoC basics and discuss different types of mesh NoCs.

One example for the described development of on-chip interconnects can be found when looking at the evolution of Intel processors. With multiple cores, a single shared bus was no longer sufficient to move the on-chip data fast enough which lead to the utilization of a ring bus starting with the second generation of Intel Core processors code-named Sandy Bridge [23]. However, with a further increasing number of cores the ring bus once again became a bottleneck which first lead to the utilization of a second ring and later—with even more cores—to an architecture with a mesh interconnect in modern Xeon processors [24].

2.2 Mesh NoC Basics

Mesh NoCs have been a research topic for about two decades now and many different approaches and topologies have been proposed, evaluated, and in some cases implemented in various research and commercial devices. This makes it impossible to cover every aspect of mesh NoCs in this thesis. Instead, this and the following sections give an introduction to mesh NoCs and discuss the basics that are required in order to comprehend the following chapters. For a comprehensive introduction to mesh NoCs the interested reader is directed to additional sources such as [18, 19, 25]. For the remainder of this thesis *NoC* refers to a mesh NoC unless stated otherwise.

As the name suggests, NoCs are an attempt to apply principles from large-scale networks to on-chip communication. The basic building blocks of a NoC-based system are *nodes*, *routers* (or *switches*), and *links*. Routers have multiple input and output ports, which are connected to other routers by links. Together, they build the actual NoC. Nodes can contain processing elements—e.g., CPUs, Digital Signal Processors (DSPs), or special accelerators—shared memory blocks, I/O ports for off-chip communication, or a combination of different components which are then often connected by a local bus. Each node also contains a Network Interface (NI) (sometimes also called Network Adapter (NA)) which connects the node to a router, thereby serving as gateway to the NoC. Figure 2.3 shows an example of a NoC with its components[1].

Similar to a full bus crossbar or a ring bus, NoCs consist of many point-to-point connections. These point-to-point connections are unidirectional connections between the routers as well as between each NI and its neighboring router. Two unidirectional point-to-point connections in opposing directions are typically combined to (and depicted as) a single bidirectional connection over which information can be transmitted concurrently in both directions (cf. Figure 2.3). Inside the routers, a crossbar switch is used to connect the incoming and outgoing connections as needed. As such, NoCs are essentially a very specific combination of point-to-point connections and crossbars to connect the different modules of a SoC.

In contrast to buses—no matter if simple shared buses, full bus crossbars, or ring buses—NoC links are not subdivided into address and data links and only have a few additional signals, e.g., to indicate valid data or implement flow control. As a result, NoC links typically have fewer wires than buses. Furthermore, different nodes or modules are

[1] The depicted NoC topology would be considered irregular.

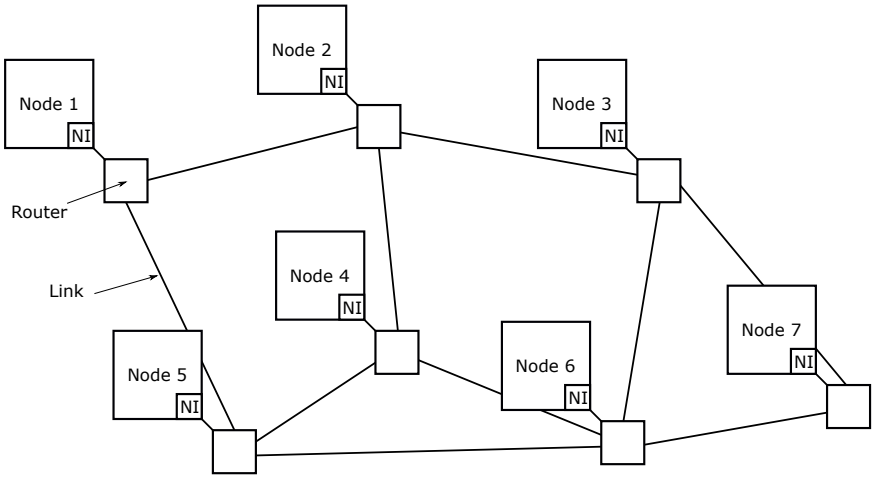
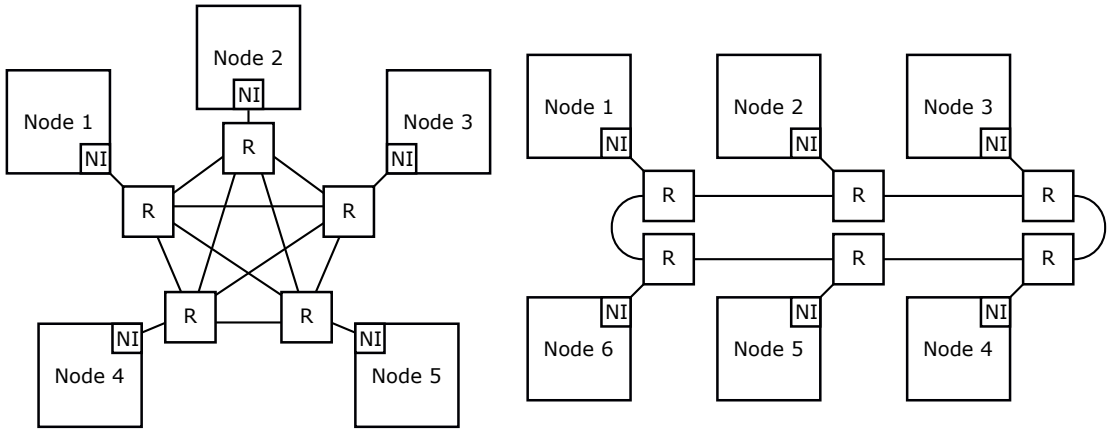


Figure 2.3: Example mesh NoC with nodes, routers, links, and network interfaces

not accessed with a memory address as is done in buses. Instead, messages are sent through the NoC according to a specific protocol (depending on the type of NoC) which will be covered in the following Sections. The main purpose of the NI is to translate the bus protocol that is used within a node to the NoC protocol and vice versa.

Different NoC topologies have been proposed and evaluated over the years and are divided into three general categories: direct networks, indirect networks, and irregular networks. Among the most common ones are direct network topologies, four of which are presented in the following. Direct networks have the property that with a growing number of nodes the number of routers and links also increases which, in turn, increases the total available bandwidth [18]. The major trade-off is between connectivity (resulting in lower latency and higher total bandwidth) and cost. Among the simplest direct topologies are the fully connected mesh NoC and the ring NoC (shown in figures 2.4a and 2.4b), as they are essentially equivalent to the full bus crossbar and the ring bus respectively. Both are viable solutions for a small amount of nodes (typically within the single digit range) but don't scale well for a larger amount of nodes. The point-to-point topology does not scale well due to the quickly increasing cost and the ring topology due to the quickly increasing communication latency.

Two very popular topologies are the 2D mesh NoC and torus NoC, shown in figures 2.5a and 2.5b respectively. In a 2D mesh, each router is connected to two, three, or four neighboring routers depending on whether it is located at a corner, edge, or in *the center* (i.e., not at a corner or edge) of the NoC. All links have the same length which is advantageous for the physical design of the network. The size of the NoC as well as the total available bandwidth scales linearly with the number of nodes. A disadvantage is that the connectivity of the corner and edge routers—and thereby the nodes—is lower than for the central routers. Furthermore, communication latency can become an issue for very large meshes as the average distance between any two nodes increases (although much slower than, e.g., with a ring topology). Both factors also typically lead to a higher



(a) Fully connected mesh NoC **(b)** Ring NoC

Figure 2.4: Simple NoC topologies

network utilization and possible congestion in the center of the NoC which should ideally be compensated by suitable routing algorithms.

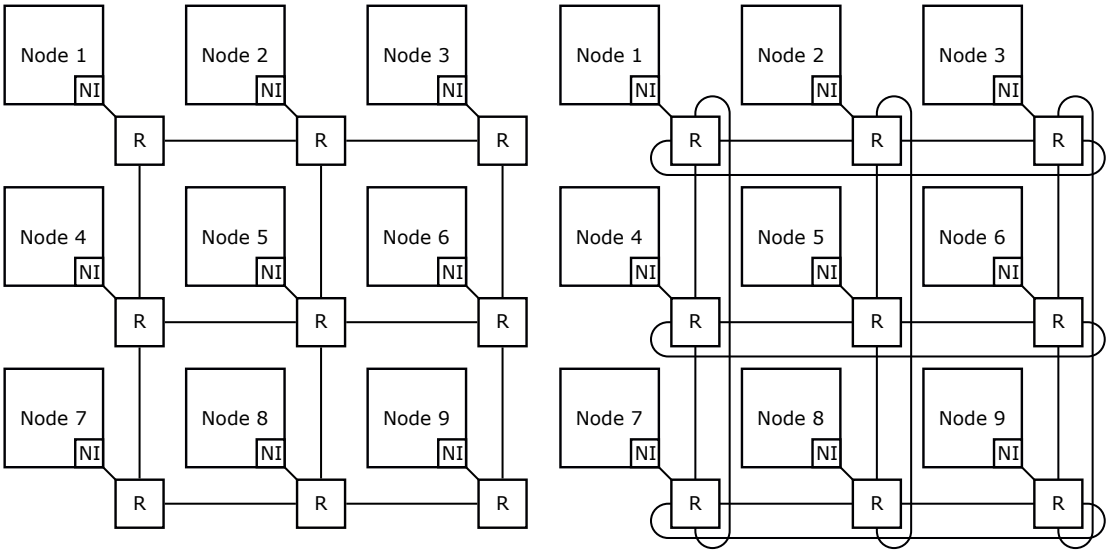
The disadvantages of a 2D mesh can be mitigated with a torus topology. Here, there are no corner or edge routers, meaning every router is a central router. This leads to a more equal utilization of the available links and, furthermore, to a lower latency than that in a mesh. However, a torus has the disadvantage that more (costly) wiring is needed, from each side fo the chip to the opposite one. This also means that the links are not all equally long which negatively affects the maximum clock speed at which the NoC can operate. However, another advantage, which both mesh and torus topology share, is that routing is fairly simple and can easily be implemented in hardware.

As can be seen in Figure 2.5, both mesh and torus have a tiled layout which simplifies chip design and is another reason for their popularity. Due to this tiled layout, nodes are also called *tiles*. In a tiled NoC, the directions in which information can travel are referred to by the four cardinal directions: north, east, south, and west. Accordingly, (central) routers have five input and output ports: northern, eastern, southern, western, and local port, the last of which is connected to the NI of the local node. For the remainder of this thesis the focus will be put on the 2D mesh topology—as it is one of the more popular ones—and MPSoCs that use a 2D mesh NoC for on-chip communication.

Besides their topology, NoCs can be categorized regarding their switching strategy and routing algorithm. The following sections will give an overview over the different approaches and their respective traits.

2.2.1 Switching Techniques

The nodes in a NoC generate and send messages of varying length to one another. How these messages are organized and how they flow through the NoC is determined by the switching technique or strategy. The two main techniques are *packet switching* and *circuit switching*. Both techniques share a few commonalities regarding data organization



(a) Regular 2D mesh NoC **(b)** NoC torus

Figure 2.5: Tiled NoC topologies

on link level but are, besides these, fundamentally different. The commonalities are described in the following paragraph.

In both, NoCs that use packet switching and NoCs that use circuit switching, data is transferred between the routers via links which have a fixed bit width. The width of the links determines the most basic unit of data in a NoC: the *phit* (physical unit). A phit is the amount of data that is transferred over a link in a single clock cycle. In order to avoid buffer overruns, the data transfer between two routers is synchronized on link level with so called *flits* (flow control units). A flit consists of at least one phit but can also comprise multiple phits. How these phits and flits flow though the NoC depends on the switching technique used. The individual traits of both packet and circuit switching are covered in the following sections 2.2.2 and 2.2.3. Hybrid approaches are introduced in Section 2.2.5.

2.2.2 Packet Switching

With packet switching, each message is partitioned into data *packets*—consisting of one or more flits—which are then sent out as individual entities. Each packet has its own header flit which contains at least the routing information for that packet and possibly additional information such as a traffic class or control data. The header flit is typically followed by a number of body flits and a tail flit marking the end of the packet. The body and tail flits carry the payload, i.e., (part of) the message. Some NoCs also use single flit packets, e.g., for control messages.

Figure 2.6 shows the hierarchy of message, packet, flit, and phit. In this example packets can be up to four flits long, including the header flit, and each flit consists of

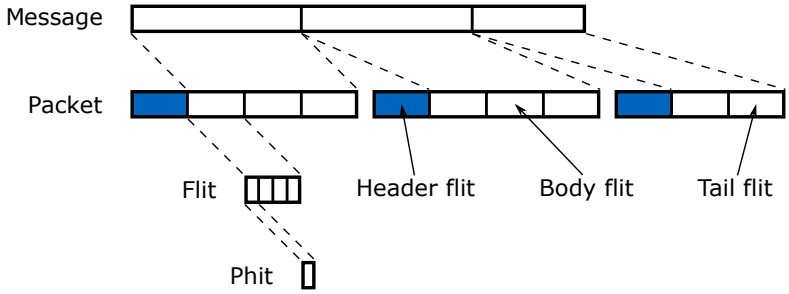


Figure 2.6: Hierarchy of messages, packets, flits, and phits

four phits. However, in most NoCs flits and phits are the same size which effectively removes phits from the hierarchy and makes flits the smallest data unit. One example for a special case is Nostrum, where one packet equals one flit which equals one phit (all three are determined by the link width which is 128 bit) [26]. For the remainder of this thesis flits are assumed to be the smallest data unit in a NoC unless stated otherwise.

Each packet is individually routed through the NoC according to the applied routing algorithm (discussed in Section 2.2.6). This means that different packets from the same source to the same destination can potentially take different paths and experience different latencies which, in turn, could lead to packet reordering. Packets are sent out by the NI as soon as they are created, no matter the state of the NoC[2]. This means that they will both cause and experience congestion in the network which can greatly increase the latency of a packet (congestion is further discussed in Section 2.2.7). The consequence is that it is very difficult in packet switched NoCs to give any Quality of Service (QoS) guarantees such as bound latency or a minimum bandwidth.

Figure 2.7 shows an overview over a typical simple packet switched router and its components. The main components are an input buffer—typically in the form of a simple First In, First Out (buffer) (FIFO)—a routing computation module at each input port, a switch fabric connecting the input ports to the output ports, and an arbiter. The routing computation module determines the output port to which an incoming packet will be forwarded. This can be as simple as a static lookup table or more complex to implement dynamic routing. The arbiter is responsible for resolving possible contention in case multiple packets are routed to the same output port at the same time. More elaborate designs are possible and can help to increase the throughput or reduce congestion (e.g., by introducing pipelining of different processing steps such as routing computation, switch allocation, etc., cf. [25]).

There are three different packet switching schemes: store and forward, virtual cut through, and wormhole switching. The first two are virtually never used in NoC since a packet is only ever sent to the subsequent (or downstream) router if that router has enough buffer space to receive the entire packet[3]. This leads to large buffer requirements

[2]Unless this is prevented by suitable measures, as is discussed in Section 2.2.7.

[3]Once again, Nostrum is a special case using store and forward switching but with only single flit packets. [26]

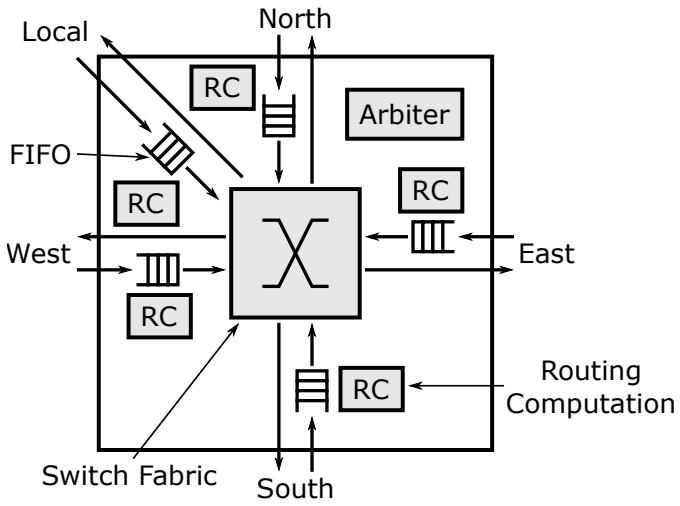


Figure 2.7: Simple packet switched router

and, in case of store and forward, additionally to a high latency. Hence, these two switching schemes will not be further discussed here.

Most packet switched NoCs use wormhole switching (e.g., [27, 28, 29]). With wormhole switching, a flit can be sent to the downstream router if that router has at least buffer space for that one flit. This means that the minimum buffer requirements for each input port of a router are reduced to just one flit[4]. However, the consequence is that a single packet is typically stretched across multiple routers (stretched through the NoC like a worm, hence the name). If a packet is blocked due to no buffer space being free at a downstream router, several links occupied by the packet are blocked as well which increases interference with other packets and, thereby, leads to a higher congestion in the NoC. Additionally, this can lead to deadlocks (both congestion and deadlocks are further discussed in Section 2.2.7). One common way to mitigate link blocking is to use Virtual Channels (VCs) (sometimes referred to as *virtual links*) [25]. VCs are implemented by instantiating multiple buffer queues in parallel which are then multiplexed on the physical link. If one VC is blocked it is still possible to transmit packets on another one. The downside, however, is that this can drastically increase the size of the routers due to the additional buffer queues and necessary arbitration logic.

Typically, the largest parts of a packet switched NoC router are the buffers which not only results in large area requirements but also a considerable power consumption [30]. Bufferless NoCs have been proposed as a way to reduce the size and power consumption of the routers [31, 32]. In a bufferless NoC, flits are only stored at each router for a single clock cycle and then immediately forwarded (or dropped) in the next cycle. Therefore, each input port of a router has only a single register stage holding one flit. The downside of bufferless NoCs is that they are only efficient for relatively low amounts of traffic.

[4]Ideally, the buffer size is at least two flits in order to avoid stop-and-go behavior and allow the full bandwidth to be used [25].

Overall, packet switching is a very popular switching strategy due to its great flexibility and relatively simple implementation of a basic NoC. In general, packets are sent individually and without knowledge about the status of the NoC which makes them suitable for systems in which the communication patterns and traffic loads are dynamic or not entirely known at design time. However, giving QoS guarantees is very difficult in packet switched NoCs—as will be discussed in Section 2.2.7—which makes it difficult to use them in safety-critical systems.

2.2.3 Circuit Switching

The origins of circuit switching go back to the early analog telephone networks where lines were physically switched in order to establish an electrical circuit between two telephones [33]. The lines used were reserved for the duration of the call and, thereby, blocked for all other communication. Similarly, with circuit switching used in a computer network or NoC, a communication channel is established by reserving a physical path—made up of a series of routers and links—between a source and destination node prior to any data being sent. Channels are typically established on-demand. One possible way of setting up a channel in a NoC is by sending a special header flit along a path and reserving the traversed links [18, 34]. If the channel is successfully set up, an acknowledgement is sent back to the initiating node to signal that the transmission can start. Once the transmission has finished, the channel is typically torn down by another special flit.

In contrast to packet switching, circuit switching does not require packetization of the messages sent. All flits of a message follow the same previously set up path. This not only makes header flits unnecessary but also ensures that all flits arrive in the same order that they were sent in. For the hierarchy depicted in Figure 2.6 this means that the *packet* layer is removed and the entire bandwidth of a communication channel can be used for the payload.

Circuit switching makes it easily possible to give QoS guarantees—at least once a channel has been established—since the entire bandwidth of the links is reserved for the channel, leading to low latency transfers. In contrast to packet switching, a message must wait initially until a channel as been set up—instead of being sent out right away—but experiences minimal latency afterwards. This makes it ideal for the transmission of larger amounts of data where the initial delay is less relevant. Circuit switching can also be a good choice when data is sent very often and/or the communication pattern is relatively static and known beforehand (e.g., known at design time, or known to be periodic so a channel can already be established right before it is needed). However, during the transmission, no other channel can use the reserved links which means the approach does not scale well with growing NoC size [18]. For this reason pure circuit switching is practically not used in NoCs [19]. An exception, however, is the SoCBUS [35].

An alternative to pure circuit switching—and a way to mitigate its drawbacks—is virtual circuit switching. There are two ways of implementing virtual circuit switching. One way is to use multiple buffers for each link, similar to mitigating the drawbacks of wormhole switching in packet switched NoCs by using VCs. NoCs that use this type

of virtual circuit switching include [36, 37]. This approach, however, also leads to large routers due to the additional buffers and to potentially very high worst-case latencies of the flits sent, even if a specific bandwidth can be guaranteed. At least the latency issue, though, can partially be addressed by using advantageous scheduling of the input buffers (e.g., [38]).

The other way of implementing virtual circuit switching is by only using a single-flit-buffer for each link and employing TDM throughout the NoC to schedule the usage of links between different virtual circuits. This switching scheme makes it easy to provide end-to-end QoS guarantees—specifically regarding the worst-case latency—and results in very small, cheap, and fast routers [39]. Virtual circuit switching using TDM plays a crucial role in the NoC presented in this thesis and will, therefore, be discussed in further detail in the following Section 2.2.4.

2.2.4 TDM NoCs

The first NoC using TDM to implement virtual circuit switching was the Æthereal NoC [40, 41, 42]. The Æthereal NoC uses "contention-free routing, or pipelined time-division-multiplexed circuit switching" in order to provide QoS to critical traffic. For the remainder of this thesis this principle will simply be referred to as *TDM NoC*.

In a TDM NoC, each network node (i.e., router and NI) has a slot *table* T with S *slots* that stores the routing information for each output and each time slot. All nodes cycle through their slot tables synchronously. In each time slot t, the output ports of the routers forward flits from a defined input port according to the slot table entry for that slot. A connection (or *TDM channel*) is set up by configuring consecutive slot tables along a path to forward flits in subsequent time slots $t + 1$, beginning and ending with the sending and receiving NI respectively. As a consequence, TDM channels are inherently unidirectional, meaning that two individual channels—one per communication direction—must be set up in cases where a bidirectional communication is necessary. TDM NoCs typically require the NoC to be fully synchronous, meaning all routers and NIs must be in the same clock domain. This might be problematic for large designs, as will be discussed in Section 2.2.7. It is, however, also possible to use TDM in mesochronous or even asynchronous NoCs as well [43, 44, 45].

Figure 2.8 shows the basic principle of a TDM NoC with two TDM channels: a and b (for simplification, only the slot table columns that are being used are depicted). In this example, the slot table size is four which means up to four different TDM channels can theoretically share any given link, with each channel being able to use 25% of the link's total bandwidth[5]. However, in the example shown channel a uses two slots, meaning it can use up tp 50% of the link's total bandwidth.

A slot can have a duration of multiple clock cycles or just one. If it lasts for multiple cycles, flits are sent over a TDM channel in *blocks*, which propagate through the NoC in a store and forward fashion. This, however, increases the buffer requirements in

[5] The total number of possible NoC channels is of course higher and is usually limited by the size of the network as well as the number of applications and their mapping.

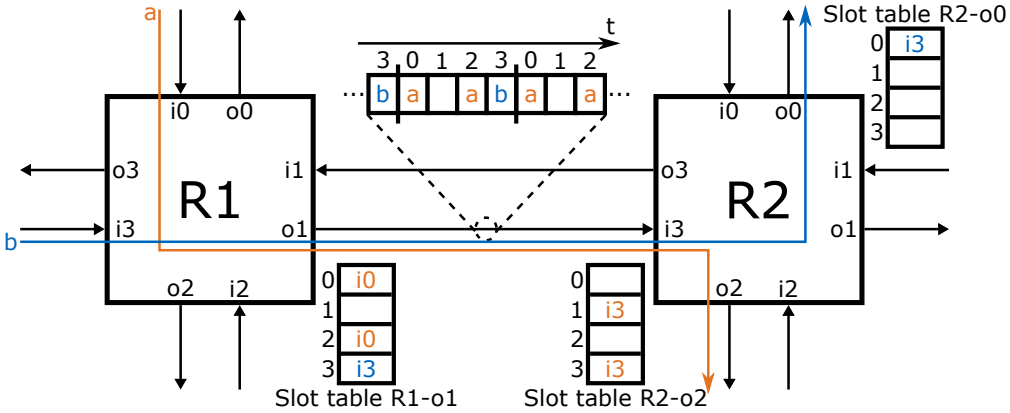


Figure 2.8: Basic TDM scheme with two channels a and b (local links not shown)

the routers. Therefore, a slot duration of just one clock cycle will be assumed for the remainder of this thesis.

With a duration of just one cycle, the input buffers at each router can be reduced to a single register stage making the routers very small. Figure 2.9 shows an overview over the resulting TDM NoC router. As can be seen, the router is much simpler than its packet switched counterpart depicted in Figure 2.7. Once injected, a flit will take exactly as many cycles to reach its destination as the destination is hops away. The NIs write/read flits to/from the network according to their slot tables. Messages sent from one node to another only ever face (a deterministic) delay at the sending NI.

TDM NoCs are advantageous for safety-critical traffic since the pipelined forwarding guarantees a deterministic latency. Furthermore, since a fixed number of slots is assigned to a channel the connection also has a guaranteed bandwidth and is isolated from other channels, thereby ensuring QoS by design. It is up to the developer to ensure that the traffic generation rate of a sending node does not exceed the amount of bandwidth reserved for that node. Contentions cannot occur in a TDM NoC since the time slots for each connection are reserved in advance. The disadvantage, however, is that the reserved bandwidth of a channel is typically overprovisioned which in turn leads to an underutilization of the network as a whole since reserved resources cannot be used by other connections, even if they are not used at a time. This is a general disadvantage of circuit switched NoCs and it is particularly problematic for dynamic and unpredictable traffic patterns. For these reasons, packet switched NoCs are generally more popular.

One of the main design parameters of TDM NoCs is the slot table size. Larger slot tables allow for more flexibility when reserving slots for TDM channels since bandwidth can be reserved in smaller quantities. With a slot table size of four, e.g., each channel is assigned at least 25% of the available link bandwidth, even if much less is used. The unused bandwidth cannot be used by other channels and is, therefore, lost. A larger slot table also allows more channels to share a single link. However, large slot tables can increase the router size considerably. Furthermore, large slot tables negatively affect the message latency (this is further discussed in Section 3.3). Therefore, a trade-off has

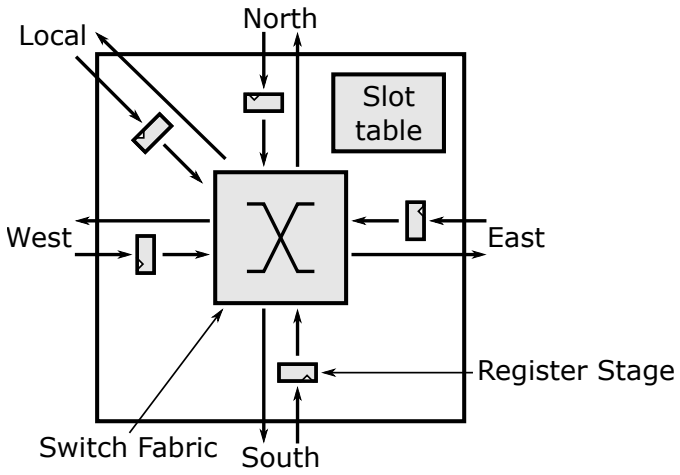


Figure 2.9: Simple TDM router

to be made when defining the size of the slot table between flexibility and bandwidth reservation granularity on the one hand and cost and message latency on the other hand.

There are other ways of implementing a TDM NoC than the one described above. One such way is by removing the slot tables from the routers and only keeping the slot tables in the NIs. This, however, means that *a)* a slot must have a duration of multiple clock cycles and *b)* routing information (in form of a header) must be added to the flits sent, essentially resulting in a packet switched NoC. The difference to a typical packet switched NoC is that the NIs only sent packets in reserved slots—according to their slot table—which eliminates contention in the routers.

Slot Table Configuration

One important aspect of any NoC that uses TDM is the question of how to configure the slot tables in the routers and NIs. This question is also tied to the general architecture and use case of the NoC and, hence, different approaches exist.

The most rigid approach would be a completely static configuration that is defined at design time and cannot be altered once the system is deployed. This has the advantage that it comes with low complexity and guarantees that a TDM channel cannot accidentally be overwritten. However, although there may be some use cases where this static configuration is acceptable, or even desired, most systems will need the flexibility to adjust the communication channels after deployment.

If the slot tables must be adjustable at runtime they must have an interface for the configuration and they must somehow be connected to the entity (or entities) configuring the tables. Generally, there are two different approaches to configure the slot tables: centralized or distributed. Furthermore, there are four different general approaches of connecting the slot tables to the configuring entity (or entities): by using an overlay

network, probe-based, direct wiring to a central entity, or by using packet switched traffic[6].

Distributed Configuration

In systems with distributed configuration the sending NIs typically try to configure a specific path without having prior knowledge of the slot tables in the system. The configuration can be done by either using packet switched traffic—if available—or probe-based, as proposed in [34].

Distributed configuration scales well since no global knowledge is necessary. Individual NIs can set up a TDM channel when needed and tear it down when it is no longer needed. However, the drawback is that it cannot be guaranteed that a TDM channel can be set up or how long it will take to find a free path (i.e., unless the channels are already defined at design time and only set up by the NIs at start time). Hence, this method is unsuited for systems with safety-critical hard real-time traffic.

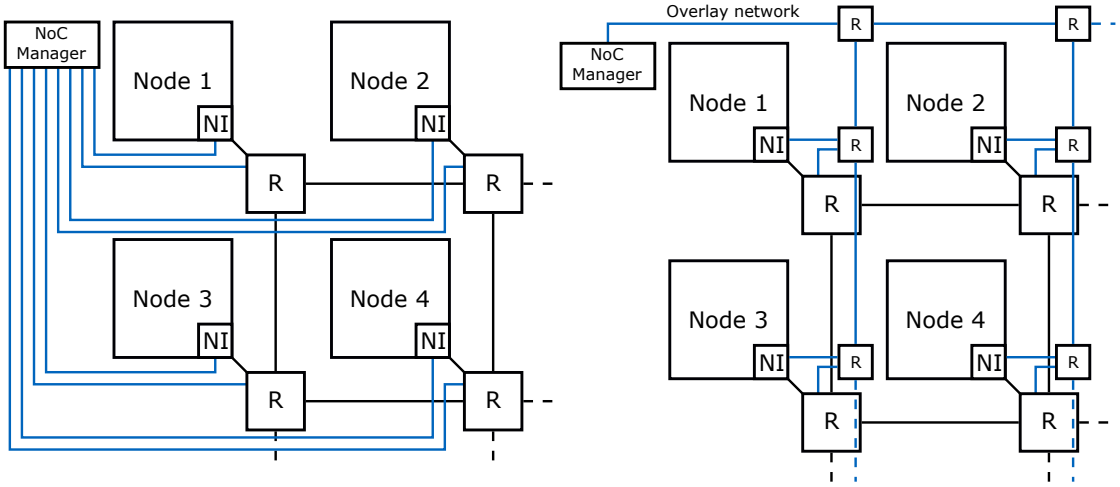
Centralized Configuration

Systems with centralized slot table configuration have a central NoC manager unit that has global knowledge of the configuration of all slot tables and that can configure the tables. This NoC manager can, e.g., be one of the processing nodes in the system or a dedicated hardware module. With its global knowledge, the NoC manager can find optimal or near-optimal paths for each TDM channel and immediately knows whether or not an additional channel can be mapped. Furthermore, with additional knowledge of the status of the NoC links the manager can quickly react to any faults in the network. The disadvantage of a central management unit is that it limits the scalability of the system [40]. It is possible to overcome this limitation by using multiple management units, however, this increases the complexity and means that a single manager no longer has global knowledge of the system.

The NoC manager can be connected to the slot tables either with direct wiring, with an overlay network, or by using packet switched traffic. Direct wiring—as depicted in Figure 2.10a—generally does not scale well and is, therefore, only a sensible choice in relatively small systems. An overlay network increases scalability but comes with additional hardware cost. However, the overlay network can be much simpler than the *main* NoC and, e.g., use smaller links and operate at a lower frequency. An example for an overlay network is shown in Figure 2.10b. There is also no need for the overlay network to be fully connected. Instead, other topologies such as a spanning tree can be used [46]. Lastly, packet switching can be used in hybrid NoCs (further discussed in the following Section 2.2.5). This has the advantage that existing infrastructure can be used.

In addition to the connection to the slot tables, a connection to the nodes is necessary which can be implemented in a similar way. This connection is necessary in order for the nodes to be able to request TDM channels. Even if the channels are not requested

[6] Only possible if the NoC also supports packet switching, cf. Section 2.2.5.



(a) Configuration using direct wiring **(b)** Configuration using an overlay network

Figure 2.10: Central configuration with a NoC manager

but automatically set up at system start time, the NoC manager must signal the NIs when the channels are set up and they can start sending data.

Centralized configuration is generally better suited for systems that require QoS since all decisions are made by a single privileged entity.

2.2.5 Hybrid NoCs

Both packet switching and circuit switching have their strengths and weaknesses. Packet switched NoCs are very flexible but the implementation of QoS is typically costly. Circuit switched NoCs, on the other hand, can easily implement QoS but are less flexible and often suffer from underutilization. It is, hence, an obvious approach to try and combine both switching techniques in order to mitigate the downsides of either. The result is a *hybrid* NoC.

There are many different hybrid NoCs that can and have been proposed and implemented. One approach, e.g., combines Spatial-Division Multiplexing (SDM) with TDM and packet switching, another one uses circuit switching for important messages alongside packet switching [47, 48]. The previously mentioned Æthereal NoC also defines a configuration as a hybrid NoC [40]. This section describes a hybrid TDM and packet switched NoC that is similar to the ones proposed in [40] and [49]. The described NoC builds the basis for one of the main contributions of this thesis: a fault-tolerant NoC that provides hard real-time guarantees to critical traffic in a MPSoC.

To implement both virtual circuit switching with TDM and packet switching, the routers of the NoC are essentially divided into two parts, one handling the TDM traffic, the other one handling the packet switched traffic. A structural overview over a hybrid router is given in Figure 2.11. Incoming flits are identified as either TDM or packet switched flits and then forwarded accordingly. TDM flits are stored in the register stage

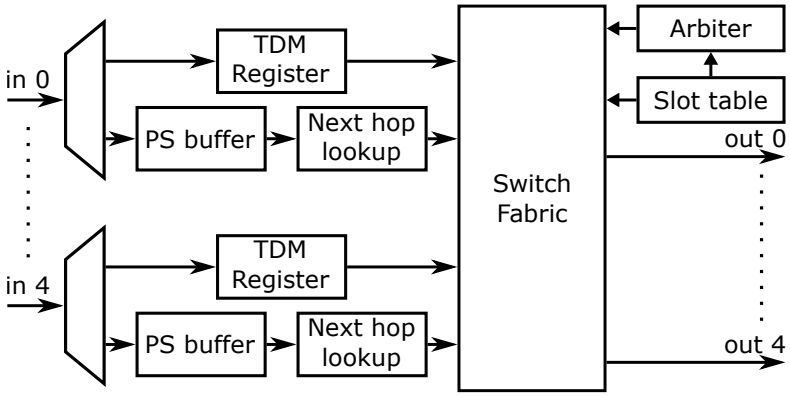


Figure 2.11: Overview of the hybrid router design

of the input port. Packet switched flits are added to the buffer queue for packet switched traffic and the header flits are inspected to determine the next hop of the packet. In each cycle, the slot table configures the switch fabric according to the connections that are reserved for TDM traffic, if any. The arbiter then configures any remaining connections that can be used by the packet switched traffic while resolving occurring contention and ensuring bandwidth is fairly shared between all input ports. It is important to note that the TDM routing is still contention-free since a scheduled TDM flit has always precedence over packet switched traffic and two TDM flits can never collide by design.

The use of a hybrid NoC increases the overall network utilization compared to a pure TDM NoC since the packet switched flits can fill the otherwise free time slots. This also means, packet switched traffic can use paths on which no additional TDM channel would be possible. This is the case if enough slots are free on each link along a path but the slots are not consecutive, thereby making a TDM connection impossible[7]. Packet switched traffic can even use reserved but unoccupied TDM slots, a technique known as time-slot stealing [49]. An example of this practice is shown in Figure 2.12. In addition to the TDM channels a and b, packet switched traffic traverses the link from router *R1* to router *R2*. The packet switched traffic not only uses the unassigned slot in the routing table (slot 1) but also the slots of channel a (slots 0 and 2) whenever the slots are not used for TDM traffic. It should be noted, though, that this practice weakens the isolation between the TDM and packet switched traffic to a certain degree. The TDM traffic will always be able to use the full bandwidth that has been reserved but the packet switched traffic might be able to observe traffic patterns of the TDM traffic, potentially enabling side-channel attacks [50, 51]. Time-slot stealing should therefore not be allowed for TDM channels that are used for secure communication.

There are different incentives to use a hybrid TDM and packet switched NoC over a pure TDM or packet switched NoC. The hybrid approach allows it to individually use the best suited switching technique for each application in a system. TDM can be used to give QoS guarantees to critical applications or to temporarily speed up large transfers

[7] This is a problem that is related to the mapping of TDM channels and will be discussed in Section 3.5.2.

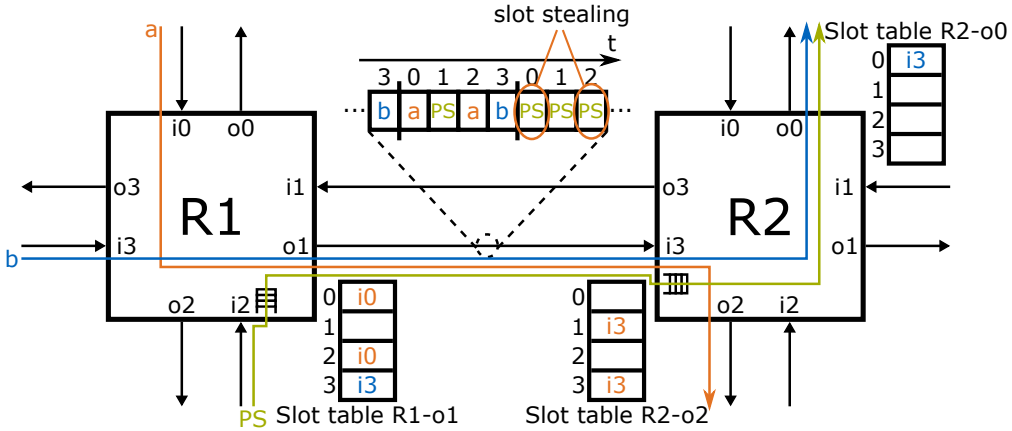


Figure 2.12: Hybrid NoC with TDM and packet switched traffic (local links not shown)

(e.g., [49]). Packet switching, on the other hand, can be used for dynamic workloads and BE applications and increases the overall flexibility and network utilization.

The downside of a hybrid TDM and packet switched NoC is that the hybrid router is more complex than either of the simple routers depicted in figures 2.7 and 2.9 respectively. The reason is obvious: the hybrid router must implement the logic to handle both types of traffic and *additionally* implement logic to arbitrate between the two traffic types. However, many packet switched NoCs go great lengths to increase performance by using VCs or implement QoS to varying degrees, thereby greatly increasing the size of the routers. In contrast, the packet switched part of the described hybrid NoC can be very basic since TDM can be used to give QoS guarantees. Therefore, the hybrid TDM and packet switched router can even be smaller than a state-of-the-art packet switched router (an example is presented in Section 4.4.1). In general, a system developer has great freedom regarding the complexity of the packet switched part of the hybrid NoC (e.g., number of virtual channels, traffic classes, routing scheme, etc.).

2.2.6 Routing Algorithms

The routing algorithm of a NoC determines how packets (or circuits) are routed through the network, i.e., which path(s) can be taken. The selection of an algorithm has an effect on different metrics of the NoC such as power consumption, area cost, robustness, and performance. Routing algorithms can broadly be classified as *static* or *dynamic* routing, *distributed* or *source* routing, and *minimal* or *non-minimal* routing [18, 19].

Static and Dynamic Routing

With static routing (also called *deterministic* or *oblivious* routing) the paths between any two source and destination nodes are determined at design time. These predefined paths are always taken regardless of the current state of the NoC (i.e., regardless of current link load or contentions). It is possible to split data among multiple paths in a predefined

manner but often only a single path is defined for any given source/destination pair (which has the advantage that out of order delivery is avoided). Static routing has the advantage that it is typically very cheap to implement in hardware. However, static routing algorithms typically suffer from poor load balancing since it is not possible to react to congestion.

One of the most common static routing algorithms used in 2D mesh NoCs is XY routing (or, more general, dimension-order routing [52]). Here, flits first travel along the X-axis of the mesh (horizontally) until the reach the same column as their destination node. They then travel along the Y-axis(vertically) until reaching the destination. Despite its shortcomings (most notably the poor load balancing), XY routing is widely used since it is very cheap to implement and it avoids deadlocks (further discussed in Section 2.2.7) [19].

In contrast to static routing, dynamic (or *adaptive*) routing algorithms base their routing decisions on the current status of the NoC which leads to better load balancing and an overall higher link utilization since congested links can be avoided. This means that the path from a source to a destination node can change over time. However, this also means that packet reordering can become an issue. Dynamic routing algorithms are more difficult to implement than static ones and require monitoring of the NoC. An example for a dynamic routing algorithm is presented in [53].

Distributed and Source Routing

The routing algorithms—both static and dynamic—can be further differentiated depending on *where* the routing information is stored or the routing decision is made. When using distributed routing in a packet switched NoC, each header flit contains information about the destination node (i.e., the identifier or, in a 2D mesh, the XY coordinates of the destination node). Using this information, each router decides independently what the next hop of the flit(s) will be. This can either be implemented with precomputed routing tables or by executing a hardware function.

Routing tables are typically a good solution for static routing algorithms and relatively small NoCs as they can grow considerably with increasing NoC size. Implementing a hardware function, on the other hand, typically leads to router sizes that are independent from the NoC size. An example for an algorithms that can easily be implemented in hardware is static XY routing. In this case each router can compare the destination coordinates to its own coordinates and then forward flits accordingly.

TDM NoCs typically also use distributed routing in form of slot tables in the routers, making header flits unnecessary. In contrast to the routing tables used for distributed routing in packet switched NoCs the slot tables do not necessarily grow in size in larger NoCs.

When using source routing, the route of the flits that are sent are determined at the sending NI. Similar to distributed routing this can be implemented using either routing tables or implementing a hardware function (or even a software function in a processing element). In this case, the header flit holds the information about the path that is taken through the NoC. This is typically done by storing the number of the output port that

is taken at each router along the way. Each router extracts the information about the output port and rotates the routing information in the header for the next router. This leads to a very compact router design and a router size that is independent from the size of the NoC.

Source routing is typically only used in packet switched NoCs due to the requirement of a header flit. However, it can also be used in a TDM NoC. As an example, source routing is used for TDM traffic in one of the configurations defined by the Æthereal NoC in which the slot tables are removed from the routers and only the slot tables in the NIs remain [40].

The downside of source routing is that it requires larger link widths with increasing NoC size since the number of required bits to store the routing information grows linearly with the maximum hop distance. In a 2D mesh topology, 3 bits are required for each hop to encode the five possible output ports (four outgoing links to neighboring routers plus the local link). This means that, e.g., in an 8x8 mesh NoC—which has a maximum hop distance of 15—a link width of 45 bits would be required. Hence, source routing does not scale well with growing NoC size.

Minimal and Non-Minimal Distance Routing

Routing algorithms can be further classified depending on whether or not they allow non-minimal distance paths to be taken. A typical example for an algorithm that only allows minimal distance paths is XY routing (which is even more strict since only exactly one path is allowed). Allowing non-minimal distance paths gives greater flexibility and can, e.g., allow to avoid congested links (i.e., if dynamic routing is used). However, non-minimal distance routing algorithms are typically more complex to implement than minimal distance algorithms and can lead to a power overhead.

2.2.7 Challenges in NoC Design

In addition to the major aspects of NoCs described in the previous sections, this section briefly covers some of the challenges that can occur when designing a NoC. Furthermore, common approaches to address these challenges are discussed.

Deadlocks and Livelocks

One problem that especially packet switched NoCs using wormhole switching are prone to are *deadlocks* [54]. When a deadlock occurs, one or more packets are blocked and waiting for an event that cannot happen. This typically happens when multiple packets are blocked by each other in a circular manner. An example can be seen in Figure 2.13. Here, packet *p1* is blocked by packet *p2*, which itself is blocked by *p3* which again is blocked by *p4* which is blocked by *p1*. The result is a situation in which all packets are blocked indefinitely.

There are two different ways to address deadlocks. They can either be resolved once they occur using *deadlock recovery* or they can be avoided from forming in the first place using *deadlock avoidance*. Examples for deadlock recovery are presented in [55]

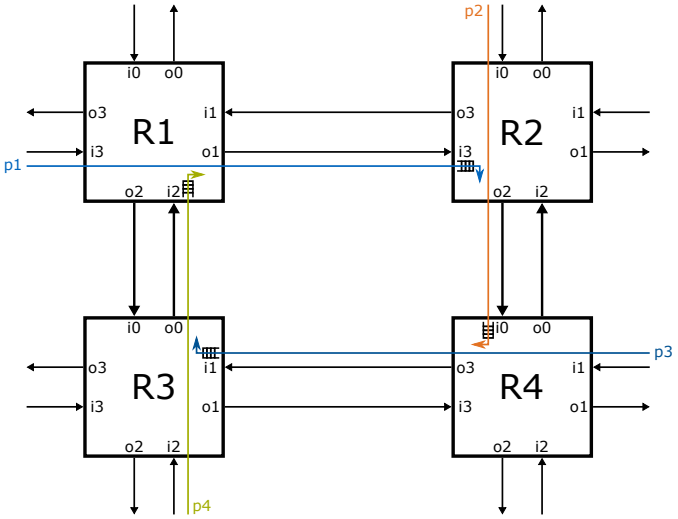


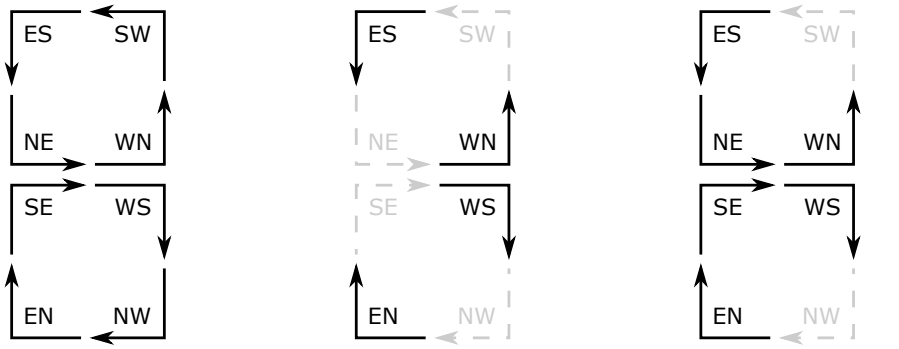
Figure 2.13: Deadlock situation

and [56]. More common, however, is the approach of deadlock avoidance. Deadlock avoidance is typically the responsibility of the chosen routing algorithm. In order to avoid deadlocks, the routing algorithm must make circular dependencies like the one depicted in Figure 2.13 impossible. This is typically done by building a dependency graph of the shared network resources and—statically or dynamically—analyzing whether the graph is cyclic.

An example for deadlock avoidance is based on the *turn model* which prohibits packets from taking certain turns in a router [57]. There are eight possible turns in a 2D mesh NoC, shown in Figure 2.14a: north-east (NE), north-west (NW), south-east (SW), south-west (SW), east-north (EN), east-south (ES), west-north (WN), and west-south (WS). By restricting the allowed turns to only a specific sub-set of these turns it is possible to guarantee that circular dependencies cannot occur.

One of the more restrictive routing algorithms based on the turn model to guarantee deadlock freedom is, once again, XY routing. XY routing—as shown in Figure 2.14b—prohibits four out of eight possible turns: SW, SE, NW, and NE. The remaining turns can never cause a circular dependency. Even adaptive routing algorithms are possible with the turn model. An example for this is *west first* routing which only prohibits two out of eight possible turns (as shown in Figure 2.14c). Using VCs can also aid in avoiding deadlocks and, furthermore, increase the overall network throughput by reducing blocking of packets [57]. However, the number of VCs needed to completely solve the deadlock problem is large which in turn leads to large and costly routers [58].

A different type of deadlock—although occurring only relatively rarely under typical circumstances—is the so-called *message dependent deadlock* [59, 60]. It occurs when messages build circular dependencies, virtually creating turns which are normally forbidden. One way of addressing this issue is by suitable end-to-end flow control mechanisms. Another way that has been proposed is by dropping selected packets [61].



(a) Possible turns in a 2D mesh NoC **(b)** XY routing **(c)** West first routing

Figure 2.14: Deadlock avoidance based on the turn model

Another potential problem are *livelocks*. A livelock can occur when adaptive routing is used and a packet is routed through the NoC indefinitely without ever reaching its destination. One scenario in which this can happen is when deflective routing is used (e.g., [26]). With deflective routing, a flit must always be forwarded in the next cycle due to a lack of buffers in the routers. If the desired output port is not free, the flit must be forwarded to another port. This can potentially lead to the flit never reaching its destination. However, such livelock scenarios can typically be avoided by applying priority rules.

Congestion and Flow Control

A typical problem in packet switched NoCs is contention and—as a direct consequence—congestion. Contention occurs when two or more packets try to access the same shared resource (link and/or buffer) at the same time. In this case, all but one of the packets must wait until the requested resource is free. Following packets are also stalled leading to congestion and potentially head-of-line blocking (meaning a following packet has to wait although the shared resource that it requests would be free) [25]. The stalled packets may themselves block shared resources down the line that other packets request. This worsens the situation and can now also affect packets that would not even traverse the router where the initial contention occurred resulting in secondary congestion [19].

A similar problem occurs if a pair of communicating nodes is not balanced regarding their respective traffic generation and consumption rates. If a source node generates and injects messages faster than the sink node can consume the messages the result is backpressure. This causes congestion much in the same way contention does.

Both congestion caused by contention and backpressure have a severe impact on the performance of a NoC and should, therefore, be avoided. Congestion is avoided or at least mitigated by *congestion control* techniques whereas backpressure is avoided by end-to-end *flow control*, both of which are briefly discussed in the following paragraphs.

Congestion control can be divided into two categories: congestion control with and without resource reservation [19]. Congestion control without resource reservation can be

achieved by dropping packets, avoiding congested links or routers by actively rerouting traffic around them, or limiting the flit injection rate into the NoC. Dropping packets is usually not used in NoCs since lossless communication is typically assumed. Furthermore, dropping packets can lead to an even higher congestion, thereby reducing the achievable utilization of a network [62]. However, some bufferless NoCs use packet dropping on contention (e.g., [31]). Another approach drops only individual flits of a packet—thereby compromising the integrity of a packet—and later recovering lost information with coding techniques [63].

When dynamic routing is used, routers can actively avoid links that are congested or on which contention occurs and forward incoming packets to other output ports instead (in order to avoid congestion). An example for this approach is Nostrum which uses deflection routing [26]. In general, deflection routing—or "hot-potato" routing—is commonly used in bufferless NoCs [64, 65]. Furthermore, if knowledge about congestion is shared with neighboring routers and propagated through the NoC it is possible to avoid congested routers entirely (e.g., [66, 67]). Lastly, congestion can be limited by limiting the flit injection rate of the NIs [68, 69]. This can be done either statically or dynamically based on measurements done with monitors in the NoC (e.g., [70, 71]).

The described congestion control techniques without resource reservation are inherently reactive but cannot avoid congestion to occur in the first place. In order to avoid congestion it is necessary to reserve resources in advance. Some packet switched NoCs with resource reservation use global scheduling [72], looped containers [26], or reserve resources on-demand [73].

Contention and congestion are problems that typically occur in packet switched NoCs but are (mostly) not an issue in circuit switched and TDM NoCs. TDM—and by extension circuit switching in general—can be considered the most rigid form of congestion control, especially if static scheduling is used. Once a communication channel is established, contention is completely eliminated from the NoC which is why it is easily possible to give QoS guarantees in TDM NoCs. However, contention can occur in TDM NoCs that dynamically set up and tear down connections at runtime in case two nodes want to establish a communication channel using the same resources at the same time.

Flow control determines how resources are allocated when packets or flits traverse the NoC. It can be viewed as either a problem of resource allocation or of contention resolution [52]. Flow control is applied on two layers: on data link layer for flow control between the routers and on network or transport layer for end-to-end flow control.

Data link layer flow control allocates router resources and ensures that a receiving router (or NI) has enough free buffer space to store the flit(s) sent. Three typical link layer flow control protocols are STALL/GO, T-Error, and ACK/NACK, which all vary in complexity, performance, and level of fault-tolerance [19, 18]. A very simple one is the STALL/GO protocol. It only uses two control wires in addition to the data wires: one going forward and signaling if a flit is valid and one going backwards and signaling whether the buffer of the receiving router is full and the transmission must be stalled.

Data link layer flow control cannot avoid contention or congestion, but is used to *handle* contention by ensuring that no data is lost. A special case are TDM NoCs which

do not require control flow on the link layer since resources are reserved beforehand and it is guaranteed that each flit will be forwarded in the next cycle.

End-to-end flow control on network or transport layer is used to eliminate congestion caused by backpressure. This can, e.g., be achieved with, credit-based flow control of sliding windows [19, 25]. Examples using credit-based end-to-end flow control are SPIN [74], QNoC [75], and the Æthereal NoC for the packet switched BE traffic [40].

Although TDM NoCs do not require link layer flow control they typically *do* need some form of flow control on the network or transport layer in order to ensure that enough buffer space is available on the receiver's side. Otherwise, flits would be dropped in the receiving NI if the receiving buffers are full. However, if it can be ensured by design that the traffic generation and injection rate of a sending node never exceeds the consumption rate of a receiving node it would be possible to refrain from using flow control.

Quality of Service

An issue that is directly connected to congestion and flow control—as well as the overall NoC architecture and applied routing algorithms—is QoS. QoS is a common network term that denotes network services that are provided by a network or required by applications. Typical QoS metrics are bandwidth (or Guaranteed Throughput (GT)), latency, latency variation (jitter), or loss rate [76, 19]. The provided or required QoS can be categorized, e.g., by average values or worst-case bounds.

Different application types require different levels of QoS. Video or audio streams, e.g., typically require high bandwidth and low jitter, but can tolerate long latencies. Data accesses due to cache misses, on the other hand, require low latencies but are more lenient on jitter [77]. Traffic from different types of applications can be categorized in different *traffic classes* with common QoS requirements (as, e.g., done in [77] and [75]). NoCs, on the other hand, typically offer at least two different *service classes*, each providing different QoS guarantees (two service classes typically means one Guaranteed Service (GS) class and one BE class). Traffic of the different traffic classes can then use the service class that best serves their requirements.

Most publications on NoCs refer to their offered service classes as traffic classes. Hence, for the remainder of this thesis and unless otherwise specified *traffic class* denotes a *service class* provided by a NoC.

The number of traffic classes that a NoC *should* provide is, in general, up to the developer and highly depends on the intended use case of the system and the number and types of applications that will communicate over the NoC. QNoC [75], e.g., provides four traffic classes, whereas the Æthereal NoC [40] offers just two (i.e., in hybrid configuration). Providing more than two traffic classes can lead to problems itself since, naturally, only one *highest* priority can exist making all other classes BE relative to this class which, in turn, makes it difficult to give (hard) QoS guarantees. Furthermore, mechanisms should be employed to avoid complete starvation of the lower traffic classes unless absolutely necessary (i.e., the entire bandwidth is used by applications with higher QoS requirements).

In general, giving hard (100%) QoS guarantees is only possible using resource reservation [19]. Relatively few NoCs have been implemented providing hard QoS. Examples include the Æthereal NoC [40] and Mango [78]. When hard QoS guarantees cannot be given it is only possible to give statistical guarantees assuming a particular average traffic load.

QoS typically assumes a fault free network which is why fault-tolerance is usually not considered when implementing QoS guarantees. Hence, mechanisms for fault-tolerance will be discussed in a separate section.

Clocking

In order for a NoC to function properly, the transmission of flits between the routers and NIs of the NoC must be synchronized. For simplification, a globally synchronous NoC has implicitly been assumed in the previous sections and will be assumed for the remainder of this thesis unless otherwise specified. However, the clock related challenges when implementing a NoC are discussed in this section and typical approaches to overcome these challenges are presented.

A fully synchronous system requires a single global clock which can only be realized with a global clock tree. However, with shrinking feature sizes and a growing complexity of modern SoC designs these clock trees become very large and complex, and account for a considerable amount of the total power dissipation [79]. Therefore, a globally synchronous system is often not feasible or desirable. A principle to overcome this issue are Globally Asynchronous Locally Synchronous (GALS) systems in which synchronicity is maintained only locally in different clock domains [80]. This, of course, causes issues for the on-chip communication and specifically the NoC.

A GALS system may use a global clock to derive local clocks to maintain local synchronicity. Two of these local clock domains can have the same frequency but the phases of the derived clocks typically differ, making them *mesochronous*. This phase difference, however, is constant in a mesochronous system. If the local clocks are not derived from a global clock but instead created locally, the system is *plesiochronous*. In this case the frequencies of two clocks will never be exactly the same which leads to an unknown phase difference that shifts over time. SoCs are typically not plesiochronous since it is more convenient to derive local clocks from one common clock. Therefore, this case will not be further considered here.

Communication in a GALS system can be challenging. Different approaches, most notably one using a self-timed ring, have been proposed in [81]. If a fully synchronous NoC is not feasible, a mesochronous or asynchronous NoC can be used, e.g., by using tokens between the routers. This is even possible for TDM NoCs as has been showed, in [43, 44, 45].

Faults in NoCs

As mentioned in a previous section, QoS implementations focus on metrics such as bandwidth and latency but typically assume a fault free NoC. However, as with any

other part of a SoC, faults can and do occur, especially with the ever shrinking feature sizes of modern systems. Therefore, this section gives an overview over possible faults that must be considered in a NoC and discusses the relevant terminology. A more comprehensive discussion is presented in [82].

There are different *physical failure mechanisms* that can cause a malfunction at the lowest layer of a NoC, i.e., in a transistor or wire, leading to a *fault*. If this fault changes information in comparison to the fault-free case it manifests in an *error*, otherwise it is *masked*[8]. If an error propagates to the output of a hardware module it is said to have cause a *failure*. Similar to a fault, an error can be masked in which case it does not cause a failure. At the next higher level of abstraction a failure is seen as a fault which, again, may lead to an error which may lead to a failure.

The set of faults that is considered in a model constitutes the *fault model*. A very common fault model in digital circuits is, e.g., the *stuck-at* model that considers the outputs of logic gates to be stuck at either '1' (stuck-at-1) or '0' (stuck-at-0). While low-level fault models often consider the physical failure mechanisms that initially cause a fault, high-level fault models typically consider functional faults in individual hardware (sub-)modules. Higher level fault models are often necessary in complex systems in which considering all possible physically caused faults is not feasible. As an example, a high-level fault model for a NoC could consider entire routers to be faulty, whereas fault models with a lower abstraction level could consider faults in individual router ports or other sub-modules.

Faults are typically categorized as being either *transient*, *intermittent*, or *permanent* faults [83]. Transient faults occur randomly for one or multiple cycles and are often caused by radiation. Intermittent faults are similar to transient faults in that faults only occur temporarily. Three criteria are proposed in [83] in order to distinguish intermittent faults from transient faults: 1.) "an intermittent fault occurs repeatedly at the same location", 2.) "errors induced by intermittents tend to occur in bursts when the fault is activated" 3.) "replacement of the offending circuit removes the intermittent fault". Permanent faults, as the name suggests, permanently occur under the same conditions. They can either be caused by an electrical short or broken connection—causing stuck-at faults—or by delay causing setup or hold timing violations (even if these timing violations only occur under certain conditions, e.g., a series of two specific words).

The physical failure mechanisms that cause faults on the lowest layer are not unique to NoCs but instead affect all modern digital circuits. Therefore, they are only briefly described here. The most notable mechanisms are radiation, electromagnetic interference, electrostatic discharge, and aging.

Radiation induced faults—also called *soft-errors*—are a typical example for transient faults and are mostly caused by alpha particles and cosmic rays creating high-energy neutrons in the atmosphere [83, 84, 85]. These particles can cause bitflips in DRAM or SRAM cells—called Single Event Upset (SEU)—or a wire or logic gate to change its logic level—called Single Event Transient (SET). The probability for this to happen

[8] An example for a masked fault would be if a fault causes a wire to propagate '0' but the information to propagate is '0' anyway.

depends on the critical charge that is necessary for a bitflip. Hence, shrinking feature sizes and lower supply voltage increase the probability of SEUs [86]. It has been shown that the probability of SETs in logic elements increases even faster with scaling [87, 88].

Electromagnetic interference is typically caused by *crosstalk* between long parallel wires [89]. Due to capacitive and inductive coupling effects the long wires affect each other and can cause increased signal delay or damped voltage oscillations. Naturally, NoCs are very susceptible to crosstalk on the links between the routers. Similar to radiation induced faults, crosstalk becomes increasingly problematic with technology scaling [89, 90]. Electromagnetic interference causes intermittent or permanent faults, depending on whether or not a specific condition will always cause a fault.

Electrostatic discharge can cause strong electric currents in a SoC. These currents can, e.g., cause oxide breakdown in CMOS transistors or fusing of wires, thereby causing permanent faults. With technology scaling lower currents suffice to cause damage and the problem becomes harder to analyze and control [91]. However, the internal circuitry of a SoC is typically not affected since protection against electrostatic discharge is placed at the pins of the chip.

Aging causes the performance of a CMOS circuit to degrade over time. This is due to multiple effects such as hot carrier injection and electromigration, to only name two [92, 93]. When the performance of the circuits degrades, aging typically causes intermittent faults at first and later permanent faults.

Another effect that plays a role in the probability of faults occurring is *process variation* (or *variability*). Due to natural variation of the dimensions of wires and transistors no two devices will ever have the exact same physical properties. Process variation affects the four aforementioned failure mechanisms and leads to differences in the probability of faults. If, e.g., the charge that a DRAM cell holds is slightly smaller than intended the critical charge that causes a SEU is smaller, thereby increasing the probability of a fault. Process variation becomes more problematic with technology scaling and a significant amount of effort is spent to keep it as small as possible [94, 95].

The different possible faults that can occur in a NoC must be addressed by suitable mechanisms in order to implement fault-tolerance. Various approaches have been proposed in the past, which will be discussed in Section 2.3.2.

2.3 Related Work

The previous sections gave an introduction to on-chip communication in general and NoCs in particular, thereby laying the foundation for the remainder of this thesis. Different NoC principles as well as common terminology were introduced and typical challenges were discussed. Based on this introduction, this section discusses related work and state-of-the-art research to approach these challenges.

Three different topics are discussed and integrated in this thesis that are typically considered separately: implementing QoS in NoC, fault-tolerance in NoC, and mapping strategies in NoC. Therefore, the related work in these field is presented in the following in designated sub-sections.

2.3.1 QoS in NoCs

Many different approaches exist to implement QoS guarantees in NoCs in general and NoCs for mixed-critical systems in particular. Most of these approaches consider packet switched NoCs, often using VCs. However, some also consider circuit switched or hybrid NoCs.

A wormhole NoC protocol for mixed-criticality systems is proposed in [28]. Two levels of criticality are defined and credit-based flow control with VCs is used. When contention occurs and threatens to cause a missed deadline for critical traffic, the NoC enters a state of graceful degradation in which only critical traffic is served. Ahmadian et al. describe an extension layer for existing NoCs in [96, 97]. Two different traffic types are differentiated and time-triggered guardian windows are used to ensure absence of interference between the two traffic types. This, however, comes with significant source rate limitation to avoid congestion. A packet switched router design for mixed-critical systems is proposed in [98]. The approach uses $n + 1$ VCs: n for critical traffic flows that share a link and 1 for BE traffic. Round robin arbitration is used between the n critical VCs while priority arbitration is used between critical and BE traffic.

Millberg et al. use looped containers to create virtual circuits for GS traffic in a packet switched NoC [26]. Routes for GS traffic are determined at design time but their bandwidth can be adjusted at runtime. A NoC design providing service guarantees and targeting GALS systems is the MANGO clockless NoC [37]. Here, VCs are used to separate different GS connections.

Tobuschat and Ernst propose a novel approach on traffic prioritization in [99]. They use four VCs: three for BE traffic and a shared one for GT traffic. BE traffic is given priority over GT traffic as long as no deadline is missed. This increases the achievable BE injection rate. Kostrzewa et al. introduce a Resource Manager (RM) to a packet switched NoCs to ensure QoS [73]. The RM distributes link bandwidth and access to shared resources to the applications of a system, ensuring the requirements by critical applications are met. The approach is further developed to increase NoC utilization and improve DRAM access [100, 101].

QoSinNoC, a framework to find NoC layouts matching given system-level requirements is presented by Avramenko et al. in [102]. A focus lies on enabling the utilization of NoCs in certifiable systems for avionics. They divide the system into critical and non-critical application domains. This division, however, is slightly relaxed by allowing non-critical traffic to pass through critical domains as long as different router ports are used. Picornell et al. apply a TDM schedule to a packet switched NoC to ensure that only one node in the NoC injects a packet in each cycle [103]. In combination with delay cycles on selected paths, they ensure that no contention can occur. This, however, comes at the price of low overall injection rates, especially with increasing NoC size. A lightweight real-time NoC with focus on FPGA implementation is presented in [104]. A bufferless NoC with deflection routing is used. By prioritizing deflections and regulating the injection rate an upper worst-case latency is guaranteed. The approach is further improved in [105].

Two different approaches apply TDM scheduling to VCs in a packet switched NoC to implement traffic isolation while keeping the maximum latency low. SurfNoC achieves this by scheduling application domains in a pipelined fashion in X and Y directions [106]. PhaseNoC further reduces the latency by removing delay cycles when switching dimensions but has rigid limitations regarding the number of application domains [38]. An improved version allows for more flexible scheduling [107].

Packet switched NoCs must typically rely on the use of VCs to implement QoS—which requires larger routers—or significantly limit the injection rate. The number of VCs has a particularly severe impact on the worst-case latency, as shown in [29]. This usually makes it necessary to use some form of priority scheduling to give QoS guarantees to critical traffic. Many approaches are very limited in the number of criticality levels they can handle. In many cases, only two different levels (critical and non-critical) are available but traffic streams from different critical applications can still interfere with each other.

A different approach to implement QoS is to use a TDM NoC, which was first proposed and then further developed by Goossens et al. [40, 108]. The Æthereal NoC defines different configurations, either as pure TDM NoC with static scheduling or as hybrid NoC with additional packet switching for BE traffic. Two other approaches based on the Æthereal NoC are aelite and dAElite, which both only use TDM traffic [109, 46]. aelite focuses on scalability and uses mesochronous or asynchronous links. dAElite describes multicasts and connection setup in TDM NoCs.

With Argo, Kasapaki et al. introduce a hard real-time NoC for GALS systems [43, 44]. Based on Argo, an extension to allow reconfiguration for mode changes at runtime is proposed in [110]. XNoC, a TDM NoC with a predictable path (de-)activation time is presented in [111]. A non-intrusive reconfiguration process is proposed that uses two slot tables that can be switched between within one clock cycle. Furthermore, a distributed control plane is proposed to improve scalability.

A hybrid NoC combining SDM and TDM with packet switching is presented in [47]. The NoC consists of a packet switched and a circuit switched sub-network which do not share links. The circuit switched sub-network is used for real-time traffic and provides several parallel links (SDM) each with their own TDM schedule in order to increase path diversity and improve resource usage. Other hybrid TDM and packet switched NoCs typically focus on improving power efficiency or network utilization rather than implementing QoS (e.g., [49, 112]). These NoCs schedule TDM connections dynamically on demand but cannot guarantee that a requested connection is successfully set up. This makes it impossible to give (hard) real-time guarantees since this would require the critical connections to be reserved and mapped at compile time [113].

While different approaches exist to implement QoS for critical traffic in mixed-critical systems, the approach described in Chapter 3 is, to the best of the author's knowledge, the first one that not only provides hard real-time guarantees and strong traffic isolation but also fault-tolerance to critical traffic streams.

2.3.2 Fault-Tolerance in NoCs

In general, implementing fault-tolerance requires some form of redundancy: *spatial* or *modular* redundancy, *temporal* redundancy, or *information* redundancy. Spatial redundancy is achieved by providing redundant components to replace faulty ones, temporal redundancy is achieved by reexecution of a faulty operation or transmission, and information redundancy is achieved by adding information that can be used for error-correction. Not all forms of redundancy are equally well suited to address transient, intermittent, or permanent faults [82]. Temporal redundancy is a convenient way of addressing transient or intermittent faults but can, by itself, not cope with permanent faults. Spatial redundancy, on the other hand, is well suited for permanent faults. Information redundancy can help with all types of faults and is often combined with another form of redundancy. Particularly information redundancy in form of coding is often used for fault-detection (e.g., parity bits).

Radetzki et al. give a comprehensive overview over different methods to achieve fault-tolerance in NoCs and categorize the different approaches into those working on the data link, network, or transport layer [82]. Hence, this section will only cover a few selected common approaches and the interested reader is directed to this paper for a more in-depth overview.

A typical approach on the link layer to both detect and correct transient faults is temporal redundancy in form of multisampling (e.g., [114, 115]). The data is sampled multiple times with a time interval. In combination with Triple Modular Redundancy (TMR) single faults can not just be detected but also corrected. Another approach using temporal redundancy are hop-to-hop retransmissions between the routers (e.g., [116, 117]).

A typical approach to implement fault-tolerance with information redundancy is coding (e.g., [118, 119]). Coding can either be used for the entire packet or just for the header flit or control signals in order to reduce the cost [120]. Furthermore, coding can be used to avoid or reduce crosstalk on the link wires [121].

A widely used form of spatial redundancy is TMR which can be applied at various points in a NoC. TMR can effectively mask single errors but comes with a hardware overhead of more than 200%, including the voter. Therefore, TMR is typically only used for critical parts such as the control signals (e.g., [116, 122]). In other approaches, the physical links are duplicated ([123]), or parts of the router logic are hardened with TMR ([115]).

An approach to use information redundancy on the network layer is *stochastic communication* [124]. Packets are duplicated and—with a certain probability—sent over different paths, potentially *flooding* the entire NoC. An improvement is *random walk* where only a predefined number of duplicates is created [125].

Spatial redundancy on the network layer is typically given by design, especially in NoCs with 2D mesh or torus topology. Enabling this redundancy to be used for fault-tolerance is, therefore, mainly a matter of utilizing dynamic routing algorithms while considering faults in the NoC. An comprehensive comparison of several such fault-tolerant routing algorithms is presented in [82].

On the transport layer, temporal redundancy can be achieved with retransmissions—similar to the link layer—for end-to-end fault-tolerance [116]. This approach alone, however, is only suitable for transient faults and only allows to determine that a fault has occurred somewhere along the path without revealing where exactly it is located. Lehtonen et al. analyze different coding methods for error correction that can be applied to achieve fault-tolerance on transport layer in [126]. These coding methods can also handle a single bit permanent fault.

An approach to implement spatial redundancy on the transport layer is proposed by Murali et al. [127]. They use a multi-path routing strategy to address permanent faults in the NoC. Fault-tolerance of the NI—a topic often overlooked—is addressed by Fiorin et al. [128]. Error detection and correction codes are used to protect lookup tables, Finite State Machines (FSMs), and FIFOs, while TMR is used for remaining components. A different approach is taken in [129] where each NI has two local ports—a primary and a secondary one—which allows to compensate for a permanent fault on the primary port.

A common method to find faulty components not at runtime but instead at system start time is the Built-In Self-Test (BIST). Different approaches of using BIST for the NoC have been presented (e.g., [130, 131, 132]). However, by design a BIST is not suited to deal with transient or intermittent faults, or permanent faults occurring at runtime.

Many of the approaches listed focus on packet switched NoCs but some are applicable in other NoC types as well. However, most of the approaches do not consider traffic isolation between critical and non-critical traffic and, to the best of the author's knowledge, none of the approaches considers hard real-time constraints.

2.3.3 Task and Channel Mapping in NoCs

Mapping tasks on NoC-based MPSoCs has been an important research topic for about two decades. For instance, Lei and Kumar proposed a two-step genetic algorithm to map task graphs to a NoC architecture in [133]. They consider a packet switched NoC and the goal of the mapping is to minimize the execution time of the tasks.

A congestion-aware dynamic task mapping based on heuristics using channel and path loads has been proposed in [134]. The NoC considered uses packet switching and the goal is to reduce congestion throughout the NoC to reduce link load and packet latency. Another dynamic runtime task mapping heuristic that is not just congestion- but also energy-aware is presented in [135]. Applications are represented by task graphs and each task is mapped as close to its parent task as possible while also considering load balancing between different NoC links.

An example of mapping research including fault-tolerance requirements can be found in [136]. A two-stage mapping scheme is proposed. First, a static genetic algorithm is used to find multiple mapping solutions, then, an optimal mapping solutions is selected at runtime.

These works propose different task mapping methodologies for a number of different purposes. However, none of these works considers mapping in TDM NoCs.

Lu and Jantsch discuss the configuration of virtual circuits in TDM NoCs in [137]. They formulate the configuration problem and discuss algorithms to find valid TDM

configurations that fulfill given requirements (e.g., bandwidth or latency). Static slot tables that are configured at design time are assumed.

In [138] integer linear programming is used to find feasible mappings for TDM channels in a NoC-based system. Reconfigurations are possible at runtime by means of an overlay network. A centralized resource allocation approach for TDM NoCs is presented by Chen et al. [139, 112]. The approach uses a centralized hardware accelerator to quickly find connections base on a trellis graph which allows to search all possible connections in parallel. The assumed used case is a system in which TDM connections are set up and torn down frequently at runtime.

A mapping algorithm for applications in a multi-FPGA system is described by Hironaka et al. [140]. The different FPGAs communicate via optical fibers employing TDM. However, the work does not consider the actual slots reserved.

While different task and channel mapping approaches exist, none of these approaches consider efficient mapping of TDM connections while at the same time maximizing the available remaining bandwidth for BE applications, which is one of the contributions of this thesis.

2.4 Protection Switching

Protection switching is a principle for fault-tolerant communication that was developed for circuit switched connection-oriented layer networks and specifically SONET/SDH [141, 142]. The principle will be utilized throughout this thesis and is, therefore, introduced and described in this section.

The basic idea of protection switching is to define a physically disjoint alternative, or backup, path for each connection. In case of a fault on one path, the other path is still fully operational. Three different protection switching versions are commonly known: 1:1, 1:n, and 1+1 protection. The principle of these different protection switching versions is depicted in Figure 2.15. The following paragraphs give an overview over the three versions.

1:n Protection

With 1:n protection—shown in Figure 2.15 on the left—there is one alternative path that is shared between n communication channels. As a consequence, this protection switching version can compensate one faulty path between the n connections. For the remainder of this thesis the actively used path will be referred to as the *primary* path, whereas the alternative and currently unused path will be referred to as the *secondary* path.

When a fault occurs on a primary path both sender and receiver must switch to the secondary path. This implies that *a)* a fault must be detected on the receiver's side (or along the path)[9], and *b)* there must be a feedback channel in order to perform the switch

[9] This, of course, is generally necessary to achieve fault-tolerance, which is why it will not always explicitly be mentioned for the remainder of this thesis.

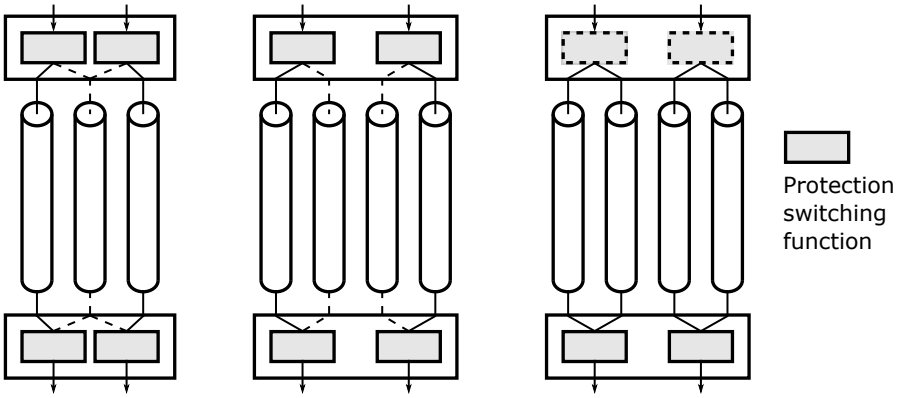


Figure 2.15: Concept of 1:n (left), 1:1 (middle), and 1+1 protection (right)

on the sender's side. Furthermore, the secondary path must be checked to ensure it is fault free even when it is not being used for data transmission.

A superset of 1:n protection is m:n protection where n connections share m secondary paths. This way, it is possible to compensate for up to m faults between n connections.

Protection switching naturally comes with a certain hardware overhead. Besides the additional physical path and the feedback channel, some administration logic and synchronization is needed to implement the switching from the primary to the secondary path.

1:1 Protection

With 1:1 (or $(1{:}1)^n$) protection, each communication channel has its own designated secondary path. This means, that each connection can tolerate a fault. The concept of 1:1 protection is depicted in Figure 2.15 in the middle. Just as with 1:n protection, the switch from the primary to the secondary path needs to be synchronized between the sender and the receiver and, therefore, a feedback channel to the sender is needed. For both 1:1 and 1:n protection the secondary path may be used to carry extra traffic (unprotected) for as long as the primary channel is fully operational.

1+1 Protection

With 1+1 protection, both paths are actively used for data transmission, as shown in Figure 2.15 on the right, meaning both paths act as primary paths. This has the advantage that no feedback channel is required since the switch is only done on the receiver's side. The protection switching function on the sender's side is essentially no longer necessary, as indicated in Figure 2.15. The data simply needs to be forwarded to both paths.

As with 1:1 protection, each connection can tolerate one fault. In comparison to 1:n and 1:1 protection, 1+1 protection is simpler to implement due to its lower complexity. However, actively using both paths has the disadvantage that the alternative path cannot

be used for uncritical extra traffic which is possible with both 1:1 and 1:n protection as long as no fault occurs.

2.5 Discussion on Functional Safety

As mentioned in Chapter 1, the research presented in this thesis is supported by the ARAMiS II research project which researched the use of multicore systems in safety- and mixed-critical applications, specifically in the automotive, railway, avionics, and industry domain. Depending on where a safety-critical system is deployed, a fault or otherwise unintended behavior in such a system can potentially have catastrophic consequences including loss of life or severe and long lasting environmental effects (e.g., in utility facilities such as nuclear power plants). Hence, the highest emphasis in a safety-critical system is put on *functional safety* while cost and power efficiency are often of lower priority[10]. This, however, begs the question: what is considered *safe*?

The truth of the matter is that there is no definitive answer to this question because there cannot be. No system can be made 100% *safe* in the sense that it can never fail and will always fulfill its task no matter the conditions. For instance, TMR is a proven and common way of implementing fault-tolerance—which is usually required for a system to be considered safe—but it can *only* compensate for faults in a single one of the replicated modules. Furthermore, fault-tolerance is typically given on an end-to-end basis—e.g., a specific processing chain—but there is often an instance *before* or *after* this processing chain that needs to be considered as well. In the case of TMR this includes the voter module, the original source of the data being processed, and the following instance that handles the processed data. Hence, whether or not a system is considered *safe* depends on the frame of reference that is defined, the fault model that is assumed, and whether the residual failure chance is considered to be acceptable.

This is where safety standards become important. Safety standards give guidelines to help determine the level of confidence in a system necessary for it to be considered *safe*. This is, in essence, a trade-off between the cost of implementing a certain level of confidence and the cost that would come with a—potentially catastrophic—failure of that system. As a consequence, the more harm a system failure would or could cause, the more effort—and money—is spent on minimizing the chance of that system to fail. This is the reason why, to this day, the safety requirements in the avionics domain are much higher than in the automotive domain.

A commonly known safety standard is the ISO 26262 which regulates functional safety for road vehicles [13]. The standard defines functional safety as "absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems". What *is* and *is not* "reasonable", once again, depends on the frame of reference which can and has changed over time. In this context, the research presented in this thesis naturally cannot by itself guarantee a system to be *safe*. Instead, it aspires to contribute to the process of

[10] Nevertheless, these costs are anything but irrelevant which is why a considerable amount of research, including this thesis, is dedicated to ensuring safety with as little hardware and power overhead as possible.

pushing the boundaries of minimizing the residual failure chance of safety-critical systems while keeping the required hardware and power overhead within reasonable limits.

2.6 Summary

Continuous technology scaling—as predicted by Moore's Law—has lead to various challenges in chip design such as the "power wall" and the "designer productivity gap". These challenges made a paradigm shift necessary and lead to the development of MPSoCs. This type of SoC can incorporate a vast number of different components—such as CPUs, DSPs, or custom accelerators—which all need to communicate with each other. However, traditional means of on-chip communication—most importantly buses—do not scale well and are, therefore, not suited to meet the communication demands in modern MPSoCs.

To overcome this communication bottleneck, new concepts for on-chip communication were proposed which lead to the advent of NoCs. As the name suggests, a NoC is essentially a miniaturized version of a computer network in which messages are forwarded by routers until they reach their destination. One of the most popular NoC topologies is the *2D mesh*. The reason is that in a 2D mesh the number of links and, thereby, the number of possible parallel transactions as well as the total available bandwidth scales up with the number of nodes communicating via the NoC.

There are two different general types of NoCs: packet switched and circuit switched NoCs, each with different advantages and disadvantages. In circuit switched NoCs resources are reserved before being used for communication to avoid contention. In packet switched NoCs information is sent in packets which can interfere with each other leading to contention that needs to be dealt with. A special sub-category of circuit switched NoCs are TDM NoCs in which resources are only reserved for a certain number of time slots. This still ensures freedom of contention but does not fully block a given resource—i.e., a link—for other traffic. Packet switched NoCs typically offer greater flexibility but providing communication guarantees—such as latency or bandwidth—is relatively difficult. Circuit switched NoCs, on the other hand, are well suited to provide guarantees and are typically cheaper to implement. However, they lack the flexibility that packet switched NoCs offer. To overcome the weaknesses and combine the strengths, hybrid NoCs combining the two types can be an option but are typically more complex.

In general, NoCs face similar challenges as larger scale computer networks do—e.g., congestion and flow control—but also face some new challenges such as clocking issues. Since NoCs can account for a considerable amount of chip area and power consumption it is generally desirable to keep the complexity relatively low. This means that, often, solutions to these challenges used in larger scale computer networks are not directly applicable in NoCs which is why NoC research was and is a very active field. An overview over related work in two of the, arguably, most important fields was given in Section 2.3: QoS in NoCs and fault-tolerance in NoCs. It was concluded that there is currently now work that adequately combines these two fields.

Based on a solid understanding of on-chip communication via NoCs and the relevant related work, the following chapter presents the main contributions of this thesis, most notably an approach to not only provide fault-tolerant but also hard real-time capable means of communication to safety-critical applications in a mixed-critical MPSoC.

3 Protection Switching Concept for Efficient Fault-Tolerance in NoC

This chapter presents the main contribution of this thesis: the concept for a NoC providing both fault-tolerant and hard real-time capable means of communication to safety-critical applications in a mixed-critical system. The concept is based on the combination of a hybrid TDM and packet switched NoC with protection switching. Three different protection switching versions are considered and compared regarding their worst-case latency and expected hardware overhead. A proof of concept with cycle-accurate simulations of a hardware model of the NoC is presented and evaluated. Furthermore, the challenge of mapping the critical applications and their communication channels to the NoC is discussed. Different mapping strategies are developed which are designed to utilize the NoC resources most efficiently in order to maximize the available bandwidth for the BE traffic. These strategies are then later evaluated in Chapter 5. Parts of this chapter have been published in [14] and [17].

3.1 Target System and Use Case

There is a vast number of different systems—specifically MPSoCs—and applications that require scalable means of on-chip communication between different building blocks such as processing cores, memory, or I/O. However, different systems can have entirely different requirements regarding throughput, latency, or robustness. Hence, it is important to define the target system and use case that the NoC design proposed in this thesis is intended for, before designing it.

Safety-critical systems in avionics traditionally rely heavily on TMR (e.g., [143]). For an aircraft, the hardware and power overhead is well justifiable given that hundreds of people all rely on the same system to operate safely. With the increasing number of safety-critical systems in automotive on the other hand—heavily driven by the development of autonomous driving—the immense cost of TMR is not as easily justifiable. The reason is that the additional cost per passenger is much higher than in avionics. This does not just mean higher costs for the hardware itself but also a much higher power consumption which directly translates to higher fuel consumption and, thereby, CO_2 emissions which—with respect to the ongoing climate crisis—should be kept as low as possible. One way of doing this is by consolidating multiple functions in a single MPSoC instead of, e.g., using multiple SoCs. This not only means that different applications of potentially different levels of criticality now share at least parts of the same resources—making it a mixed-critical system—but also that fault-tolerance mechanisms must now be built into the MPSoC itself instead of using multiple SoCs in parallel. All this directly

affects the considerations necessary when designing one of the largest shared resources of a modern MPSoC: the NoC.

As stated in Chapter 1, the NoC is intended for safety- and mixed-critical systems in which at least the critical applications must not fail at runtime. Furthermore, although in principle applicable in other domains such as avionics, the NoC is designed with automotive domain and its current and upcoming requirements in mind. In this environment, it can be assumed that the critical applications are known either at system design time or—at the very least—at compile time of the system's software. Furthermore, it is assumed that the critical applications are static in nature in the sense that they are not dynamically started or stopped during runtime (i.e., while a vehicle is in motion). Moreover, the behavior of the critical applications—specifically their communication patterns and both bandwidth and latency requirements—is known at compile time and does not change at runtime[1].

On the other hand, it can be assumed that BE applications might be dynamically started and stopped on demand at runtime[2]. Their number, exact behavior, and communication patters are not necessarily known at compile time. Therefore, BE applications can be executed to the system's ability, but it cannot be guaranteed that resources are always available.

In summary, the target system is a mixed-critical NoC-based MPSoC with safety-critical applications that do not dynamically change at runtime and whose behavior is predictable and known at compile time, and BE applications that can dynamically be started and stopped at runtime and whose behavior might not always be known in advance.

3.1.1 Requirements

With the target system defined it is possible to derive the requirements which the NoC must fulfill.

Multiple applications run on the same platform which means that the NoC must support multiple—ideally an arbitrary number of—critical data streams in parallel. Each critical data stream must be isolated, not only from the other critical data streams but also the BE traffic in order to be protected against faulty or malicious behavior of other applications running on the same system. Furthermore, the NoC must provide fault-tolerance for the critical communication and ensure that all data sent is delivered uncorrupted and in-order. Additionally, the critical communication must provide deterministic bandwidth and latency guarantees in order to enable the implementation of hard real-time applications in the system. Last but not least, the NoC should provide sufficiently flexible communication for BE applications that are started and stopped at system runtime.

In summary, the requirements are as follows:

[1] Or *if* the behavior changes at runtime then it does so in a way that is known beforehand and can and has been accounted for at compile time.

[2] Examples could be third party applications that passengers might want to use during transit or generally a vehicles infotainment system.

- Multiple concurrent critical data streams.
- Isolation between different critical traffic streams.
- Isolation between critical traffic and BE traffic.
- Fault-tolerance of critical communication.
- In-order delivery of critical traffic.
- QoS guarantees for critical traffic (i.e., deterministic bandwidth and latency guarantees).
- Flexible communication for BE applications.

3.1.2 The Fault Model

Whenever fault-tolerance is discussed it is important to define the fault model that is assumed. As described in Section 2.2.7, the fault-model is the set of faults that is considered in a model. Fault models can consider physical faults on the lowest level of a system, or they can be more abstract and consider, e.g., entire modules to be faulty without considering the original cause of that failure.

Rambo et al. present a comprehensive Failure Mode and Effects Analysis (FMEA) for a packet switched NoC for mixed critical systems in [144]. The NoC presented in this thesis is a hybrid TDM and packet switched NoC which uses TDM for critical traffic. In contrast to the work presented in [144], only the faults that can corrupt the critical TDM traffic are of interest. This means, the biggest threat—other than faults affecting the data path of the TDM flits—are faults in the slot tables and the TDM cycle counters, which must both be addressed by appropriate measures. However, a full FMEA is not in the scope of this thesis.

For simplification, only faults that corrupt the flits of a TDM channel are considered in the scope of this thesis, but not the status flags (cf. Section 4.2.1.1), slot tables, or cycle counters. Common mode faults affecting power supply or clock distribution are also out of the scope of this thesis. Furthermore, it is assumed that faults affecting the flits can be detected through suitable measures (as will be discussed in Section 4.2.1.1). Beside these simplifications, it is assumed that faults can affect any parts of a TDM path between the NIs (i.e., wires, registers, and logic) and that they can be transient, intermittent, or permanent in nature. This means, the fault model essentially considers faults on an abstract level on the transport layer (end-to-end on a TDM path). However, an essential part of the approach presented in this thesis is that the hybrid NoC router is designed in a modular fashion with independent sub-modules, meaning that no single fault on the physical layer—e.g., stuck-at faults—can affect multiple input or output ports nor the wiring between them. In that regard, the fault model considers faults on individual links and input/output ports of the NoC routers (i.e., faults on the data link layer). Lastly, it is assumed that faults occur only sporadically and that the chance of two faults occurring on different TDM paths at the same time is negligible. For

completeness, possible approaches to harden the slot tables, cycle counters, and status flags against faults will be briefly touched on as future work in Section 6.1.

3.1.3 Fault-Tolerant and Fail-Operational Systems

A term that has gained increased attention in recent years, specifically in the automotive domain, is that of the *fail-operational* system. This begs the question what exactly qualifies a system as fail-operational and how it differentiates from a fault-tolerant system. Unfortunately, there is no clear answer to this question as different works use different interpretations of the term and often *fail-operational* and *fault-tolerant* are treated as synonyms. Therefore, a brief discussion of the two terms is presented here, followed by a definition of how the terms are used throughout this thesis.

Generally speaking, fault-tolerance tries to mask faults or errors that occur on a specific layer of abstraction (regarding their respective fault model) in order to prevent them from manifesting as failures (which would be considered a fault on the next higher layer of abstraction, cf. Section 2.2.7 or [82]). Hence, the focus is on preventing failures from manifesting. A fault-tolerant system is, therefore, fully functional in the presence of faults (i.e., as long as the number of faults stays within the limits the fault-tolerance has been designed for). A fail-operational system, on the other hand, focuses on handling faults and errors that *have* manifested into failures, often on system level. *How* such a failure is handled can vary widely and whether or not the system must remain fully functional to be considered fail-operational is not universally agreed upon.

A term that is sometimes used alongside *fail-operational* is *graceful degradation*. Graceful degradation means that a system can no longer maintain all its functions but, instead, focuses either on keeping the important ones running (e.g., safety-critical functions), or on maintaining all functions but with reduced quality (e.g., lower bandwidth, higher latency, etc.). A system *can* be considered fail-operational when it reacts to a failure by entering a state of graceful degradation in which the safety-critical functionality of the system is maintained while reducing or stopping other functionality.

The NoC presented in this thesis is designed to provide fault-tolerant and hard real-time capable means of communication to safety-critical applications in a mixed-critical system. However, the "fault-tolerant" in this context is only with respect to the safety-critical communication but *not* with respect to the NoC as a whole. As will be discussed in Section 3.2, faults *can* corrupt BE traffic, meaning the NoC itself is not fault-tolerant—meaning it would be fully functional in the presence of a fault—but fail-operational by maintaining the safety-critical communication. This means: even if a *fault* causes parts of the NoC to *fail*, the critical communication in the NoC remains fully *operational*.

3.1.4 A Word on Security

The NoC presented in this thesis is designed to support the implementation of functional safety in safety-critical applications. A related topic that can have an impact on the functional safety is the *security* of a system, the difference being that security deals with hardening a system against malicious intent. Security in NoCs is an active field

of research where, e.g., side-channel attacks or even hardware trojans are a concern [51, 145]. Although the work presented in this thesis could potentially also be deployed in a system where security is a concern—possibly with some alterations—the topic of security is explicitly *not* in the scope of this thesis. However, a short discussion regarding traffic isolation and side-channel attacks will be presented in Section 4.2.1.3.

3.2 Fail-Operational Hard Real-Time NoC Concept

This section describes the first main contribution of this thesis: the concept for a NoC providing fault-tolerant communication with hard real-time guarantees to safety-critical applications in a mixed-critical MPSoC while meeting all the requirements laid out in Section 3.1.1.

3.2.1 Basic NoC Concepts

As described in sections 2.2.2 and 2.2.3, both packet and circuit switching have different strengths and weaknesses respectively. Generally, packet switching offers great flexibility but makes it difficult to give QoS guarantees. Circuit switching, on the other hand, is well suited to provide QoS guarantees but is less flexible and often suffers from under-utilization of the available resources. Both switching types can be combined to form a hybrid NoC combining the strengths of the two individual types.

The basis of the NoC presented in this thesis is a hybrid TDM and packet switched NoC similar to the one described in Section 2.2.5, including time-slot stealing. TDM traffic is used for the critical communication in the system whereas packet switching is used for non-critical BE traffic. By using TDM for critical traffic, several of the requirements laid out in Section 3.1.1 are already met. Specifically, the support of multiple concurrent critical data streams (all with the same priority), isolation of the critical traffic stream from other traffic (critical or non-critical), and QoS guarantees for critical traffic (i.e., once a TDM channel has been set up). Using packet switching for non-critical traffic, on the other hand, offers great flexibility to BE applications.

A critical part of providing QoS guarantees in a TDM NoC is the way a communication channel is set up. As discussed in Section 2.2.4, there are essentially two different approaches for the slot table configuration necessary for setting up TDM channels—distributed configuration and centralized configuration—and it was concluded that a centralized configuration—although limited in scalability—is the approach better suited to give QoS guarantees. Therefore, a centralized configuration is the approach taken in the scope of this thesis. However, depending on the implementation, centralized configuration alone does not guarantee hard real-time capabilities. The configuration policy and the way in which the central entity is connected to the individual slot tables are just as important.

Typically, the slot table configuration in TDM NoCs is either static or dynamic, meaning TDM channels are either completely fixed and cannot be adjusted or they are completely dynamic and are temporarily created on demand and torn down right after. The former lacks flexibility while the latter is unsuited for safety-critical environments

since guarantees can be given once a TDM channel is set up but no guarantees whatsoever can be given regarding *how long* it will take to set up a channel or *if* a channel can be set up at all. As an example, if packet switched traffic is used to connect a central NoC manager to the slot tables, the BE traffic would interfere with the configuration packets. But even if a dedicated and high priority traffic class would be used for configuration packets or, better yet, direct wiring or an overlay network would be used for slot table configuration, different requests and channel configurations would interfere with each other, thereby making QoS guarantees very difficult to implement.

In order to both ensure QoS and keeping some flexibility, the hybrid NoC presented in this thesis uses a *semi-static* configuration of the TDM channels. This means the TDM channels are *defined* at compile time of the software and *configured* at system start time. After the initial configuration, the channels are only reconfigured to react to faults, as will be explained in the following Section 3.2.4. This means, the TDM channels are long-lived—as opposed to set up on demand and torn down after—but the system still has the flexibility to alter the channels when needed, e.g., when the system is updated.

With this configuration policy, the question of *how* the central NoC manager is connected to the slot tables is less significant since reconfigurations should only happen very rarely at runtime. Even using packet switching without a dedicated traffic class for configuration packets would be feasible when—at system start time—the TDM channel configuration is done first before any other BE traffic is allowed. However, the approach to implement fault-tolerance for the critical communication—described in the following Section 3.2.2—has an influence on which ways of connecting the NoC manager to the slot tables are suitable, which is why this aspect will be discussed at a later point in Section 3.2.3.

3.2.2 Protection Switching in NoC

The approach taken in this thesis to implement fault-tolerance for safety-critical communication is by adapting protection switching to NoCs. With protection switching, two (or more) disjoint paths are provided for each communication channel. In case a fault occurs on one of the paths it is possible to immediately switch to the backup path. As described in Section 2.4, the concept of protection switching was originally designed for wide area networks. Therefore, some adjustments are necessary when applying protection switching to the hybrid NoC described in the previous Section 3.2.1.

In large scale networks, a disjoint path typically means a redundant physical wire or optical fiber. Similarly, in a NoC with a 2D mesh topology—which is inherently redundant—two paths are disjoint as long as they do not share any links or routing nodes. This poses a problem since in conventional NoCs at least the sender's and receiver's local router and link will be shared, even if messages are sent via otherwise disjoint paths.

In order to provide two fully disjoint paths between the NIs of any two tiles it is necessary to add a second link to connect each NI to its respective local router. Additionally, the input and output ports of the routers must be sufficiently independent from each other to ensure that no single fault can affect more than one input or output port of a router. This is done by designing the router in a modular fashion with fully independent

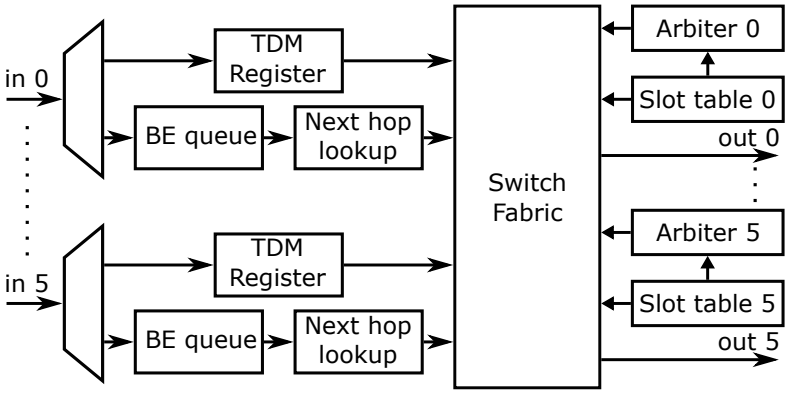


Figure 3.1: Overview of the hybrid router design supporting protection switching

input and output ports respectively. Each output port has its own slot table and arbiter for BE traffic. Furthermore, the switch fabric is designed in a way that a single fault can only ever affect the connection between a single input-output pair. As a result, this means that two TDM channels are disjoint even if they traverse the same router as long as they use different input and output ports respectively. A structural overview over a hybrid router supporting protection switching is given in Figure 3.1.

With the additional local link, the adaption of 1+1 and 1:1 protection to the hybrid NoC is, in principle, fairly straightforward: in both cases two disjoint paths are reserved using TDM, with 1+1 protection using both paths and 1:1 protection only using one path at a time. In both cases, paths can fully or partially be shared with paths from other TDM channels since different slots are reserved in the slot tables. All TDM paths are configured—meaning the corresponding slots are reserved in the slot tables—at system start time, including the secondary paths for 1:1 protection. Since time-slot stealing is allowed, the unused paths can be utilized by packet switched BE traffic, but not by other TDM channels.

The difference between 1:1 and 1:n protection is a bit more subtle than just sharing a common backup wire. With TDM, two paths overlap when they have an overlap in at least one slot in the table of at least one router along their path. This means that the overlapping slots cannot be configured at system start time since it is not yet known which TDM channel might need these slots for its secondary path. A consequence is that switching to the secondary path will take longer than with 1:1 protection since at least parts of the path must first be configured at runtime.

Figure 3.2 shows how the three protection switching versions work in a TDM NoC. The figure shows two TDM channels with two disjoint paths each. In all three cases, two paths of the channels share the link between the local routers of tile C and B. However, for 1+1 and 1:1 protection these paths don't overlap and can be used simultaneously whereas for 1:n protection the two secondary paths overlap in slot 1 of the eastern outgoing port of the local router of tile C, meaning that only one of the TDM channels can use this slot for its secondary path.

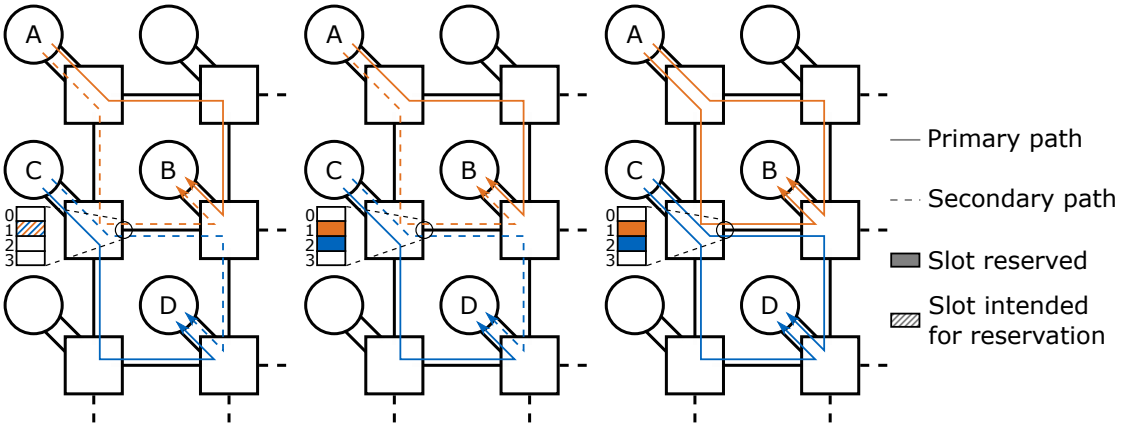


Figure 3.2: 1:n (left), 1:1 (middle), and 1+1 protection (right) in a TDM NoC

It should be noted that all three examples in Figure 3.2 can, in the worst case, only tolerate one fault between the two channels if the fault occurs on the link between the local routers of tile C and B. This is because the fault *hits* both channels at the same time. However, the difference between the switching versions becomes clear when considering a fault on the primary path of one of the channels (or on the path that does not share a link with a path of the other channel in case of 1+1 protection). In this case, only the channel that is *hit* by the fault loses its fault-tolerance—but continues to be operational—when using either 1+1 or 1:1 protection, whereas the other channel keeps its fault-tolerance. With 1:n protection, on the other hand, both TDM channels lose their fault-tolerance.

All three protection switching versions have different properties and consequential requirements regarding their implementation in NoC. One of the main contributions of this thesis is to evaluate which of the versions is best suited to implement fault-tolerance for safety-critical traffic in MPSoCs.

3.2.3 Overlay Network

As discussed in Section 2.4, for two of the three protection switching versions—1:n and 1:1 protection—the switching is done on the sender's side which requires a feedback channel from the point where a fault is detected (typically the receiver) back to the sender (in this case the sending NI). Furthermore, and as discussed in Section 3.2.1, a central NoC manager is used to configure the TDM channels. This NoC manager must be connected to the slot tables in the system (and must, in case of 1:n protection, also be notified about faults in order to configure the secondary paths). Both can be achieved via an overlay network.

The other two potential options mentioned in Section 3.2.1 are to use either direct wiring or packet switching for this feedback channel and the connection between the NoC manager and the slot tables. Using packet switching is problematic since it undermines the isolation of critical and BE traffic. But even if these flits are given higher priority,

using the hybrid NoC itself for the feedback channel means the message sent back could be corrupted by the very fault it tries to report. Direct wiring, on the other hand, does not scale well as discussed in Section 2.2.4. Furthermore, if direct wiring is used to connect the NoC manager to the slot tables and NIs, the feedback channel to the sending NI also has to be via the NoC manager, thereby making it a single point of failure. This, of course, is anyhow the case—specifically for 1:n protection—and will, therefore, be further discussed in Section 3.2.4.

In contrast to 1:n and 1:1 protection, 1+1 protection does not necessarily require a feedback channel. Yet still, the NoC manager must be able to configure the slot tables of the system. Since it is part of this thesis to determine which protection switching version is best suited to be used in NoCs—and for the reasons mentioned above—an overlay network will be considered for the feedback channel required by both 1:n and 1:1 protection, and for the configuration of the slot tables.

To achieve fault-tolerance of the TDM communication in the hybrid NoC this overlay network must itself be fault tolerant. This seems to just shift the original problem of implementing fault-tolerance to a different level. However, this overlay network has much lower requirements in terms of throughput and concurrency since it only has to forward very few messages. Therefore, a high degree of redundancy—even TMR—comes at a much lower cost. For the time being—and for the simulation based proof of concept in Section 3.4—it will be assumed that a suitable overlay network exists and can be used.

3.2.4 Maintaining Fault-Tolerance

Fault-tolerance with protection switching can tolerate at least one fault occurring on a critical channel. Even multiple faults can be tolerated as long as only one of the two paths of a channel is affected. The critical communication is maintained and critical applications remain fully operational. However, after switching the communication to the secondary path the TDM channel is no longer fault tolerant.

Operating the system in a non fault-tolerant state might be acceptable for a limited period of time. For instance, an autonomous driving system might bring the car to a safe halt in a location where it is convenient or the system could take other actions such as reducing speed, informing the passengers, and handing over the control of the car to a passenger. To continue operation long term, however, it is necessary to bring the communication back to a fault-tolerant state.

Bringing an operational but no longer fault-tolerant TDM channel back to a fault-tolerant state can be done by reconfiguring individual TDM paths at runtime. For this purpose, a set of additional backup paths could already be pre-computed at compile time. Alternatively, the central NoC manager could try to find an alternative path at runtime considering the current state of the NoC. Setting up this new backup path should be done in a timely manner, but the time frame can be a bit more lenient since normal operation of the system is ensured in the meantime. If no new backup path can be found—or there are other problems in the system, e.g., a fault in the NoC manager—the system could take further actions such as the ones described above. Figure 3.3 shows an overview over the described concept.

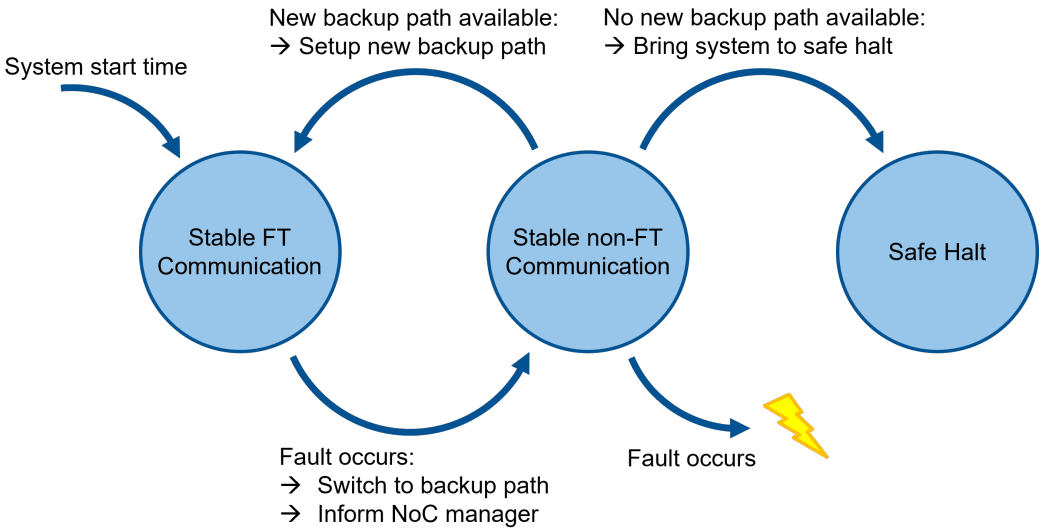


Figure 3.3: Concept for maintaining fault-tolerance

The central NoC manager plays a crucial role in the overall operation of the NoC and could, therefore, be seen as a single point of failure. However, it is only directly involved in switching from the primary to the secondary path when 1:n protection is used since it must configure at least parts of the secondary path. In this case, measures to harden the NoC manager against faults are imperative. On the other hand, for both 1:1 and 1+1 protection the NoC manager is not directly involved in switching paths. Therefore, fault-tolerance of the NoC manager is not vital, only that a fault can be detected and reacted to accordingly. All paths are set up at system start time before the normal operation of the system starts. If a fault is detected in the NoC manager at this time—e.g., by means of a BIST—the system will not start its normal operation. If a fault is detected at a later point—e.g., when a TDM channel is no longer fault-tolerant—the system could react the same way it would if no new backup path could be found. The fact that the NoC manager is vital if 1:n protection is used is a first argument against using that version of protection switching in NoCs.

3.2.5 Path Synchronization

An important aspect of the fault-tolerance concept presented in this thesis is that it is completely transparent to the safety-critical applications using it. This means that, with or without a fault, all data sent must arrive at its destination within a given time frame, in correct order, and with neither data lost nor delivered twice.

When using 1:1 or 1:n protection the receiving NI will only ever receive data on one path at a time. However, the data stream can be cut off by an occurring fault at an arbitrary time, even mid-message. In order to prevent data loss, a cut-off message must be sent again on the secondary path. This means, all messages must be stored in the sending NI until it can be assumed that they have been fully received without a fault.

Ideally, only the faulty flits would be sent over the secondary path and the receiving NI could immediately add them to the input buffer. However, determining which flits exactly must be re-sent is non-trivial since it depends on multiple factors such as the path length, the slots that are reserved, and the current cycle. Therefore, it is more feasible to re-sent an entire message. The receiving NI, on the other hand, must be able to distinguish new flits from flits that have already been received before without a fault.

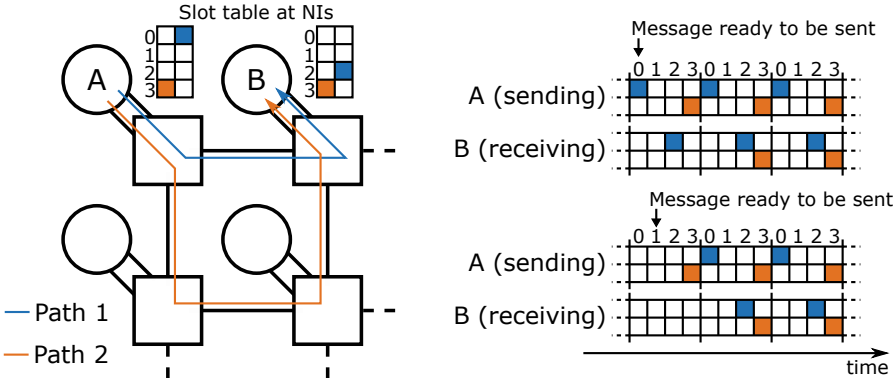
To solve this issue some kind of message synchronization between the two paths is necessary. This can be done by periodically inserting checkpoints into the data stream. Checkpoints are special flits—similar to header flits in a packet switched NoC—that divide the data stream into data *units*. The receiving NI uses the checkpoints to distinguish new data units from duplicates, which are discarded. The sending NI, on the other hand, keeps the sent data units in its internal buffer until it can be assumed that a data unit has been received fault free. This is the case if both the worst-case latency for sending a data unit and the latency of the feedback channel have passed since the data unit has been sent (cf. Section 3.3). Different ways of implementing these checkpoints are discussed in Section 4.2.1.1.

With 1+1 protection the sending NI does not have to store the messages after sending. However, the receiving NI must decide which incoming flits it forwards to the input buffer and which ones it discards[3]. In general, corresponding flits do not arrive in the same clock cycle on the two incoming links. With a slot table size of S and s reserved slots for a TDM channel there can be a gap of up to $S-s$ cycles between sending two corresponding flits over the two links. If the two paths are not equidistant the gap between the two flits can even be larger than S at the receiving NI. An example of this is shown in Figure 3.4a. Here, if a flit is sent out from node A on path 1 in slot 0 it will arrive at node B two cycles later in slot 2. The same flit is sent out on path 2 in slot 3 but due to the longer path it will arrive four cycles later in slot 3 of the next TDM period. At this time, a second flit can already have been sent and received via path 1. In this case, the gap between receiving the first flit on path 1 and path 2 is five cycles, which is more than the size of the slot table.

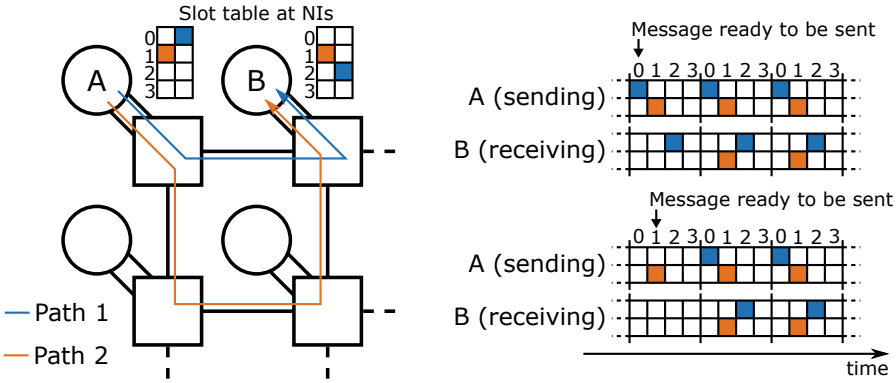
It cannot generally be assumed that a flit will always arrive first on one link since this depends on the time slot in which a message is enqueued (and ready to be sent out) at the sending NI. In the example shown in Figure 3.4a this would, in fact, be the case as flits will always be received on path 1 first. Depending on when a message is enqueued path 1 can be up to two flits ahead of path 2. However, with a small change of the reserved slots this is no longer the case. If slot 1 is reserved for path 2, instead of slot 3, flits can first be received on either path, depending on when a message is enqueued. This is shown in Figure 3.4b. In this case, if a flit is ready to be sent out in slot 0 it will first be received on path 1. If it is ready to be sent out in slot 1 it will first be received on path 2.

In case of 1+1 protection it would be possible to simply count the number of flits that have been received on both paths in order to forward the correct flits to the input buffer and discard duplicates. However, this approach breaks down when reconfigurations are

[3]Given that both data streams are fault free.



(a) Path 1 always ahead at receiving NI



(b) Either path can be ahead/behind at receiving NI

Figure 3.4: Change of flit distance at receiving NI for 1+1 protection due to different path length, slot reservation, and time of enqueuing a message

done at runtime. In this case, no flits would be received on the faulty path for the duration of the reconfiguration, meaning the receiving NI has no way of determining which flits correspond to one another once both paths are active again. This issue can be solved much the same way as the path synchronization for 1:1 and 1:n protection: by dividing the data stream into data units and adding checkpoints to the data stream. When a data unit has been received without a fault it can be forwarded to the input buffer, faulty or duplicate data units are discarded.

Adding checkpoints to the data stream naturally increases the number of flits sent and, thereby, the bandwidth requirement of TDM channels. This must be considered when reserving the time slots for each channel. The distance between two checkpoints is a trade-off between overhead of flits sent and buffer space in the NIs. With a distance of 1—meaning there is only one data flit between two checkpoints—the flit overhead is 100% and the required channel bandwidth must be twice as high as without checkpoints. The larger the distance between two checkpoints the lower the required bandwidth overhead but the more flits must be stored in the receiving NI before they can be forwarded (or

discarded). This trade-off will be further discussed in Section 4.2.1.2 where a hardware architecture is presented.

3.3 Worst-Case Timing Analysis

In this section, a formal analysis of the worst-case latencies of TDM channels in general and the three protection switching versions in particular is presented.

There are five parameters that affect the worst-case latency of a message sent over a TDM channel: the latency per hop, the number of hops N, the size of the slot table S, the number of slots s assigned to the channel, and the number of flits f of the message.

The latency per hop is typically 1 cycle in a TDM NoC and will, therefore, not be considered any further[4]. The number of hops will typically be relatively low with the proposed concept since the critical applications are mapped at compile time which allows to optimize for short paths. But even if the applications would be spread out and a system with a 16x16 NoC mesh were to be considered—which is well beyond any system currently deployed in automotive and avionics—the largest hop-count between any two tiles would be 31 and, thereby, cause a latency of just 31 cycles.

A major system design parameter is the slot table size S which is defined at system design time and typically cannot be changed afterwards[5]. S limits the number of TDM channels that can share a single link and, at the same time, defines the minimal bandwidth BW_{min} that can be assigned to a channel by determining the quantization of the available bandwidth BW: $BW_{min} = BW/S$. The number of slots s assigned to a TDM channel defines the maximum bandwidth BW_{max} the channel can use ($BW_{max} = s \cdot BW_{min}$) but also has an influence on the worst-case latency of both a single flit as well as a message composed of multiple flits.

Once a TDM flit is injected into the NoC its latency is exactly $N+1$ cycles (one cycle per hop plus one cycle until it reaches the receiving NI). Additionally, a flit might have to wait for up to $S - s$ cycles at the sending NI before it can be injected into the NoC. This means, the worst-case latency C_1 in cycles for a single flit is:

$$C_1 = (S - s) + (N + 1), \forall s, S, N \in \mathbb{N}^+, s \leq S \tag{3.1}$$

For better readability, the domain and constraints of the parameters are not listed for each of the following formulas but are instead listed in Table 3.1 along with all parameters used throughout this section.

When sending a message with multiple flits the latency heavily depends on the slot table size S, the number of reserved slots s, and the number of flits f. On the other hand, the effect that the number of hops N has on the total worst-case latency decreases with an increasing number of flits f. The reason is that only s flits can be sent out every S cycles. As stated in Equation 3.1 the first flit must, in the worst case, wait for $S - s$

[4] If the latency per hop is more than 1 it simply becomes a multiplier for the number of hops N.

[5] A system with configurable slot table size is proposed in [49]. However, the upper bound must still be defined at system design time and adjusting the slot table size at runtime causes all slot tables to be wiped, thereby making the approach unsuited for safety-critical applications.

cycles. Afterwards, flits can be sent out in the following $s-1$ cycles before the next flit must wait another $S-s$ cycles.

In the simplest case with just one slot assigned to the TDM channel ($s=1$) the flits following the first flit cause an additional delay of $S \cdot (f-1)$ cycles, causing a total worst-case latency C_2 of:

$$C_2 = (S-1) + (N+1) + (S \cdot (f-1)) = S \cdot f + N \tag{3.2}$$

It is immediately obvious that with growing S and f the hop-count N has only little effect on the worst-case latency. Equation 3.2 can be further generalized to also include cases where more than one slot is reserved for a channel. After the first flit, s flits can be sent every S cycles for as long as at least s flits remain, afterwards, the remaining flit(s) can be sent out in the following cycle(s). Equation 3.3 determines the worst-case latency for TDM channels in general (i.e., without protection switching and with no fault occurring). Even with multiple slots assigned to a TDM channel, the hop-count only has a minor influence on the overall worst-case latency when multiple flits are sent, especially with a large slot table.

$$C_{TDM} = (S-s) + (N+1) + \left(S \cdot \left\lfloor \frac{f-1}{s} \right\rfloor + (f-1) \bmod s \right) \tag{3.3}$$

When protection switching is used, the worst-case latency of the different protection switching versions must consider an occurring fault on the primary path (or one of the paths in case of 1+1 protection). Since checkpoints are added to the data stream the parameter f must be redefined to include these checkpoint flits. With a checkpoint distance of d—meaning one checkpoint is added for every d data flits—the number of checkpoint flits c that are added is:

$$c = \left\lceil \frac{m}{d} \right\rceil \tag{3.4}$$

with m being the number of data flits in the message sent. The total number of flits sent when using protection switching is therefore:

$$f = m + c \tag{3.5}$$

For 1+1 protection, the worst-case latency is equal to C_{TDM}—but with the definition of Equation 3.5 for f—if and only if the two paths are equidistant. Otherwise, Equation 3.3 must be applied to both paths individually to determine the worst-case latencies C_{pathA} and C_{pathB} of the two paths. The overall worst-case latency is then:

$$C_{1+1} = max\{C_{pathA}, C_{pathB}\} \tag{3.6}$$

The worst-case latency is lowest for $d=m$ since this results in only one checkpoint flit being added to the data stream. However—depending on the checkpoint implementation—this leads to large receiving buffers in the NIs, as will be discussed in Section 4.2.1.2.

For both 1:1 and 1:n protection an additional parameter must be considered: the latency F caused by the feedback channel via the overlay network mentioned in Section 3.2.3. Furthermore, the additional latency P to configure the secondary channel in case of 1:n protection must be considered as well. For simplification the following considerations assume equidistant paths.

The worst-case scenario is that the last flit of a data unit gets corrupted by a fault in which case the entire unit must be discarded and re-transmitted. This means that, contrary to 1+1 protection, the worst-case latency for 1:1 and 1:n protection is highest for $d = m$. For this case, the worst-case latency is given with Equation 3.7 (with $P = 0$ for 1:1 protection). Even with instantaneous feedback channel ($F = 0$) and path configuration ($P = 0$) the worst-case latency is almost twice as high as C_{TDM}:

$$C_3 = (2C_{TDM} - 1) + F + P \tag{3.7}$$

The worst-case latency of 1:1 and 1:n protection can be reduced by choosing a lower checkpoint distance in order to reduce the amount of data that must be re-transmitted. The baseline worst-case latency for both 1:1 and 1:n protection is the fault free transmission C_{TDM} with the definition of f given in equation 3.5. The worst-case latency C_{DU} of a single data *unit* can be determined similar to C_{TDM} by replacing f with $d+1$ in Equation 3.3:

$$C_{DU} = (S - s) + (N + 1) + \left(S \cdot \left\lfloor \frac{d}{s} \right\rfloor + d \bmod s \right) \tag{3.8}$$

Both protection switching versions require a re-transmission of all flits since the last *safe* checkpoint over the secondary path. A checkpoint can be considered safe if at least $C_{DU} + F$ cycles have passed since its injection without a fault being reported via the overlay network. Accordingly, the worst-case latency C_T for triggering a switch in case a data unit is affected by a fault is:

$$C_T = C_{DU} + F - 1 \tag{3.9}$$

With this, the worst-case latency for 1:1 protection is:

$$C_{1:1} = C_{TDM} + C_{DU} + F - 1 = C_{TDM} + C_T \tag{3.10}$$

And the worst-case latency for 1:n protection is:

$$C_{1:n} = C_{1:1} + P \tag{3.11}$$

In case the two paths are not equidistant, C_{TDM} and C_{DU} must be determined for the shorter path—assuming the shorter path is used as primary path—and the difference in the number of hops of the two paths must be added.

Symbol	Meaning	Domain / Constraint
C_x	worst-case latency in clock cycles	$C_x \in \mathbb{N}^+$
S	slot table size	$S \in \mathbb{N}^+$
s	number of reserved slots	$s \in \mathbb{N}^+, s \leq S$
N	distance in number of hops	$N \in \mathbb{N}^+$
f	number of flits sent	$f \in \mathbb{N}^+$
c	number of checkpoint flits	$c \in \mathbb{N}^+$
m	number of data flits in a message	$m \in \mathbb{N}^+$
d	number of data flits per checkpoint flit	$d \in \mathbb{N}^+$
F	latency for feedback path in clock cycles	$F \in \mathbb{N}^+$
P	latency for path config. in clock cycles	$P \in \mathbb{N}_0$

Table 3.1: Parameter list for formal analysis

3.3.1 Comparison of Switching Versions

Equations 3.6, 3.10, and 3.11 show that—out of the three protection switching versions—1+1 protection has the lowest and 1:n protection the highest worst-case latency. The worst-case latency of 1+1 protection is virtually equal to the worst-case latency of a TDM channel without protection switching. The worst-case latencies of 1:1 and 1:n protection can be more than twice as high, depending on the checkpoint distance d, and depending on the additional latencies from F and P.

Both parameters F and P are implementation dependent and can only be approximated here. However, it can be assumed that both parameters will—in the worst case—grow linearly with the number of hops, thereby making short paths more desirable. For both 1:1 and 1:n protection the worst-case latency can be reduced by inserting more checkpoints into the data stream (meaning, lowering d) whereas for 1+1 protection it is advantageous to insert fewer checkpoints. When designing a system using protection switching, the tolerable worst-case latency of the applications must be considered when deciding which protection switching version to use, how to dimension S, s, d, and N, and how to implement the overlay network which determines F and P.

3.4 Proof of Concept

The previous sections of this chapter described the concept for a fault-tolerant hard real-time NoC using protection switching and provided a formal analysis of the worst-case latencies for the different protection switching versions. Based on these sections, this section describes the experimental setup and the simulations that have been done in order to allow a first evaluation of the proposed concept.

Both the fault-tolerance and the hard real-time capability of the safety-critical traffic are given by design—as described in the previous sections 3.2.2 and 3.3—and would, for itself, not need to be tested in a simulation. Instead, these first simulations are intended to evaluate the impact that the TDM traffic has on the packet switched BE traffic as well as the overall network utilization. Furthermore, the simulations provide a basis for

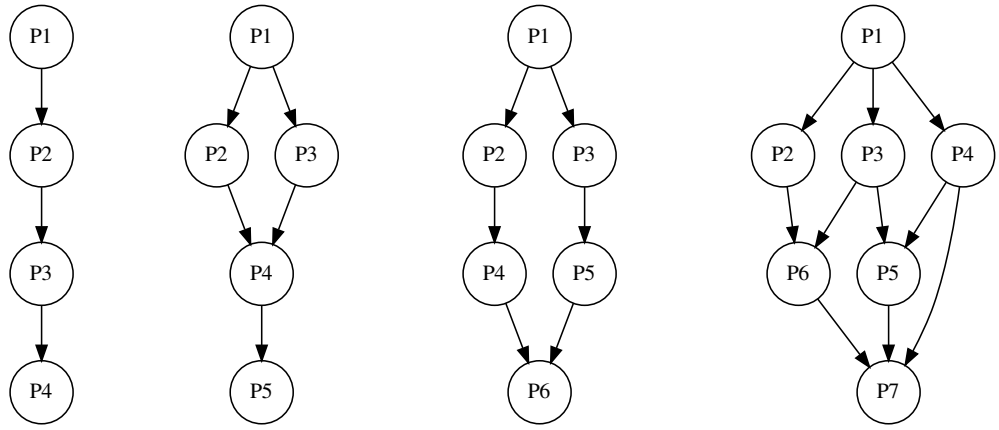


Figure 3.5: Task Graphs used to represent critical applications

a first comparison of the different switching versions. A more in-depth hardware-based evaluation is later presented in Chapter 5.

3.4.1 Experimental Setup

In order to accurately evaluate the impact the TDM traffic has on the BE traffic cycle-accurate simulations were made. The simulations were done with an 8x8 NoC mesh of hybrid routers implemented in SystemVerilog Hardware Description Language (HDL). The architecture of the packet switched part of the router is kept very simple: static XY routing is used and the router does not provide virtual channels. The slot table size for TDM is set to 8. The simulation of the NoC mesh is done with Synopsys VCS. The remaining parts of the system (i.e., the NIs, the overlay network, the central NoC manager, the processing tiles, and the processes running on them) are abstracted in a testbench created with cocotb [146] and Python.

4 randomly chosen task graphs—shown in Figure 3.5—were used to represent a set of critical applications for the simulations. All graphs have one source and one sink vertex, are directed, acyclic, and have vertices with a degree of up to 3. Two different mapping scenarios—shown in figures 3.6 and 3.7 respectively—were simulated, each with different TDM injection rates and different protection switching schemes.

The two mapping scenarios differ greatly in number of critical tiles and distribution of the critical traffic. In the first scenario, 11 task graphs—taken from the set of randomly chosen task graphs—are spread across the entire 8x8 system in a way that almost all nodes are part of a critical task graph and, therefore, produce and/or consume critical TDM traffic. An average flit generation rate of both 5% and 10% per node was simulated for the TDM traffic, meaning that, on average, 3.2 respectively 6.4 flits are generated in each clock cycle in all nodes combined. However, since not all nodes generate TDM traffic, the generation rate of a single node can be significantly higher. Furthermore, these *generation* rates describe the net traffic, meaning that the *injection* rates are effectively

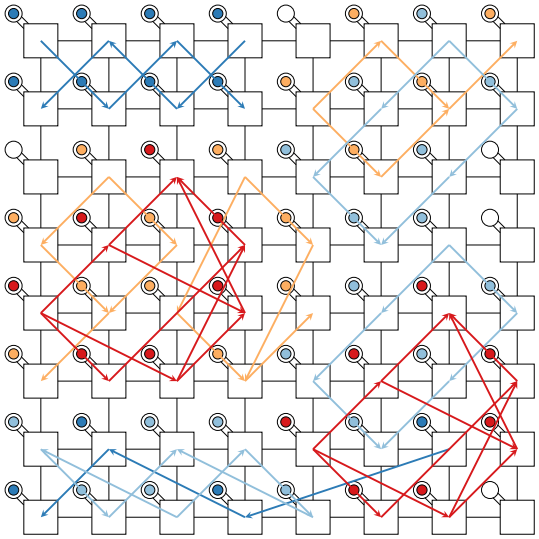


Figure 3.6: Scenario 1 - 11 task graphs

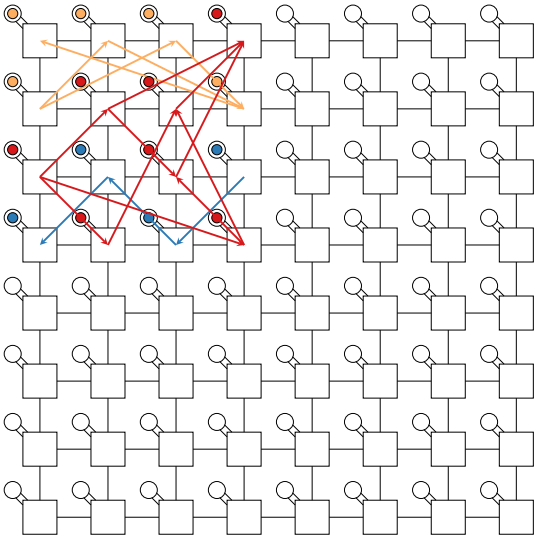


Figure 3.7: Scenario 2 - 3 task graphs

twice as high for 1+1 protection. Figure 3.6 shows the logical connections representing the channels that are configured for the critical communication (each connection has a disjoint primary and a secondary path, not shown in the figure). In the first scenario, all tiles generate and consume uniform random BE traffic to measure the effect the TDM traffic has on the packet switched traffic. It is important to note that this is purely for testing and would typically not be done in a deployed system since it violates isolation of critical and non-critical applications in the processing tiles.

In the second scenario, 3 task graphs are mapped to just 16 nodes, clustered in a 4x4 corner of the system, as shown in Figure 3.7. The critical tiles do not generate or consume BE traffic. Instead, uniform random BE traffic is sent and received by the remaining 48 tiles, meaning the critical and BE applications are isolated from one another. In this scenario, due to the dense mapping and lower amount of critical tiles that generate traffic, only an average per-node TDM flit generation rate of 5% could be achieved—meaning a total average of 3.2 flits per cycle. However, this means the 16 participating critical nodes have an average per-node injection rate of 20%—40% for 1+1 protection—since only a quarter of the NoC is used. The critical traffic is only located in the 4x4 corner of the system. The BE traffic, on the other hand, still traverses the *critical region* of the system—due to the XY routing—which means it is still affected by the critical traffic.

These two very different scenarios were chosen to test protection switching in NoC under two completely different conditions (high density of critical nodes on one hand and high amount of overall traffic generated and injected into the NoC on the other hand) and to get a first impression of the effects on the BE traffic. The results are a first point of reference for comparison with a more in-depth evaluation being presented in Chapter 5.

In the simulations, the BE packets have an average size of 30 flits and each router has an input buffer that can hold 16 flits. Per-tile BE flit generation rates of 10% to 30% are simulated in 2.5% steps to measure at which point the network saturates. When a BE packet is generated in the testbench, a timestamp is added to the packet which, upon arrival of the packet at its destination, is used to determine the latency of that packet. The packet is then enqueued at the sending NI which will eventually inject it into the NoC mesh. In the simulation, these outgoing queues have a virtually infinite length (limited only be the memory of the machine running the simulation), meaning that the queue can grow indefinitely if the packet generation rate exceeds the maintainable injection rate. As a consequence, this also means that the packet latency grows indefinitely if the packet generation rate exceeds the injection rate. This is, of course, impossible in a real system. However, using infinite queues is useful to quantify the results of the simulations by using both the average packet latency and the average number of enqueued packets per node as an indicator to determine *if* the NoC is saturated and rank different simulation results by *how severe* the saturation is.

Each simulation runs for 100 000 cycles after a warm-up period of 10 000 cycles, after which the link load has stabilized. Typically, longer simulations would be preferred to increase confidence in the obtained results. However, the cycle accurate simulation of an entire 8x8 NoC is a compute-intensive task which can take several hours for each run—even for just 110 000 cycles—which is why the runtime was limited for this first proof of concept.

3.4.2 Simulation Results

Three major metrics were monitored during the simulation runs and were used to evaluate the results: the utilization of each individual link for both TDM and packet switched traffic, the number of BE packets in the outgoing queues of each tile, and the latency from generation of each BE packet until its delivery. For both scenarios, simulations in which only BE traffic was generated and injected into the NoC were run as a baseline reference.

Scenario 1

Figure 3.8 shows the average BE packet latencies for scenario 1. Several observations can be made here. First, it can be seen that all curves progress the way one would expect in a NoC with slowly increasing latencies for lower generation rates and then a sudden steep increase once the network saturates. For the reference simulations, saturation is reached when going from a 22.5% to a 25% flit generation rate. The simulations with TDM traffic and protection switching reach saturation at lower BE flit generation rates which is of little surprise since the overall amount of traffic in the NoC is larger.

The second observation that can be made is that the BE packet latencies are higher the more TDM traffic is present, which could also be expected. The simulations of 1:n and 1:1 protection with a 5% TDM flit generation rate are almost identical and always have slightly higher BE packet latencies than the reference simulations. Next,

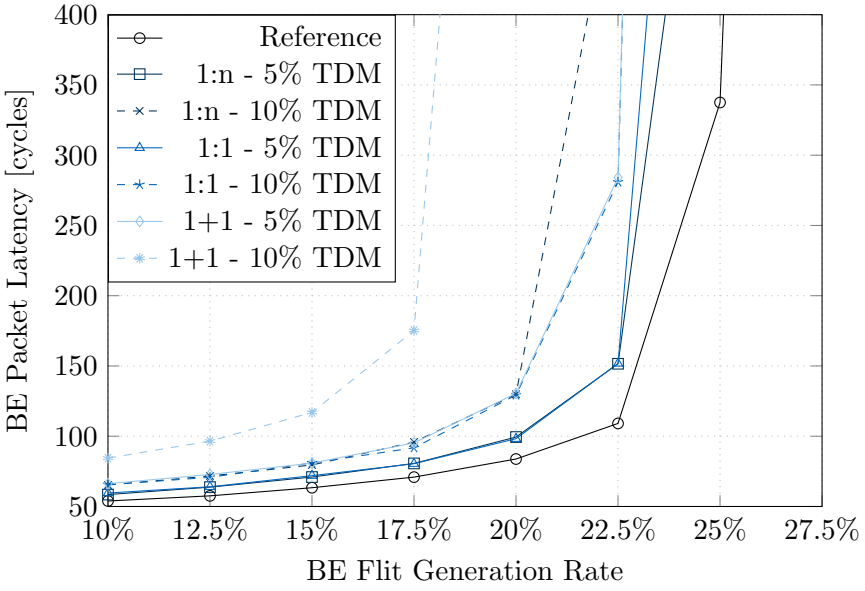
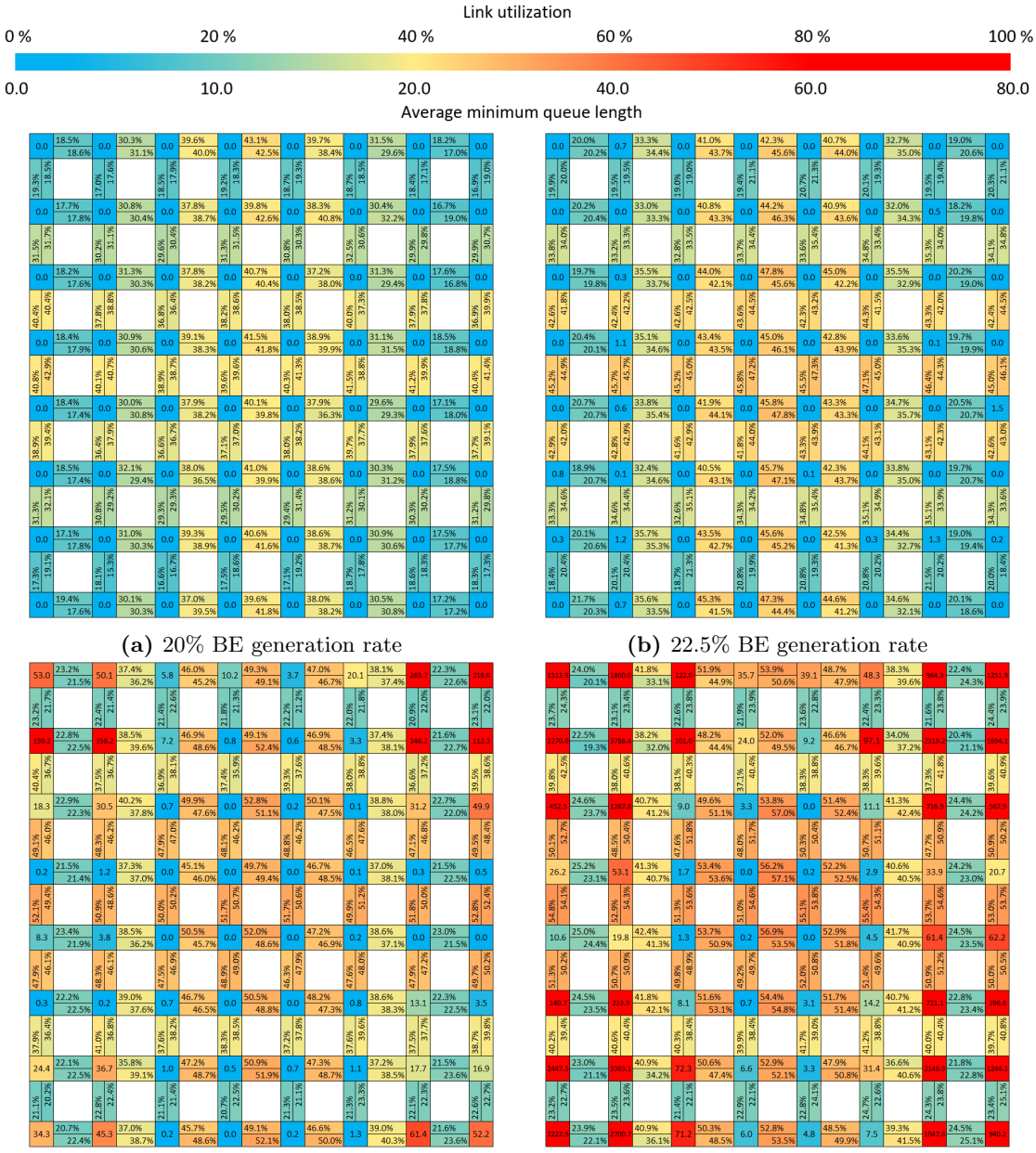


Figure 3.8: Average BE packet latency - Scenario 1

the simulations of 1:n and 1:1 protection with a 10% as well as 1+1 protection with a 5% TDM flit generation rate are almost identical until saturation is reached. They all show higher BE packet latencies than the simulations of 1:n and 1:1 protection with a 5% TDM flit generation rate. The reason for the similar progression of these three curves is that they all inject the same amount of TDM flits into the NoC. The difference of the curves for 1:n/1:1 protection with 5% TDM flit generation rate and the curves for 1:n/1:1 protection with 10% and 1+1 protection with with 5% TDM flit generation rate can be explained by the fact that these differences occur at higher BE flit generation rates when the network is already in saturation. Once this is the case, small differences in the traffic generation triggered by the random number generators of the testbench can lead to big differences in the observed latencies, especially with the current test setup and the limited number of simulated clock cycles. Lastly, the simulation runs of 1+1 protection with a 10% TDM flit generation rate show the highest average BE packet latencies and reach saturation at the lowest BE flit generation rates of all simulations.

Since the link utilization of the individual links as well as the number of packets in the outgoing queues of each tile were monitored during each simulation it is possible to create a heatmap for each simulation. Figure 3.9 shows the heatmaps of four reference simulation runs with different BE generation rates. The link utilization shows the average utilization of a link for the entire simulation (not including the warm-up period). It is important to note that in the heatmap a link is considered to be utilized when a flit actually traverses the link as opposed to when the link is blocked by a flit that could be sent but cannot yet be received by the downstream router due to backpressure.

To determine a meaningful value for the number of elements in the outgoing queue of a tile is not as straightforward. The number of elements in a queue will typically



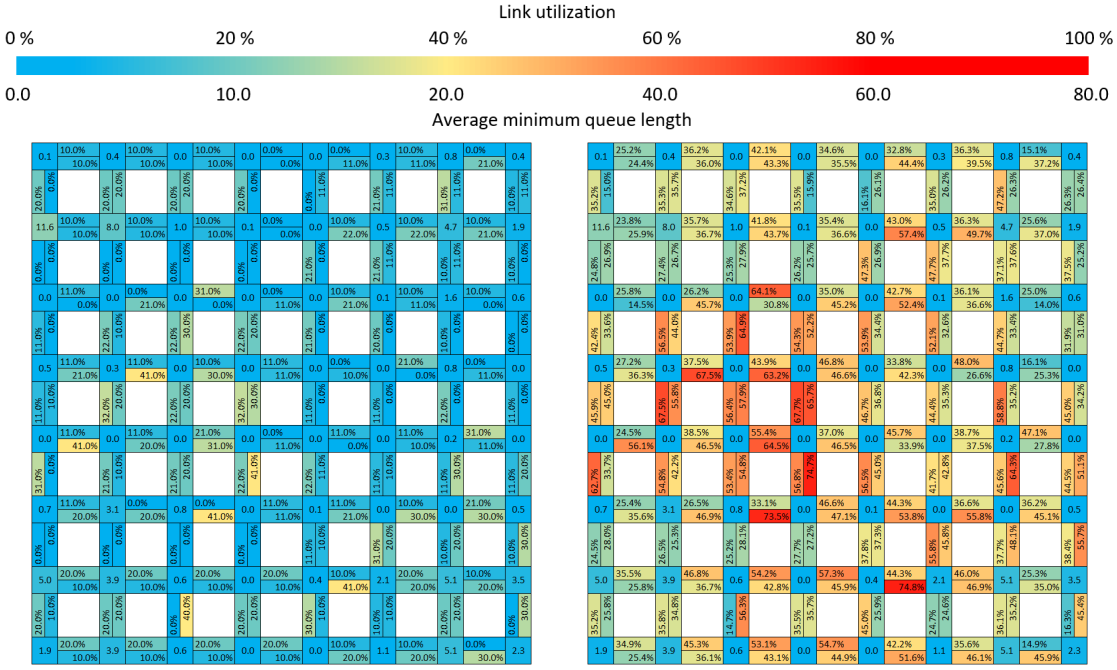
(a) 20% BE generation rate

(b) 22.5% BE generation rate

(c) 25% BE generation rate

(d) 27.5% BE generation rate

Figure 3.9: Heatmaps scenario 1 - reference simulation runs with the average utilization of each link and the average minimal queue length of each tile



(a) TDM utilization **(b)** Total utilization

Figure 3.10: Heatmaps scenario 1 - 1+1 protection, 10% TDM, 17.5% BE flit generation rate

vary greatly during a simulation run depending on the traffic pattern and, therefore, simply calculating the average across the entire run has little significance. Instead, it is important to know if, overall, the injection rate can keep up with the generation rate. Therefore, time windows were determined and the average of the minimal queue lengths in each window was calculated and added to the heatmap. If the value is zero or close to zero it can be concluded that the injection rate can, overall, keep up with the generation rate. On the other hand, if the value is significantly larger than zero—typically around 20 or more, depending on the number of tiles—it can be concluded that the injection rate is lower than the generation rate and the queue will grow in length indefinitely which indicates that the network is saturated. The higher the average minimum queue length the faster the queue grows in length. It should be noted that the coloring visualizing the average minimal queue length of the tiles in Figure 3.9 is not linear to account for the fact that even relatively low numbers already indicate saturation.

Overall, Figure 3.9 shows the expected behavior of a packet switched NoC with XY routing. The links in the middle of the network face a higher utilization due to the routing algorithm. With growing flit generation rates the tiles in the corners are the first ones that are affected by saturation, followed by the tiles at the edges of the NoC. As can also be seen, the achievable link utilization typically stays below 50%.

Figure 3.10 shows two heatmaps of a simulation run with 1+1 protection and a 10% TDM flit generation rate. The figure shows both the utilization caused by just the TDM traffic and the total utilization. In this simulation, the NoC is close to being saturated

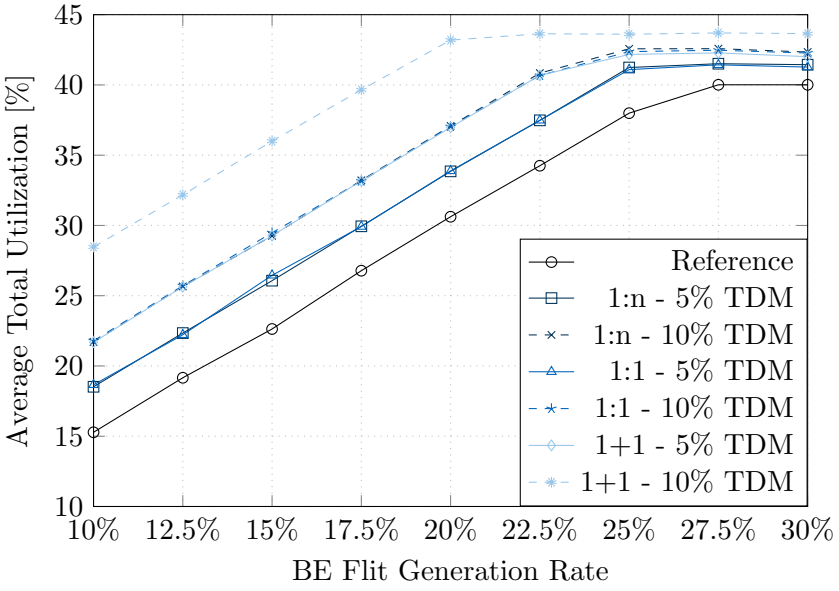


Figure 3.11: Average total network utilization - Scenario 1

which causes some of the tiles to have an average minimum queue length greater than zero. The heatmaps show that the TDM traffic causes bottlenecks for the BE traffic on some of the links which in turn cause backpressure and network saturation at higher BE flit generation rates. However, the heatmaps also show that the total utilization of individual links can go as high as 74.8% which is much higher than any sustainable link utilization in the reference simulations. Furthermore, the combination of TDM and packet switched traffic seems to allow a higher overall utilization of the NoC.

This hypothesis is supported by Figure 3.11 which shows the average total network utilization of the different simulation runs. The figure shows that the utilization of the NoC grows linearly with increasing flit generation rates until saturation is reached and the amount of injected BE flits is limited due to backpressure. As can be seen, the achievable network utilization is significantly higher when using TDM and packet switched traffic in combination.

It is important to keep in mind that Figure 3.8 only considers BE flit generation rates and, thereby, shows only half the picture. Figure 3.12 shows the total flit generation rates of the individual simulation runs on the X-axis, meaning that the curves of the simulations with protection switching are shifted to the right compared to Figure 3.8. As can be seen, the simulations with TDM traffic allow for a higher overall flit injection before the network saturates, even when using 1+1 protection. This comparison is not entirely fair since most of the TDM channels in scenario 1 have a hop count of just 2 or 3 which is significantly lower than the average hop count of the uniform random packet switched BE traffic. However, it indicates that the bandwidth *cost* that protection switching has for the BE traffic is not as high as suggested by Figure 3.8. Since

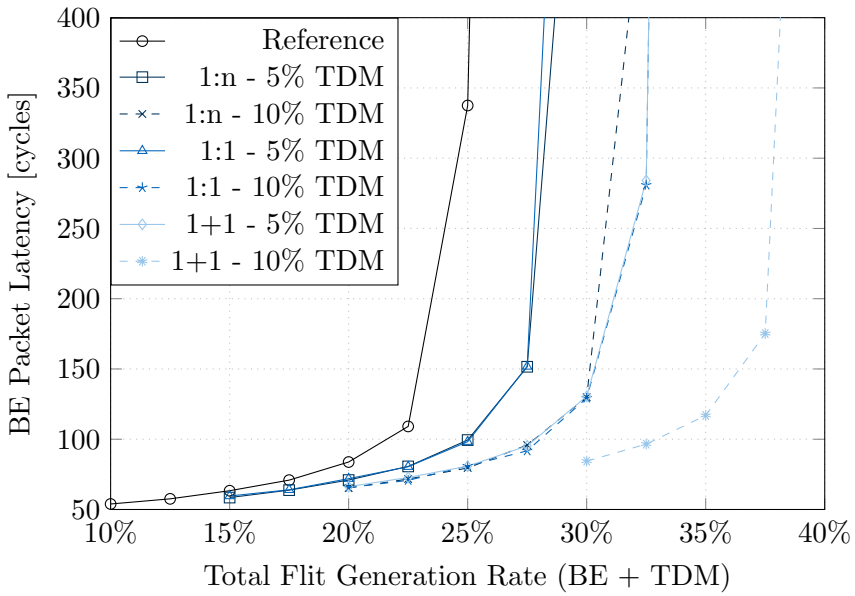


Figure 3.12: Average BE packet latency - Scenario 1, total flit generation rate

simulations of a single scenario do not allow for a conclusive answer, this will be further investigated in Chapter 5.

Scenario 2

The simulations of scenario 2 show very different results than the ones from scenario 1. In this scenario the system is divided into critical tiles generating and consuming TDM traffic and non-critical tiles generating and consuming BE traffic. Figure 3.13 shows the average BE packet latencies for the different simulations. As can be seen, the network saturates at the same BE generation rate regardless of whether or not TDM traffic is present. Furthermore, the NoC saturates at a lower BE generation rate in comparison to the simulations of scenario 1, especially when looking at the reference simulations. It is important to keep in mind, though, that the generation rate is given with regards to the total number of tiles. Since only 48 of the 64 tiles generate BE traffic this means that the BE tiles can maintain an injection rate of 20% and only cause saturation at higher generation rates. However, this is still a considerably lower rate than could be achieved with scenario 1.

The reason for this earlier saturation can be seen in Figure 3.14. The static XY routing causes an uneven utilization of the links in the NoC. For instance, packets from the south-west quarter of the system have a very high chance of having a destination in the eastern half of the system which causes a high utilization on the links from the south-west quarter to the south-east quarter. On the other hand, packets from the north-east quarter of the system have an equal chance of having their destination be in the south-west and south-east quarter, meaning that the links away from the north-east

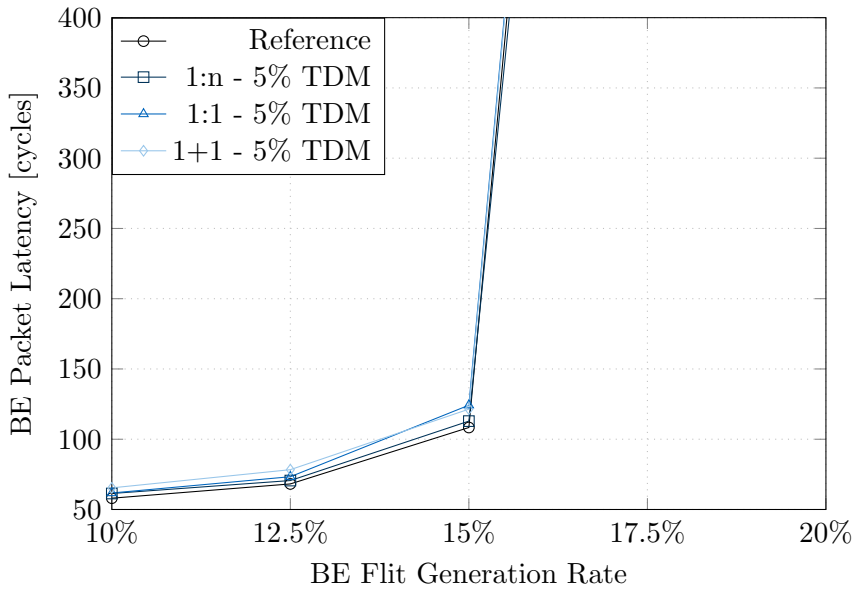


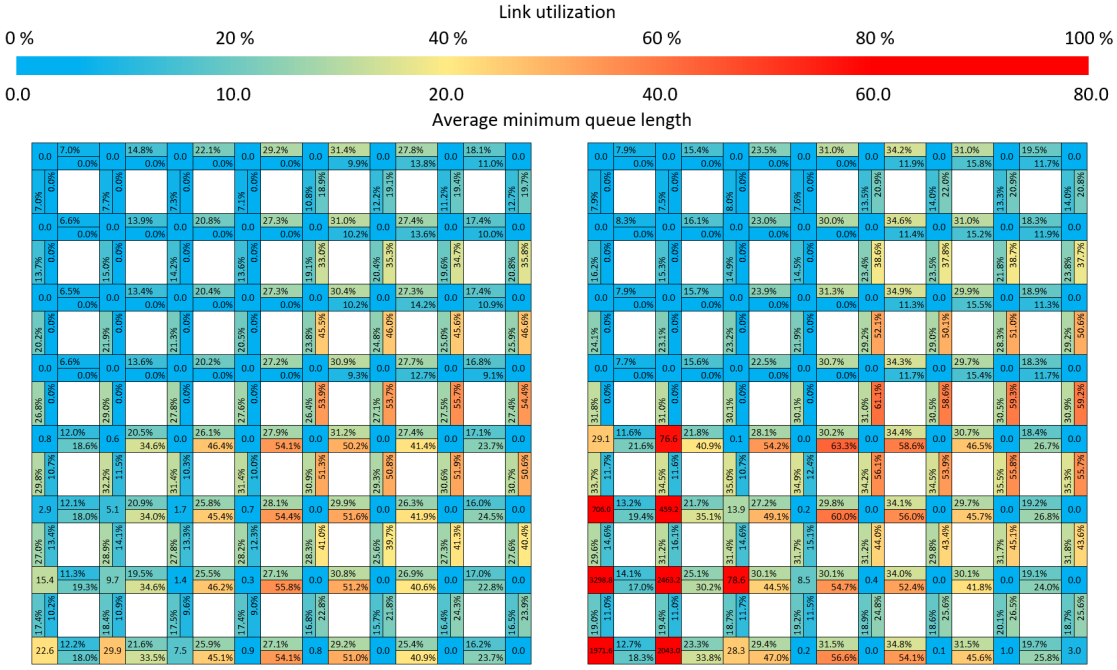
Figure 3.13: Average BE packet latency - Scenario 2

quarter are much more evenly utilized. With increasing traffic generation rates, this uneven link utilization eventually causes backpressure—especially on the links from the south-west quarter to the east and from the south-east quarter to the north—which first affects the tiles in the south-west quarter.

The fact that the TDM traffic has only little effect on the BE traffic in scenario 2 can be explained much the same way. Figure 3.15 shows the link utilization caused by both TDM traffic alone and TDM and BE traffic combined. As can be seen, the only BE packets that can be affected by the TDM traffic are the ones sent form the north-east quarter to the south-west quarter of the system. Since the backpressure on the links from the south-west quarter to the east is responsible for the saturation, the only effect that the TDM traffic has is a slight increase of the average latency of the BE packets. Furthermore, the results show once again that the achievable utilization rate of individual links can be much higher with TDM and BE traffic combined than with BE traffic alone, going as high as 82.2% in scenario 2.

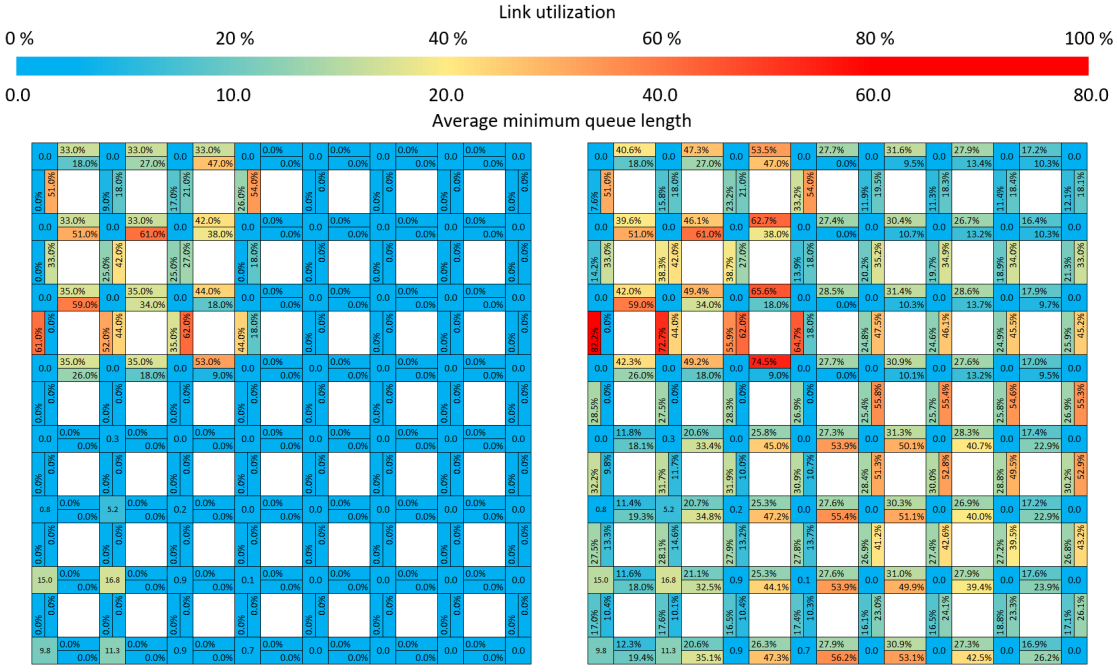
Similar to scenario 1, Figure 3.13 only considers the BE flit generation rate and not the overall achievable flit generation rate. The packet latencies with regards to the total flit generation rate is shown in Figure 3.16. Once again, the achievable flit injection rate is higher with a combination of TDM and BE traffic. However, it is still much lower than for scenario 1.

The results of the simulations for scenario 1 and 2 show that strictly dividing a system into a critical and non-critical application domain can be tricky and even disadvantageous. In scenario 2, part of the BE traffic is traversing the critical domain which is acceptable since the isolation of the critical traffic from the BE traffic is ensured by TDM. Another approach to dividing the system into different application domains would be



(a) 15% BE generation rate **(b)** 17.5% BE generation rate

Figure 3.14: Heatmaps scenario 2 - reference simulation runs



(a) TDM utilization **(b)** Total utilization

Figure 3.15: Heatmaps scenario 2 - 1+1 protection, 5% TDM, 15% BE flit generation rate

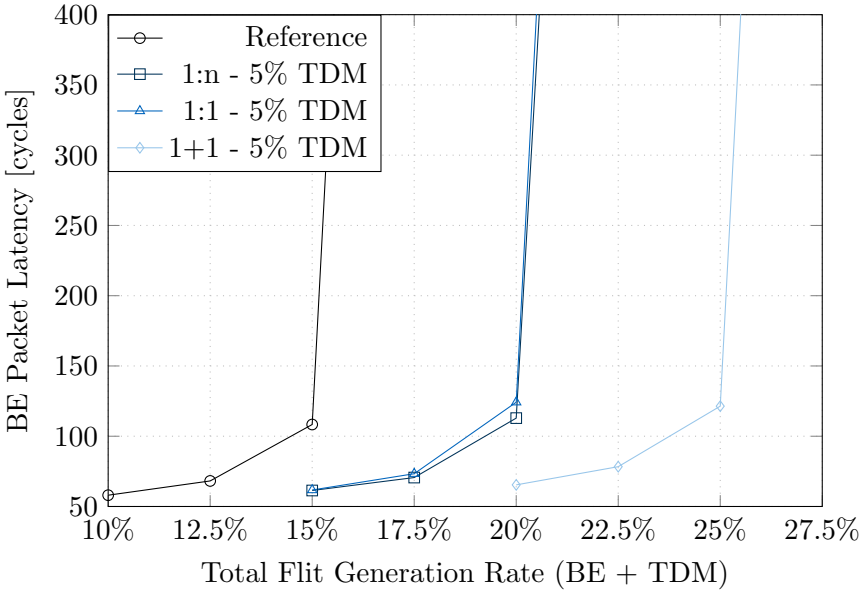


Figure 3.16: Average BE packet latency - Scenario 2, total flit generation rate

to only use rectangular arrays, which would be preferable for the BE traffic due to the XY routing. However, this heavily limits the possible mappings of the critical and non-critical applications in a system. In the 8x8 system, e.g., it would mean that the number of both critical and non-critical tiles should be a multiple of 8 in order to divide the system row- or column-wise. This can lead to problems when mapping the TDM channels, especially due to protection switching and the two disjoint paths required. On the other hand, the simulation results of scenario 1 indicate that it is preferable to spread out the critical traffic. This has another advantageous effect on the protection switching since in means that a permanent fault on a single link has a lower chance of affecting more than one TDM path, thereby increasing overall resilience.

In conclusion, the simulations show that the concept of protection switching in NoC works and that it can even help to increase the overall network utilization. The results suggest that, although the TDM traffic generally has an adverse effect on the available BE bandwidth, this effect is more than outweighed by the additional bandwidth for the critical TDM traffic. Even 1+1 protection seems to be a viable choice regarding its effect on the BE traffic, especially when considering its lower worst-case latency compared to both 1:1 and 1:n protection. However, more tests with a greater number of different scenarios are necessary to confirm or disprove these preliminary findings. Furthermore, since the two scenarios showed very different effects on the BE traffic, strategies to minimize the adverse effect on the BE traffic and, thereby, maximize the achievable network utilization should be researched. Therefore, the following Section 3.5 will propose different approaches of improving the network utilization. These approaches are then later evaluated in-depth in Chapter 5.

3.5 Improving Network Utilization

The proof of concept described in the previous Section 3.4 showed the general feasibility of protection switching in NoC and its effect on the packet switched traffic in a hybrid NoC. However, this first preliminary simulation based evaluation demonstrated that the safety-critical TDM traffic does have an adverse effect on the packet switched BE traffic and that bottlenecks can easily occur on individual links which cause the whole network to saturate at a comparably low rate. The simulations also showed that different application scenarios—or, more general, the way in which a given set of critical applications and their communication channels are mapped to the tiles and links of a system—have a severe influence on the amount of packet switched traffic that can be injected before the network saturates.

This begs the question of *how* to best map a given set of safety-critical applications with given bandwidth and latency requirements to a system in order to maximize the available bandwidth for BE applications and, thereby, the overall network utilization? In order to find an answer to this question different mapping strategies are proposed in the following Section 3.5.1. These mapping strategies can be used to map a given set of TDM channels to a system while considering protection switching requirements (i.e., mapping two disjoint paths for each channel) as well as various additional constraints that can be defined. The intention of the mapping strategies is to consistently find mappings that are beneficial for the packet-switched BE traffic and, thereby, maximize the overall achievable network utilization. The proposed strategies will later be evaluated in-depth in Chapter 5.

Parts of this section have been published in [3, 16, 17].

3.5.1 Mapping Strategies

A common approach of mapping critical and non-critical applications in a NoC-based mixed-critical MPSoC is to partition the NoC into critical and non-critical areas (e.g., [147, 102]). The reason is that this makes it easier to guarantee traffic isolation between the different traffic types. However, since this traffic isolation is already guaranteed by using TDM for the critical communication, partitioning the NoC in such a way is not necessary and there is generally more freedom when mapping the applications to the system[6].

In order to find strategies that, when followed, consistently produce good mappings it is important to first define what makes a mapping measurably *good* (or better or worse than another mapping). The goal in this thesis is to maximize the maintainable generation (and injection) rate of uniform random packet switched BE traffic for a given set of critical applications in a system, without the NoC going into saturation. For the remainder of this thesis the generation rate of uniform random packet switched BE traffic at which a given mapping causes the NoC to go into saturation will be referred to as the *saturation rate* of that mapping. Hence, the higher the saturation rate of a mapping, the more bandwidth is available for the BE traffic and the better the mapping. An

[6] Which does not mean the task is simple (cf. Section 3.5.2).

additional parameter that can be used to compare two mappings is the average latency from generation until delivery of a BE packet. This is especially useful to compare two mappings that achieve the same saturation rate.

This definition of *good* mappings, although intuitive, has a major flaw. In order to accurately determine both the saturation rate and the average packet latency it is necessary to test the mapping either in an implemented system or, at the very least, in a simulation. Both options pose a major inconvenience and, depending on the method chosen, can require a considerable amount of time and resources. Ideally, a mapping should have one or more quantifiable attributes that allow to infer its saturation rate—or at the very least an estimation of it—as well as a direct comparison of different mappings, without running tests or simulations. Four such attributes are defined in the scope of this thesis and are used as optimization objectives (minimization) for a mapping algorithm designed to find *good* mappings (cf. Section 3.5.3):

- *O1*: total number of reserved slots
- *O2*: standard deviation (SD) of reserved slots per link
- *O3*: SD of TDM path lengths
- *O4*: SD of number of critical tasks per row and column

All four optimization objectives directly affect how the critical applications and the TDM channels are mapped. However, the packet switched BE traffic—which the objectives are intended to improve—can only be considered indirectly. The reasoning behind these four optimization objectives is explained in the following.

O1: total number of reserved slots Minimizing the total number of reserved slots is an intuitive approach of trying to minimize the affect the TDM traffic has on the BE traffic. By minimizing the resources that are reserved for the TDM traffic more slots are free to be used by the packet switched traffic even if the TDM channels would use every slot that is assigned to them in every single TDM period. This objective naturally causes the critical applications to be mapped in clusters since the TDM path length will be minimal. However, individual clusters are not necessarily next to each other.

O2: SD of reserved slots per link By minimizing the standard deviation of the reserved slots per link it is ensured that all links are—more or less—evenly used by the TDM traffic. The intention of this objective is to avoid the creation of artificial bottlenecks by the TDM traffic. This objective, naturally, results in mappings that reserve a larger amount of slots for the TDM channels than mappings created by optimizing for *O1*. Furthermore, the TDM path length will generally be longer, especially when only few TDM channels are present. In essence, while *O1* typically results in the critical applications being clustered, *O2* typically spreads them out as much as possible.

Strategy	Optimization Goal (minimize)
S1	total number of slots (*O1*)
S2	SD of used slots per link (*O2*)
S3	SD of channel path lengths (*O3*)
S4	SD of number of critical tasks per row and column (*O4*)
S5	simultaneous optimization of *O1* and *O2*
S6	simultaneous optimization of *O2* and *O4*
S7	simultaneous optimization of *O1*, *O2*, and *O4*

Table 3.2: Mapping strategies

O3: SD of TDM path length Minimizing the standard deviation of the TDM paths means that all paths will have a similar length. This also means that all TDM channels have a similar worst-case latency. However, the main intention of *O3* is to spread the critical applications across the system, similar to *O2*. In contrast to *O2*, this spreading should only occur with growing number of paths. As the total number of paths increases it is no longer possible to keep all paths short and the critical applications are spread out resulting in similar path lengths. With only a small number of TDM paths, on the other hand, all paths can be kept similarly short which should lead to small clusters of the critical applications.

O4: SD of number of critical tasks per row and column By minimizing the standard deviation of the number of critical tiles in each row and column of the tiled system, this optimization objective produces mappings that evenly spread the critical applications across the system. Since all tiles that do not run a critical application are BE tiles, this objective also minimizes the standard deviation of the number of BE tiles in each row and column. Hence, this objective is the only one that directly affects the mapping of the BE tiles. It is also the only one that does not directly consider the mapping of the TDM paths throughout the NoC, i.e., the reserved resources on the links. The intention of this objective is to map the BE applications in a way that is beneficial for the static XY routing used by the packet switching in order to avoid bottlenecks created by the BE traffic itself.

The different optimization objectives can be used to define mapping strategies by optimizing for one or more of the objectives. Seven mapping strategies are defined and evaluated in the scope of this thesis. The strategies are listed in Table 3.2.

Strategies *S1* to *S4* are single-objective strategies that produce mappings respectively optimizing for *O1* to *O4*. *S5* to *S7* are multi-objective strategies that combine different optimization objectives. *S5* combines *O1* and *O2* with the intent to avoid the creation of bottlenecks for the BE traffic while at the same limiting the amount of reserved resources. *S6* combines *O2* and *O4* to define a strategy that considers both the mapping of the applications as well as the paths throughout the NoC. And finally, *S7* extends *S6* by additionally limiting the amount of reserved resources. The reason for choosing these

particular multi-objective optimizations is based on the evaluation of the single-objective optimizations and will be discussed in Section 5.2.1. Whether or not these strategies work in the intended way and reliably produce *good* mappings will be evaluated in Chapter 5 through tests performed in hardware on an FPGA.

3.5.2 Mapping Challenge

Mapping a given set of TDM channels to a TDM NoC is no trivial task to begin with, even without considering protection switching. The task becomes more challenging with increasing number of channels and decreasing slot table size, the latter meaning fewer channels can share a link. Depending on the mapping strategy, even the order in which channels are mapped can affect whether or not a set of channels can be mapped.

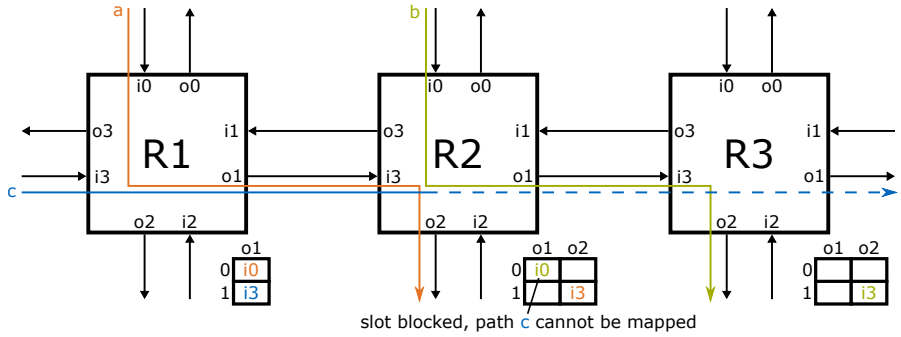
A simplified example with a slot table size of just two is shown in Figure 3.17a. In this example, the first free slot for a given path is chosen[7]. If the three paths are mapped in the order *a, b, c*, then path *c* cannot be mapped even though enough bandwidth is available on each link. The reason is that the free slots on the two links shown are not in consecutive cycles. If, however, path *c* is mapped first—shown in Figure 3.17b—then all three paths can be mapped.

In the example shown, the mapping of path *c* is initially neither blocked by just path *a* or *b* alone but by the combination of the two. Another solution in this particular scenario might be to shift the reserved slots of path *b* by one slot (i.e., use slot 1 in R2 and slot 0 in R3 for path *b*). However, this shift might be hindered by additional paths that share links with path *b* which are not shown in the figure. In this case, the mapping of path *c* might be blocked by a combination of multiple different paths—some of which it does not even share links with—although enough bandwidth would be free on all links that would be used by it.

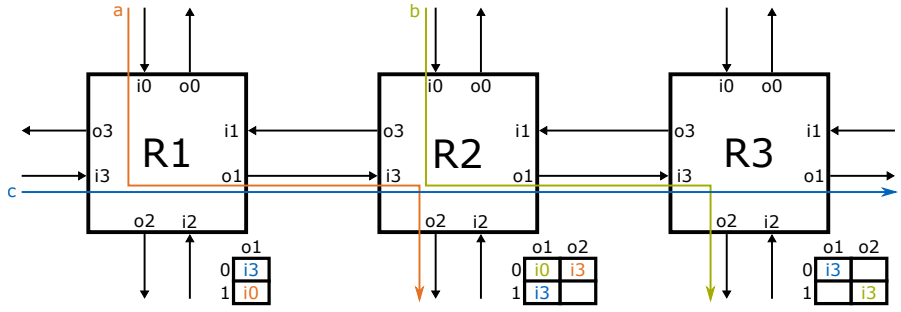
The example is, admittedly, very simple and pessimistic but illustrates the complexity of mapping TDM channels to a system. Similar—but more complex—examples can be found even for larger slot tables and more sophisticated mapping strategies.

The described mapping challenge becomes even more difficult in the scope of this thesis since protection switching must be considered. This not only means that the amount of TDM paths is twice as high as without protection switching—i.e., twice as high as the number of channels—but also that one half of the paths affect (i.e., reduce) the mapping possibilities of the other half of the paths, since each pair of paths has the same source and destination tile but must use disjoint routes through the NoC. An algorithm creating mappings from a given set of TDM channels must consider these mapping challenges. Additionally, it should be able to not just find *any* valid mapping but also consider a specific mapping strategy, as described in the previous Section 3.5.1.

[7] It is assumed here that the paths are already determined, e.g., following XY routing or following other mapping strategies such as the ones described in Section 3.5.1, but the slots for each path must still be determined.



(a) Path c cannot be mapped due to blocked slot by path b



(b) All three paths can be mapped successfully

Figure 3.17: The order in which TDM paths are mapped can affect the success of a mapping

3.5.3 Mapping Algorithm

The mappings of the two application scenarios evaluated in Section 3.4 was done *by hand* which is a tedious, time consuming, and error-prone task. In order to enable the evaluation of a large number of different traffic scenarios and mapping strategies it is necessary to automate this task by implementing a suitable mapping algorithm. The mapping algorithm used to create the mappings that are evaluated in Chapter 5 will be briefly introduced in the following.

For full transparency and disclosure: while both the definition of the optimization objectives and the mapping solution space exploration *are* contributions in the scope of this thesis, the algorithmic approach to finding these optimized mappings and the implementation of the mapping tool are *NOT* the author's contribution but have been done by Nguyen Anh Vu Doan. Therefore, the algorithm itself is only briefly covered in this thesis. For more details, the interested reader is directed to [3].

When representing both the NoC mesh and the critical applications to be mapped as graphs, mapping the channels to the NoC can be considered as similar to a subgraph matching problem. This problem is NP-complete, meaning it has a huge solution space. The requirement of two disjoint paths makes this problem even more complex. An exhaustive search is, therefore, not feasible which is why an approximate method was chosen for the mapping algorithm.

The mapping algorithms works in two steps. As a first step, an initial set of feasible mappings with disjoint paths that fulfill the application's bandwidth requirements is generated with a semi-greedy heuristic. In contrast to a typical greedy heuristic, the decisions of a semi-greedy heuristic are not deterministic but instead allow for a certain degree of randomness. In the second step, the initial set of mappings is used as a basis for a subsequent optimization process. The optimization is done with a genetic algorithm, specifically Non-dominated Sorting Genetic Algorithm (NSGA)-II (an extension of the original NSGA), which allows multi-objective optimization [148]. During the iterative optimization process in step two, new mappings are created through crossover and mutation operations which are designed in a way to ensure that the generated offspring(s) only contain feasible mappings. An overview over the mapping algorithm is given in Figure 3.18.

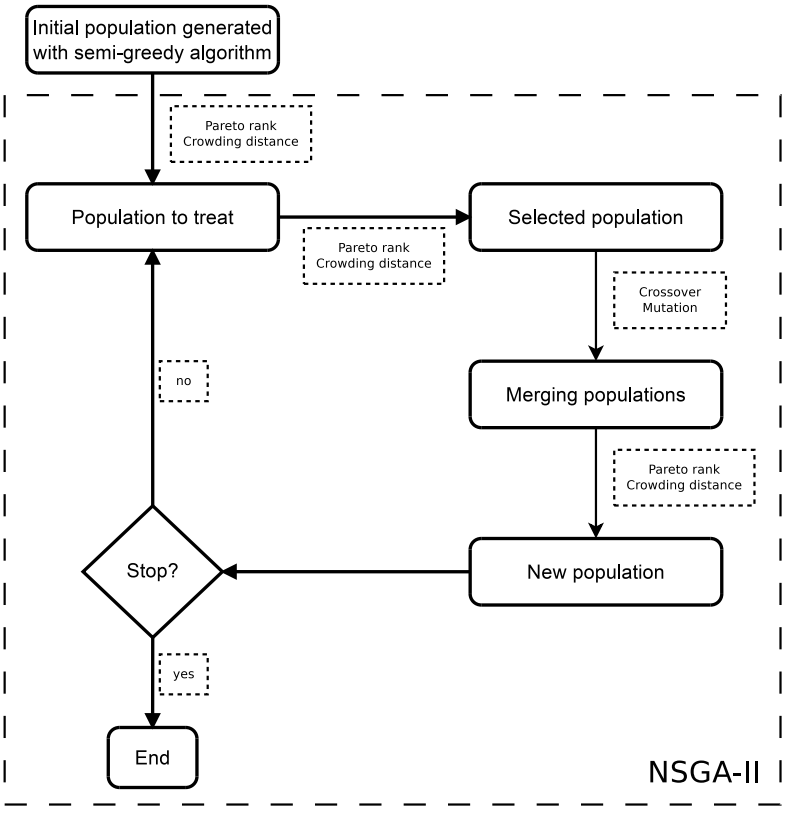


Figure 3.18: Mapping algorithm flowchart [3]

3.6 Summary

Mixed-critical MPSoCs require a scalable means of on-chip communication which must satisfy a number of requirements. Most importantly, the critical communication must provide hard real-time guarantees, be fault-tolerant, and guarantee isolation from other

critical and non-critical traffic as well as in-order delivery. The type of system considered in the scope of this thesis is a mixed-critical NoC-based MPSoC with safety-critical applications that behave predictably and that can be statically mapped at compile time, and BE applications with potentially unknown behavior that are dynamically started and stopped during system runtime. The fault model considered assumes that a fault can corrupt the flits of a TDM channel by affecting any parts of the channel between the sending and receiving NIs—i.e., wires, registers, and logic—and that this flit corruption is detectable by suitable means. Furthermore, the faults considered can be transient, intermittent, or permanent in nature.

The concept presented in this thesis us to use a hybrid NoC with TDM for critical traffic and packet switching for BE traffic. Using TDM for critical traffic not only allows to give hard real-time guarantees but also enforces isolation between different TDM channels as well as between critical and BE traffic. Fault-tolerance of the critical traffic is guaranteed by adopting protection switching. This is done by providing two disjoint paths for each TDM channel—a primary and a secondary path. To enable disjoint paths and remove the local link of the tiles as single points of failure, a second local link is added connecting the tiles to the NoC. If one of the paths is affected by a fault, the other one—and thereby the channel as a whole—is still fully operational, but the channel is no longer fault-tolerant. To return to a fault-tolerant state, a new disjoint path can be found and configured at runtime.

Three different protection switching versions are proposed in the scope of this thesis: 1:n, 1:1, and 1+1 protection. The different versions are evaluated regarding their estimated hardware overhead and by means of a formal timing analysis. The preliminary conclusion is that 1+1 protection—although using more resources and leaving less bandwidth for BE traffic—is the most promising candidate of the three protection switching versions.

A first proof on concept and evaluation is given based on a cycle accurate simulation of two different traffic scenarios. The simulations confirm that the TDM traffic has an adverse effect on the packet switched BE traffic. However, this adverse effect is tolerable and using a combination of both TDM and packet switched traffic can even increase the overall available bandwidth and achievable network utilization.

Different mapping strategies are proposed to maximize the available bandwidth for the BE traffic. Four different optimization objectives are introduced and used to define seven mapping strategies. The strategies are intended to reliably produce *good* mappings, i.e., mappings that allow for the most amount of BE traffic to be injected into the NoC before causing network saturation. Whether or not these strategies work in the intended way will be evaluated in Chapter 5. To enable this evaluation, a hardware architecture of the proposed NoC using protection switching is presented in Chapter 4.

4 Hybrid NoC Architecture Design and Implementation

With the concept for the hybrid NoC with protection switching being defined, this chapter describes the design of the architecture of the NoC and an implementation on FPGA. Furthermore, the architecture and implementation of two different systems using the hybrid NoC are presented.

One system—the evaluation system—is used for a more in-depth evaluation of the three protection switching versions. This is required in order to truly evaluate the effect that the safety-critical TDM traffic has on the packet switched BE traffic. Furthermore, the different mapping strategies proposed in Section 3.5.1 must be evaluated by determining both saturation rate and average packet latency of the BE traffic for a variety of different traffic scenarios. Such a Solution Space Exploration (SSE) of the possible mappings for a given traffic scenario would take a prohibitive amount of time to perform using cycle accurate simulations. Hence, a manycore system with the hybrid NoC implemented on an FPGA is used which allows to run a large amount of tests in a relatively short amount of time. The evaluation is presented in the subsequent Chapter 5.

The other system—the demonstrator system—is part of an interactive demonstrator that was developed and implemented for the ARAMiS II research project. This system demonstrates the traffic isolation and fault-tolerance of protection switching in NoC. It is connected to a host-PC enabling interactions with the demonstrator via a Graphical User Interface (GUI).

For both systems, synthesis results are presented which allows to evaluate the hardware requirements of protection switching in NoC. The results are also compared to the hardware cost of a state-of-the-art packet switched NoC which provides throughput guarantees (but not fault tolerance) in a mixed-critical system.

4.1 Baseline System Architecture

Implementing hardware is hard and designing the system architecture of an entire MPSoC from scratch is a time consuming task that is way out of the scope of this thesis. Therefore, an already existing framework, Open Tiled Manycore System-on-Chip (OpTiMSoC), is used as a basis for the architecture of the two systems [149]. OpTiMSoC is an open source framework which allows to design and implement MPSoCs with relative ease. Systems build with OpTiMSoC can be either simulated with Verilator or synthesized for an FPGA.

The framework provides processing tiles with a configurable number of processing cores, a local memory, and a NI to connect the tiles to the NoC. The different com-

ponents of a processing tile are connected by a local wishbone bus. A processing tile can contain up to four OpenRISC cores—a relatively small and simple processing unit [150]. Furthermore, the framework provides a basic packet switched NoC and a debug sub-system that allows to control and debug the implemented manycore system.

For the two systems designed and implemented in the scope of this thesis OpTiMSoC was extended by replacing the packet switched NoC with the hybrid NoC described in Chapter 3—including the NIs—and implementing additional modules. The architectures of these additional modules are described in Section 4.2. Furthermore, optional clock-domain-crossing was implemented to enable GALS designs.

Both the evaluation system and the demonstrator system are implemented on the "Xilinx Virtex UltraScale FPGA VCU108 Evaluation Kit" which contains a Xilinx Virtex Ultrascale FPGA (XCVU095). The FPGA has 1176 thousand logic cells, 768 DPS slices, and 60.8 Mbit on-chip memory. Additionally, the VCU108 provides 8GB of DDR4 memory and a large variety of off-chip interfaces. Information about the VCU108 evaluation kit is available online [151].

4.1.1 The Debug Sub-system

OpTiMSoC uses Open SoC Debug (OSD) as debug sub-system [152, 153]. OSD enables communication between the different modules of a generated system and a host-PC and is used for controlling and debugging the system. Currently, the debug sub-system is the only way to get information in and out of a system created with OpTiMSoC. Specifically, the sub-system is used to load the software into the local memory of the processing tiles, reset and start the system, and trace the behavior of the software during runtime including text output. OSD is also used for the communication with the modules implemented in the scope of this thesis and, therefore, its general structure is briefly described in this section.

OSD is divided into a hardware and a software part—implemented on system side and host-PC side respectively—and provides the infrastructure that allows the different hardware modules on system side and the software modules on the host-PC to communicate with one another. The hardware modules are connected to a Debug Interconnect (DI) which has the form of a 16 bit wide packet switched ring NoC. This DI is connected to all debug modules in the system as well as a host interface module which serves as an off-chip gateway. Details of the data exchange between the system and the host-PC differ depending on the used off-chip interface (e.g., Universal Asynchronous Receiver Transmitter (UART) or Universal Serial Bus (USB)). The debug sub-system uses the Generic Logic Interfacing Project (GLIP), a collection of off-chip communication interfaces which allows the off-chip interface to be treated like a FIFO without having to worry about the implementation details [154]. For the two systems implemented in the scope of this thesis USB 3.0 is used as off-chip interface.

The software part of OSD is implemented in C but provides an interface to the Python programming language. This allows to create a software connecting to and communicating with a specific system on a high level of abstraction with relative ease. The Python

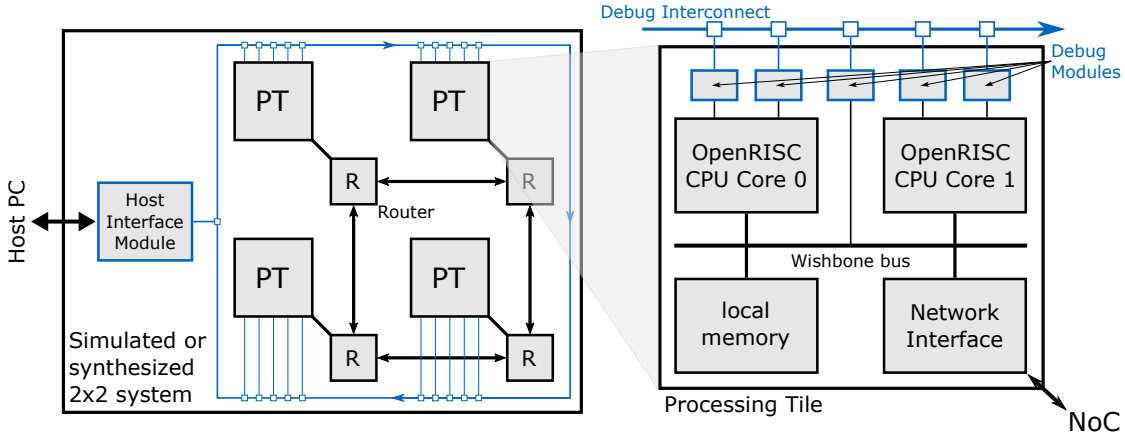


Figure 4.1: Overview over OpTiMSoC 2x2 reference design

interface also simplifies the implementation of host debug modules which are typically created to interact with a corresponding hardware debug module.

OpTiMSoC provides a basic project with 2x2 processing tiles and two cores per tile as a reference design. Figure 4.1 shows a simplified overview over this reference design with the debug sub-system. Note that not all debug modules and their specific functions are shown in this figure. For a more detailed introduction the interested reader is directed to [152, 153].

Limitations

Although OSD is a convenient solution for the communication between the implemented system and the host-PC it has some limitations which are briefly covered here.

One limitation is the width of the DI which is currently hard-coded to 16 bits and cannot easily be changed. This means that if a debug module has, e.g., registers that are wider than 16 bits the data must be serialized in order to be sent over the DI. Furthermore, the DI uses debug packets with a maximum size of 12 flits, three of which are used for source, destination, and packet type (cf. [152]). This means that larger messages must be divided into multiple packets which limits the bandwidth that is available for payload data to 75% of the DI's total bandwidth.

As mentioned above, the DI is a packet switched NoC with a ring topology. Consequently, it can quickly become a bottleneck with growing system size and it is impossible to give hard real-time guarantees. This must be considered for the architecture of the debug modules for the evaluation system and designing a suitable test setup for the intended SSE. Furthermore, this has consequences for the demonstrator system which will be further discussed in Section 4.3.2.

4.2 Module Architecture

The following sections describe the architecture of the hardware modules designed and implemented in the scope of this thesis, both for the NoC itself as well as the additional modules necessary for the evaluation system and the demonstrator system.

4.2.1 Network Components

First, the architecture and design choices of the different modules that build the hybrid NoC are discussed, as well as some modules necessary for testing.

4.2.1.1 Data Exchange Format

Before designing the individual hardware modules it is important to define the format or *handshake* that is used to exchange data between the NIs and routers. In addition to the flits some *flag* signals are required in order to interpret the flits correctly and to implement flow control which is important for the packet switched traffic.

The first flag that is necessary is a *valid* flag indicating that a flit is being transmitted over a link. This flag is needed for both pure TDM and packet switched NoCs. A pure TDM NoC would only need this one flag since the data transfer is streaming based and it is up to the application to define logical data structures, messages, etc.

A packet in a packet switched NoC typically consists of a header flit, multiple body flits, and a tail flit. The header flit holds information about the packet such as source, destination, path, packet class, etc. This is the only strictly necessary flit[1]. The body and tail flits carry the payload of the packet with the only difference between body and tail flit being that the latter one terminates the packet which is typically indicated by a *last* flag. The header flit can either be indicated by an additional flag or it can implicitly be identified by being the first valid flit after a tail flit as indicated by the *last* flag (or the first valid flit all together). The latter version is used for the architecture in this chapter. Lastly, a router needs to know if the downstream router has enough free buffer space to receive a flit. This information can be provided by a *ready* flag from the downstream router to the upstream router[2]. In summary, three flags are used for the packet switched traffic: a *valid*, a *last*, and a *ready* flag.

For the hybrid NoC it is necessary to have two individual *valid* flags for both TDM and packet switched traffic in order to tell them apart. Furthermore, the use of checkpoints for the protection switching and the resulting division of the TDM traffic stream into data *units*—as described in Section 3.2.5—makes it necessary to identify these checkpoint flits. Similar to indicating the header flits for packet switched traffic this can be done in two different ways: either by indicating the checkpoint directly with a *checkpoint* flag, or indirectly by indicating the last flit of a data unit—i.e., the last flit before a new checkpoint—with a *last* flag. Both versions were implemented and evaluated and their

[1] Some NoCs use single flit messages that are either used for configuration, handshakes, or directly carry a small payload.

[2] Some NoCs use more flow control that requires additional flags, such as credit based flow control, but at the very least a single *ready* flag is needed.

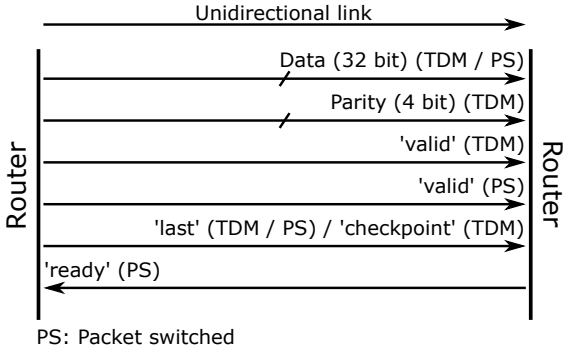


Figure 4.2: Signals of a unidirectional link (in brackets the traffic type they are used by)

respective implications are discussed in Section 4.2.1.2. For both versions the *last* flag that is also required for the packet switched traffic can be utilized, thereby bringing the total number of required flags to four.

In Section 3.1.2 it was mentioned that faults must be detected through suitable measures. For the implementation presented in this thesis parity bits are used to enable fault detection. Specifically, one parity bit is used for each byte of a flit resulting in four parity bits per link for the 32 bit wide hybrid NoC implemented for both the demonstrator and the evaluation system. The parity bits for each TDM flit are calculated at the sending NI—using odd parity—and are then sent alongside the flit. The receiving NI then checks if the odd parity holds true for each byte and the corresponding parity bit. This simple yet effective implementation of fault detection can detect up to four faults per flit, provided they affect different bytes. However, the fault detection mechanism is orthogonal to the contribution of this thesis and more complex fault detection mechanism could be utilized as well, if needed. Figure 4.2 shows a unidirectional link of the hybrid NoC as used in the two systems designed in the scope of this thesis with all its flags.

Faults on the flag signals have a high chance of having severe consequences since they directly affect the traffic flow, meaning they should be hardened against faults by suitable measures. However, as stated in Section 3.1.2, such faults are out of the scope of this thesis and are only briefly touched on as future work in Section 6.1.

4.2.1.2 Network Interface

The NI is—next to the router—one of the two main components of the hybrid NoC architecture and is arguably the most crucial one since it implements the protection switching function. The CPU of a tile—or other components—can access the NI via the tile internal 32 bit wide wishbone bus. Internally, the NI is divided into a packet switched and a TDM part with the former providing a single endpoint for packet switched traffic and the latter providing multiple TDM endpoints each supporting an outgoing and an incoming TDM channel. The number of TDM endpoints provided is a design parameter that can be adjusted depending on the requirements of the applications. Traffic can be

sent out on an endpoint by writing to its output buffer and traffic that has been received on an endpoint can be read from its input buffer. The dimensioning of these buffers will be discussed further down. When data has been received on an endpoint, an Interrupt Request (IRQ) is issued to indicate that data can be read from the input buffer. Two different IRQs are used in order to prioritize the critical over the non-critical traffic: TDM IRQ—a combined IRQ for the TDM endpoints—and BE IRQ—the IRQ for the BE endpoint.

The TDM endpoints are implemented to support 1+1 protection but with the option to switch one path off. The reason is that both 1:n and 1:1 protection have a similar effect on the BE traffic since they both only use one path at a time[3]. By being able to switch one path off, the same implementation can be used to evaluate the effect of the TDM traffic on the BE traffic for both 1:n/1:1 protection on one hand and 1+1 protection on the other hand.

The TDM endpoints must implement the protection switching function, i.e., take care of sending out the flits over both paths—potentially in different cycles—and ensuring that only correctly received flits are forwarded to the input buffer (and only once). As described in Section 3.2.5, checkpoints are periodically inserted into the data stream by a checkpoint insertion logic to implement the path synchronization at the receiving NI. When sending out a flit, a parity bit is added for each byte of the flit in order to enable fault detection. On the receiving side, the parity bits of the received flit are calculated again and compared to the received parity bits to check for faults.

Typically, a flit will not be sent out on both paths of a TDM channel in the same clock cycle since this would greatly limit the mapping possibilities. Hence, flits must be buffered until they have been sent out on both paths. This could be done with two individual *out*-queues—one for each path—with a minimum size of $S/2$ for a slot table size of S (one queue can never be more than $S/2$ flits *ahead* of the other queue). However, this would result in a large buffer overhead. Instead, a custom FIFO architecture with two individual read ports and pointers is used for the implementation. This design results in a relatively low hardware overhead that is mostly independent from the size of the FIFO.

As described in the previous Section 4.2.1.1, two different versions of path synchronization with checkpoints were implemented and both have different consequences for the implementation of the NI. With version 1, the data units of the TDM traffic are treated similar to packets of the packet switched traffic. The checkpoint acts as header of a data unit and the last flit of a data unit is indicated by a *last* flag. This is implemented with two individual *in*-queues in the TDM endpoints. Once an entire data unit has been received on a link without a fault and been stored in the corresponding *in*-queue the checkpoint evaluation logic checks if the received checkpoint matches the next expected checkpoint. On a positive check the data unit is written to the input buffer of the endpoint, otherwise the data unit is discarded.

[3] 1:n protection would potentially allow to pack TDM channels more densely and, thereby, allow more TDM channels to be mapped to a system in comparison to 1:1 and 1+1 protection. However, the number of TDM channels that can be mapped to a system will rarely be the limiting factor when mapping critical applications to a system, as will be discussed in Section 5.4.

With version 1, the receiving NI only switches between paths on a data unit granularity. An entire data unit is either accepted or discarded. Consequently, the checkpoint of a data unit is its sequence number ("wrapped around" on an overflow). However, the two individual *in*-queues cause a hardware overhead, even if the buffers are kept small. The design parameter affecting the minimal size of these queues is the number of data flits d that is sent between two consecutive checkpoints, which should be kept small for this implementation. A small d, however, increases the bandwidth requirements for the TDM traffic (cf. Section 3.2.5).

Version 2 of the path synchronization with checkpoints treats the TDM traffic as a continuous stream of flits and allows the receiving NI to switch between paths on a flit granularity. Consequently, the checkpoint of a data unit is the sequence number of the next flit ("wrapped around" on an overflow). Two local counters must keep track of the number of flits received on both paths and an additional counter needs to keep track of the number of flits that have been written to the input buffer. This implementation requires more logic for the checkpoint evaluation and handling of corner cases—such as updating checkpoints of a path that has been reconfigured at runtime—but does not need any additional *in*-queues. On the other hand, version 2 path synchronization can cause a higher load on the local bus—particularly with fast execution of the application's Interrupt Service Routine (ISR) or reaction of a hardware module reading from the NI—since the IRQ will be set even if only a single flit has been received. Furthermore, if a transient error corrupts a single flit on a path that is ahead of the other one, the receiving NI will—correctly—discard the flit and wait for the fault free flit on the other path. If, however, the first path *stays* ahead of the second one, the receiving NI will never be able to switch back to the first path even if flits on the other path are corrupted[4]. It is, therefore, important to allow the two paths to catch up with each other in regular intervals[5].

Both synchronization implementations support all three protection switching versions on the receiving side. At the sending side it is possible to use $1{:}n/1{:}1$ protection instead of 1+1 protection by continuously draining the read port of the *out*-queue of the deactivated path.

Four slot tables—one for each outgoing and incoming link—store the TDM configuration—i.e., which endpoint, if any, is connected to which link in each cycle—and control a multiplexer for each outgoing and de-multiplexer for each incoming link. When a slot is reserved for a TDM channel, the slot table will connect the output or input of the corresponding TDM endpoint to the reserved outgoing or incoming link. Slots can be reserved independently for both links and directions, meaning that different endpoints can send/receive on different links in the same cycle.

The packet switched traffic is only used for BE traffic, meaning that fault-tolerance is not necessary. Therefore, no parity bits are used for BE traffic and BE traffic is only sent/received on link 0. This helps to reduce the resource overhead of using two

[4] 1+1 protection is assumed in this example.

[5] In reality, this should rarely be a problem since TDM channels can typically not utilize 100% of the link's total bandwidth, as will be discussed further down.

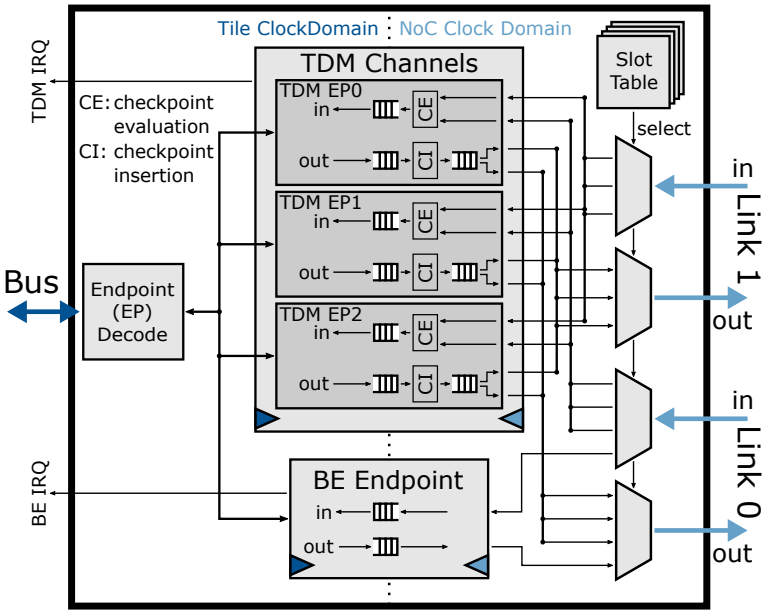


Figure 4.3: Hybrid NI with protection switching

links between each tile and the adjacent router. Whenever a slot on link 0—outgoing or incoming—is not reserved for or used by TDM traffic, BE traffic can be served in that direction. Only one endpoint is provided for BE traffic since that endpoint can be used for different traffic streams.

Figure 4.3 shows the architecture of the NI with its sub-modules[6] (the fault detection based on parity bits is not shown in the figure). In order to enable a GALS design, the NI provides clock-domain-crossing FIFOs in the endpoints which result in additional logic.

Discussion: Design Choices and Buffer Dimensions

As discussed in Section 2.2.5, using TDM for the critical traffic allows to give hard real-time guarantees. However, this guarantee can only be given between two endpoints of a TDM connection, i.e., the two NIs. In order to give hard real-time guarantees between two communicating applications it is necessary to also consider writing to and reading form the NI. Although this is out of the scope of this thesis, the following paragraphs provide a discussion about the general problem, possible solutions, the design choices made for the implementation in the scope of this thesis, and the implications this has for the buffer sizes of the NIs.

As described in the previous section, data is sent out and received over the different endpoints by writing to and reading from their respective output and input buffers. Both requires processing time and is, thereby, time consuming especially for larger chunks

[6] This is the design implementing version 2 of the path synchronization. Version 1 would require two individual *in*-queues before the checkpoint evaluation of the TDM endpoints.

of data. This additional processing time must be considered when implementing an application. The most critical situation is a buffer overrun on the input buffer of a TDM endpoint. Since incoming TDM traffic cannot be stalled, this inevitably results in data loss which must be avoided at all cost. Therefore, the TDM IRQ that is set whenever TDM traffic is ready to be read from an endpoint of the NI must have the highest priority.

In the current implementation the TDM IRQ is a logical OR combination of the individual IRQs of all TDM endpoints. When reading from the NI, the ISR that is executed must first determine which endpoint has data to be read. This is done by reading a bit vector from the NI which holds all individual IRQs[7]. It is then up to the developer to implement an appropriate arbitration scheme between the different endpoints inside the ISR. Similarly, it is up to the developer to ensure that the application can keep up with the amount of traffic that is received by the NI. Particularly, the developer must ensure that incoming data is always drained fast enough from each input buffer in order to avoid a buffer overrun. At the same time, an application must still have enough remaining time to do all required processing on the received data as well as sending out whatever responses or messages to subsequent processing nodes are necessary. Overall, the tile internal bus and the performance of the processing element are most likely to be the limiting factor regarding the maximum bandwidth for both incoming and outgoing traffic. This is especially the case if the NoC is clocked at a higher speed than the tile.

In order to avoid buffer overruns the buffer size was chosen to be rather large for the two implemented systems presented in this thesis. For both the input and output buffers of each endpoint a size of 512 flits was chosen, resulting in a buffer size of 2KiB (with a flit size of 32 bit). This not only gives a large safety margin to avoid buffer overruns but also allows to effectively utilize the FPGA's Block RAM (BRAM) modules, thereby helping to minimize the required hardware resources of the implementation—i.e., the number of logic and register cells of the FPGA.

Naturally, large *in*-buffers can only temporarily mitigate times when traffic is received with a higher bandwidth than it is drained with. In order to guarantee that no buffer overruns can occur in the two implemented systems two different approaches were taken. The evaluation system generates and consumes data in special hardware modules which use the same clock as the NoC does. Furthermore, the combined amount of incoming and outgoing traffic of a tile is never higher than the bandwidth of the local bus. In the demonstrator system the amount of data that is sent over a TDM channel is limited in software and new data is only sent when the input buffer is known to be empty (further described in Section 4.3.2).

A possible way to decrease the processing power that is required—or blocked—by actively writing data to and reading it from the NI would be to utilize Direct Memory Access (DMA). A common approach is to attach a dedicated DMA controller to a bus

[7] This effectively limits the possible number of TDM endpoints in the current implementation to 32. A possible way to overcome this limitation would, e.g., be to use additional bit vectors or to implement arbitration directly in the NI.

that takes care of moving large data chunks without direct involvement of the CPU. This way, data could be sent via the NoC by initiating a DMA transfer from the local memory to the outgoing buffer of an endpoint. On the receiving side the NI would then have to initiate a DMA transfer into the local memory and set an IRQ once the transfer is finished.

In [155] it was proposed to add a DMA controller directly to the NI and utilize a dual read port memory in order to minimize the required buffer space and make DMA transfers over the NoC possible. The approach is promising but cannot directly be used with protection switching. A way of enabling DMA transfers for the hybrid NoC with protection switching was evaluated and could significantly improve the achievable transfer rate and overall performance but lead to increased complexity and hardware requirements and is, therefore, not part of the final architecture presented in this thesis [156].

Another situation that must be avoided is a critical task running on a processing tile being interrupted by faulty or malicious BE traffic. To ensure this, the BE IRQ should be disabled on tiles running critical applications. However, this rises another undesirable, although non-critical, issue regarding the BE traffic. If packet switched traffic is sent to a tile, be it intentionally or as a result of a fault, but is never read from the NI, backpressure will sooner or later block parts of the NoC for BE traffic. In order to avoid this, the endpoints of a NI can be enabled or disabled (which is the default). Flits received by a disabled endpoint will simply be discarded. This means that the software running on a processing tile must enable the endpoints it intends to use as well as enable the corresponding IRQs.

This leads to the question of how to know when the receiver of a message has finished its startup routine and is ready to receive data. This question, however, is out of the scope of this thesis. In the current implementation—which is similar to how this is handled in OpTiMSoC—a special configuration message can be sent to the BE endpoint of a remote NI. A response to this message will be generated and sent back in hardware in case the BE endpoint is enabled. A similar system could be implemented for the TDM traffic.

4.2.1.3 Network Router

The design of the hybrid NoC router is fairly straightforward as it largely follows the architecture depicted in Figure 3.1 (described in Section 3.2.2). The packet switched traffic is implemented with static XY routing, without virtual channels, and with round-robin arbitration between different input ports that request the same output port. Wormhole switching is used which means that the BE buffer size at the input ports is independent from the packet length and could be as low as just two flits[8]. The input buffer size is a design parameter that can be defined at synthesis time.

Figure 4.4 shows an overview over the router architecture. The input port splits the traffic into TDM and packet switched parts. The TDM traffic can optionally be

[8]Two flits is the minimum in order to enable continuous transmission of flits.

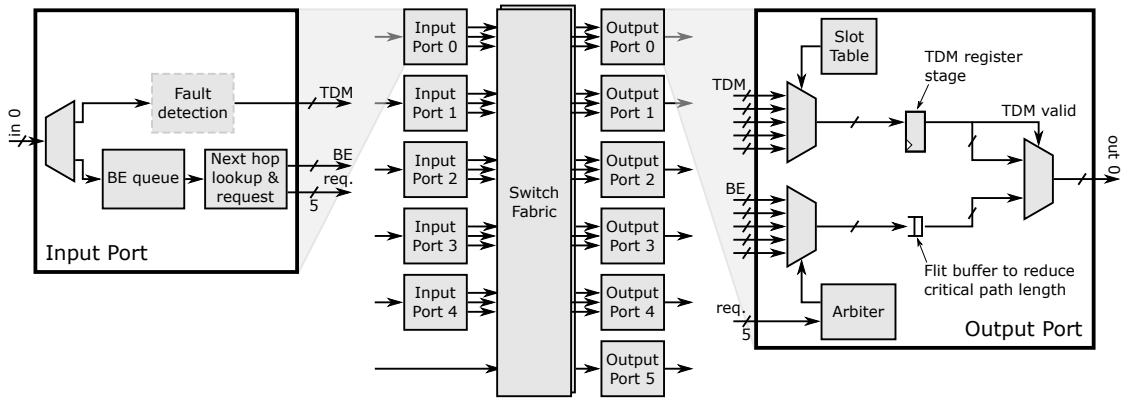


Figure 4.4: Hybrid router architecture

checked by a Fault Detection Module (FDM) which is described in the Section 4.2.1.5. This, however, is only used in the demonstrator system as for protection switching it is sufficient to check for faults in the receiving NI. The packet switched traffic is added to the BE queue and the output port of a packet is determined. A request is then issued to the output port to which the packet must be forwarded[9].

The switch fabric is split into two planes: one for the TDM traffic and one for the packet switched traffic. It is implemented as a set of wires that connect all input ports with all output ports. A connection from an input port to an output port is made active by multiplexers in the output ports. In principle, this design also allows multicasts from one input port to multiple output ports (for the packet switched traffic this is currently not supported by the input ports, for TDM traffic this is supported but out of the scope of this thesis).

The slot table in the output port determines from which input port a TDM flit is stored in the register stage. If a valid flit is stored in the register stage then this flit is forwarded to the downstream router in the next clock cycle. Otherwise, the packet switched BE traffic is served. The arbiter in the output port arbitrates between competing requests for packet switched traffic from different input ports. A small buffer stage is used to reduce the critical path length.

Port 5 is special in that it does not support packet switched traffic. This port is connected to the second local link which is only required for the protection switching. The router is implemented in the depicted modular way for two reasons. First and foremost, the design ensures that no single fault in one of the sub-modules or the connecting wires can affect more than one input or output port. Second, the design allows to remove individual ports that are unconnected. This significantly reduces the size of routers at the edges and corners of a 2D mesh NoC.

In the current design, the main resources that are shared between the TDM and the packet switched traffic are the links between the routers. It would also be possible to use two completely separate NoCs instead which are only connected by the NIs in the tiles.

[9] A packet cannot be forwared to the same port it arrived on.

However, in a typical MPSoC the wires between the routers have a considerable length and, hence, both a significant area requirement and power consumption. Furthermore, this implementation would mean that the TDM NoC would experience the typical underutilization which in this case could not be used by the BE traffic, thereby lowering the combined utilization of the two NoCs. It is, therefore, the authors believe that the hybrid approach, potentially with additional features for the BE part, is more suitable for communication in mixed-critical MPSoCs. A possible improvement of the design would be to use a single switch fabric for both traffic types which would be possible with adjustments in both the input and output ports. This, however, is out of the scope of this thesis.

Traffic Isolation The implementation of the router output port directly supports time-slot stealing as packet switched traffic will be served whenever no valid TDM flit is stored in the register stage. However, as mentioned in Section 2.2.5 this practice weakens the traffic isolation between the TDM traffic and the packet switched BE traffic and potentially opens a vulnerability to side-channel attacks. This, could be avoided by forbidding time-slot stealing entirely. Another way would be to slightly adjusting the implementation of the output port and the slot table in order to provide a secure TDM traffic class for which time-slot stealing is deactivated. This could be achieved by a special flag in each slot of the slot table that, when set, would disable time-slot stealing for this slot. This way, the reserved slot would be blocked for packet switched traffic whether it is being used or not.

4.2.1.4 NoC Control Module

As described in Section 2.2.4, there are generally two different possibilities to configure a TDM NoC: distributed or centralized. A centralized NoC manager—the NoC Control Module (NCM)—is used for the configuration of the hybrid NoC in the scope of this thesis. The NCM is directly wired to all slot tables of the routers and NIs of the system. This solution does not scale well but is sufficient to demonstrate and evaluate the capabilities of the hybrid NoC with protection switching. The NCM is designed as an OSD debug module that is connected to the DI, thereby allowing direct communication with a host-PC. A corresponding host debug module to interact with the NCM is implemented in Python on the host-PC.

Other than slot table configuration the NCM has four additional functions: enabling/disabling individual links for the endpoints of the NIs, monitoring the traffic on all links of the system, monitoring the FDMs (described in the following Section 4.2.1.5), and enabling/disabling the Fault Injection Modules (FIMs) (described in Section 4.2.2.4). Monitoring both the FDMs and the traffic on the links as well as controlling the FIMs is only necessary for the evaluation system and the demonstrator system designed in the scope of this thesis. These functions would not be required in a deployed system. All

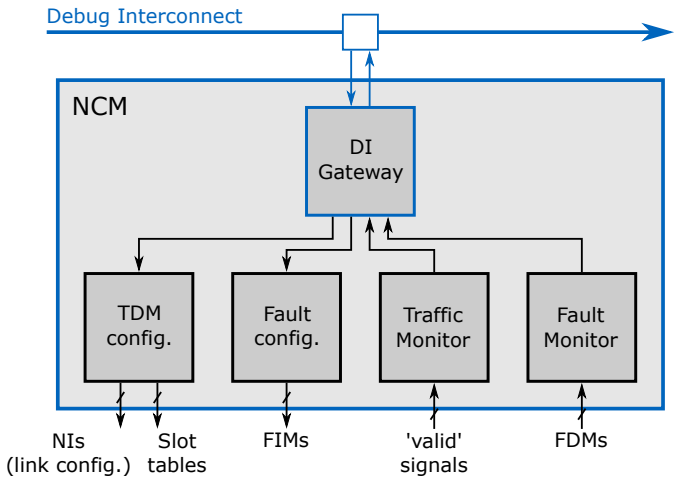


Figure 4.5: Network Control Module (NCM) overview

tasks are handled in different sub-modules of the NCM[10]. Figure 4.5 gives an overview over the NCM.

To configure a slot table the host-PC sends a configuration message to the slot table configuration sub-module of the NCM via the debug sub-system. This message defines a specific slot in a slot table that is to be configured as well as the configuration to be written to this slot. After a TDM channel has been fully configured it must be enabled. To so do, another configuration message is sent and the links that are used—only one link for 1:1 or 1:n protection and both links for 1+1 protection—are enabled in the endpoint of the sending NI. This is necessary for two reasons. First, it ensures that no data is sent before the channel is configured—which would result in data being lost—and second, it configures the sending endpoint to drain the unused *out*-queue which would otherwise block the transmission. When a TDM path must be reconfigured—e.g., due to a fault—the path must first be disabled in the sending endpoint and must only be re-enabled once the reconfiguration is finished.

Injecting faults works similar to configuring TDM channels by sending a dedicated debug message for every link on which a fault should be injected. This, however, is only used by the demonstrator system in order to interact with the system and demonstrate the fault tolerance of the critical communication.

To monitor the network utilization, the traffic monitor sub-module simply collects and accumulates the status of the *valid* flags for both TDM and packet switched traffic for each link and in each clock cycle. The consequence for the packet switched traffic is that the module does not count the amount of transmitted BE flits but rather the amount of cycles in which a link is blocked by a packet switched flit. This is intentional as it allows to identify links that are heavily affected by backpressure. The sub-module is used by both the evaluation and the demonstrator system. Naturally, monitoring the utilization of each link results in a high amount of data that must be sent off-chip via the debug

[10]Slot table configuration and link enabling/disabling is integrated in a single sub-module.

sub-system. To not overload this system the information is only sent out periodically for the demonstrator system (every 250ms). For the evaluation system the information is collected for each test and only sent out once at the end of a test.

Lastly, the fault monitor records and accumulates the amount of faults that are detected by the FDMs in the system. Just as the fault configuration sub-module this module is only used for the demonstrator system. Similar to the traffic monitor used in the demonstrator system the number of recorded faults is sent out only periodically (every 250ms).

4.2.1.5 Fault Detection Module

As described in Section 4.2.1.1, parity bits are used to enable fault detection. The FDMs are, therefore, just realized as a series of XOR gates checking the parity of each byte in a flit and the corresponding parity bit line. One such FDM must be instantiated for each incoming link of each NI in order to enable end-to-end fault detection. However, for the demonstrator system a FDM is additionally instantiated in each input port of each router. This way it is possible to detect the exact link on which a fault occurred.

4.2.2 Additional Modules

The modules described in this section are not part of the hybrid NoC but are additional modules required by either the evaluation system or the demonstrator system.

4.2.2.1 Traffic Generator Module

The evaluation system is implemented for the SSE of different system parameters and mapping strategies. It is intended to verify or falsify the initial results obtained from the simulations in Section 3.4.2 as well as evaluate the mapping strategies defined in Section 3.5.1. In order to enable this evaluation it is crucial to reliably generate traffic with a defined generation rate in the tiles of the system. The generation rate must be adjustable at runtime in sufficiently small steps (a step width of 1% is commonly used in related work and is, therefore, also chosen for the evaluation system). Additionally, it must be possible to track the average latency of the BE packets sent over the NoC in number of clock cycles. Both tasks are very difficult to reliably achieve in software.

In order to gain meaningful results the evaluation must be conducted with a sufficiently large NoC. Just as for the simulation a system size of 8x8 tiles is used. However, a system with 64 OpenRISC cores and sufficient local memory to run an application to generate and consume traffic would be too large for the selected UltraScale FPGA. For these reasons, a special Traffic Generator Module (TGM) is used in order to accurately generate and consume traffic as well as keep track of packet latencies in hardware.

The TGM replaces the processing core and local memory of each tile and is directly connected to the NI using the wishbone bus. This way, a *generator tile* is created that is used in the evaluation system instead of the usual processing tile. From the perspective of the NI the TGM must behave the same way as a CPU would. To enable the SSE of mapping algorithms and obtain results that are comparable to the ones from the

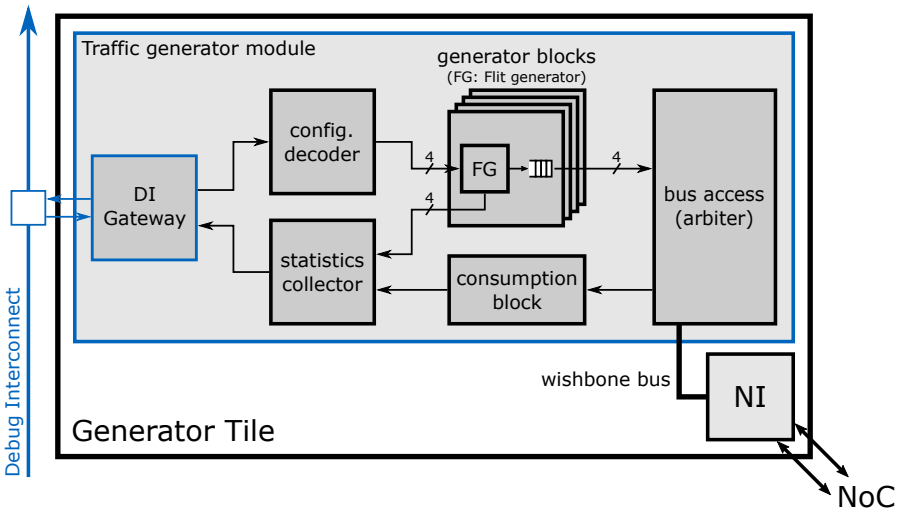


Figure 4.6: Generator tile overview

simulation described in Section 3.4.2 the TGM must work in a similar way. Specifically, the module must:

- be able to generate random uniform packet switched BE traffic,
- be able to generate TDM traffic for multiple TDM channels,
- send the generated traffic out and read received traffic via the NI,
- keep track of generated and consumed traffic as well as packet latencies,
- allow traffic generation rates to be configurable at runtime,
- and report all gathered statistics back to the host-PC.

All these functions are implemented in different sub-modules of the TGM. Figure 4.6 shows the architecture of the generator tile with the TGM, its sub-modules, and the NI.

The core function of the TGM—the generation of traffic—is implemented in multiple *generator blocks*. One of these blocks generates BE traffic, the other (configurable number of) blocks each generate traffic for one TDM channel. Each generator block contains a Flit Generator (FG) that can be configured to generate flits with different generation rates. The individual FGs of the generator blocks are configured by the host-PC via a special *configuration decoder* module. A *bus access* module acts as arbiter and forwards the generated traffic to the corresponding endpoints of the NI. This module also reads received traffic from the NI—if either the BE or TDM IRQ are set—and forwards the traffic to a *consumption block*. The consumption block evaluates the received flits and determines the type (TDM or BE), source, and latency of the message. Finally, a *statistics collector module* gathers the information about both the generated and received traffic

and sends the results out to the host-PC. These sub-modules will be further described on the following pages. The TGM is an OSD debug module and is connected to the DI to allow communication with the host-PC.

Generator Block

The generator blocks are the heart of the TGM. Since the reliable traffic generation is crucial for obtaining meaningful results—particularly the generation of sufficiently uniform random background traffic—this sub-module is described in greater detail.

Each block generates either TDM or packet switched BE traffic that is then injected into the NoC. The blocks each contain a FG that manages the traffic generation. For TDM traffic the generation is relatively trivial as the flits can simply be generated with a steady rate—e.g., 5 messages of 8 flits every 400 clock cycles for a generation rate of 10%[11]—and a simple FSM can output the flits to the bus access module. However, this method cannot be used for the uniform random packet switched BE traffic which typically does not show such steady behavior. Therefore, each BE FG is equipped with a configurable Random Number Generator (RNG) and additionally has a FIFO buffer to which the generated flits are written[12]. The following paragraphs mainly focus on the RNG as it is used in the BE FG.

Generating random numbers on an FPGA is no trivial matter [157]. RNGs are divided into two categories: *true* random number generators and *pseudo* random number generators. Pseudo random number generators are algorithms that produce, as the name suggests, no truly random sequence of numbers but instead a deterministic sequence based on an initial value, the *seed*. True random number generators, on the other hand, generate a truly random sequence of numbers based on measurements of a physical phenomenon such as thermal noise. Generating true random numbers on an FPGA is generally possible but typically requires a considerable amount of logic, or many clock cycles, or both (e.g., [158, 159]). This is not feasible for the evaluation system since each generator block needs its own RNG. However, true random number sequences are typically only necessary for cryptographic use cases which the evaluation system is not. Therefore, a pseudo RNG is used for the traffic generation. To ensure that not all FGs generate the same traffic, each RNG is seeded with a different value that is randomly created on the host-PC[13]

The core of the FG's RNG is a 32 bit wide Linear-Feedback Shift Register (LFSR) with the maximum length feedback polynomial described in [160]. XOR-feedback is used which means that the LFSR must be seeded with a non-zero value (all bits zero is a 'forbidden' state for XOR-feedback since it constitutes a locked state). Once started—

[11] The checkpoints are ignored in this example. To consider the checkpoints it is necessary to determine how many checkpoints will be inserted for a given number of flits and to then adjust the number of generated flits in order to achieve a desired injection rate.

[12] To be exact, the RNG is also used in the TDM FG but only steady rates are used in the scope of this thesis which is why the FIFO is omitted in order to reduce the resource requirements.

[13] This, of course, only shifts the problem of generating sufficiently randomized number sequences. However, every modern PC provides well tested libraries to generate such sequences.

and seeded with a value other than '0'—the LFSR cycles through all $2^{32} - 1$ possible non-zero values in a pseudo random order determined by the feedback polynomial.

One problem when generating (pseudo) random numbers is ensuring the even distribution of all possible values for an arbitrary range of numbers. A brief example: When using a 4 bit LFSR with the maximum length feedback polynomial and XNOR-feedback, it (pseudo randomly) cycles through the values 0 to 14 (all bits '1' is the forbidden state for XNOR-feedback). The distribution of the numbers 0 to 14 is even. However, if only random numbers from 0 to 9 are needed there is a problem. One possibility would be to wait for the next valid number that is in the desired range. This would result in a delay which might not be tolerable for some applications (e.g., for random traffic generation it would result in a lower generation rate caused by the delay cycles). Another possibility would be to "cut off" any numbers higher than the upper limit of a desired range. This, however, means that the numbers are no longer evenly distributed. For the mentioned example this would mean that the numbers 10 to 14 would be treated as 9, thereby giving the number 9 a probability of $\frac{6}{15} = 0,4$ and all other numbers a probability of $\frac{1}{15} = 0.0\overline{6}$.

A very common approach in software is to use the modulo operation in order to determine the reminder of an integer division, thereby limiting a random number to an arbitrary range. With $r = n \bmod u$ follows $r \in [0, u], \forall n \in \mathbb{N}, u \in \mathbb{N}^+$. However, the distribution of r is only even if n can get infinitely large. For the mentioned example a 10 to 14 would result in a 0 to 4. This, again, violates the even distribution of numbers by giving the numbers 0 to 4 a probability of $\frac{2}{15} = 0.1\overline{3}$ and the numbers 5 to 9 a probability of $\frac{1}{15} = 0.0\overline{6}$.

For the RNG of the FG, the problem of even distribution of the generated random numbers is addressed in two ways:

1. The LFSR has a (much) larger range than the random number(s) created from it.

2. Random ranges are limited to ranges that cover a power of 2 number of numbers (e.g., 0 to 255 or 10 to 41).

For the reminder of this thesis an XOR-feedback is assumed for LFSRs unless otherwise specified.

When a maximum length feedback polynomial is used, an n-bit LFSR cycles through all possible $2^n - 1$ possible non-zero values. However, it is often desirable to have random ranges start with zero. One way of achieving this with the mentioned LFSR is by dimensioning it for a larger range than needed and only using some of the bits. An example would be a 4 bit LFSR that is used to generate a 3 bit random number. The three low bits can have all eight values from 0 to 7. However, since the 4 bit LFSR can only cycle through the fifteen values from 1 to 15 the three bit value cannot be zero when the Most Significant Bit (MSB) is set. This means that in a full cycle, zero will appear only half as often as any other number which, again, results in an uneven distribution of the generated numbers. This effect, however, can be weakened by making the LFSR even larger. With two additional bits any non-zero value appears 33% more often than zero (4 times vs 3 times in one full cycle) and with ten additional bits the

additional percentage is less than 0.01%. This can still make a difference—especially for long sequences of random numbers and depending on the application—but is negligible for the evaluation in the scope of this thesis.

As mentioned above, a 32 bit wide LFSR is used in the FG's RNG. The rate at which traffic is generated can generally be controlled with three different parameters: the number of flits in a packet (or message), the number of packets per burst, and the number of cycles before the next burst is generated. The LFSR is used to create random numbers for two of these parameters: the number of packets in a burst and the wait cycles until the next burst. A fixed packet length of 15 flits is used in the scope of this thesis since the other two parameters are sufficient to achieve the desired generation rates.

The lower 22 bits of the LFSR are used to generate the wait cycles until the next burst, the upper 10 bits are used to generate the burst length. To enable different ranges of the generated random numbers, a bit vector can be configured as a mask for both of these parameters. Furthermore, an offset can be defined that is added to the randomly generated numbers. This enables the generation of virtually arbitrary ranges of random numbers as long as the range covers a power of 2 number of numbers. It is even possible to configure the FG to generate bursts with a fixed size and a fixed periodicity by setting both masks to zero and setting the offset to the desired value.

When the FG is enabled and starts traffic generation, the RNG reads the current random value for the wait cycles from the LFSR and starts a wait counter. Once the wait counter matches the previously read value, both the random number for the burst size and the next random number for the wait cycles are read from the LFSR and the wait counter is restarted. The random number for the burst size is then used by the FG to generate the individual flits.

Depending on the configuration it is possible that the generation of a new burst is triggered before the previous one is fully generated, thereby causing a *super*-burst. In order to avoid the new burst "overwriting" the old burst—which would lead to a lower generation rate—a small FIFO is used in the FG to buffer the burst size. Naturally, this buffer can overflow either due to being too small, having "bad luck" with the random numbers, or because the configured generation rate is over 100% (which is possible and must be avoided). Another cause of overruns is backpressure from the NoC. For BE traffic this is an indicator that the traffic injection rate cannot keep up with the traffic generation rate. For TDM traffic this happens when the traffic generation rate is higher than the bandwidth that has been reserved for a TDM channel. Therefore, buffer overruns are monitored by the statistics collector module. During all of the test runs evaluated in Chapter 5 such overruns did never occur for TDM traffic.

Figure 4.7 shows the architecture of the RNG and the LFSR used. In combination with the offset, the implementation allows for burst lengths of up to $2^{11} - 2 = 2046$ packets and wait periods between two bursts of up to $2^{23} - 2 = 8\,388\,606$ cycles. These possible ranges are more than enough for the SSE and allow to configure the desired traffic generation rates with sufficient precision.

The FG contains an FSM that generates the individual packets/messages of a traffic burst. The FSM adds a timestamp which is read from a timestamp counter that is

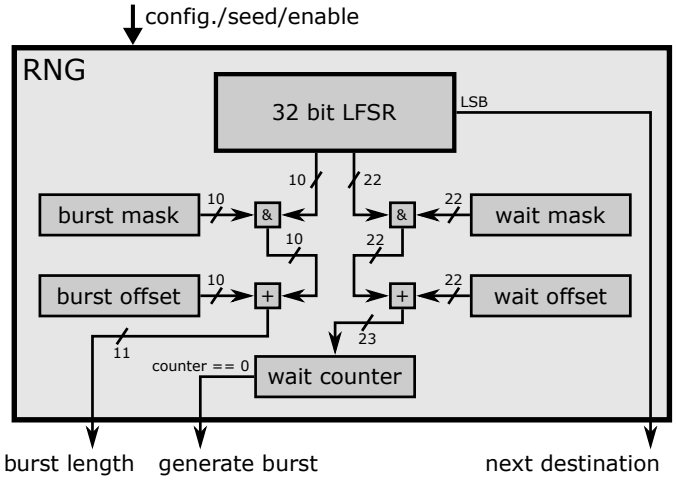


Figure 4.7: Random number generator (RNG)

started when the traffic generation starts. For the BE FG, the FSM must also generate the header flit which contains the source and destination of the packet. All possible destinations—i.e., IDs of the BE tiles—are stored in a central list in the test setup module (described in Section 4.2.2.2). The destinations are stored in the list in random order and each FG has an individual pointer pointing to an entry in this list. Each pointer is seeded with a different random value when the FG is configured and the pointer is incremented—or wrapped around from the last to the first list element—in each cycle where the Least Significant Bit (LSB) of the RNG's LFSR is '1'.

To account for different traffic types, the FG can generate bursts in two different ways. The destination is either determined once for all packets of a burst, or individually for every single packet. The former—hereafter called *burst mode*—is the typical burst where all packets are sent to the same destination. The latter—hereafter called *batch mode*—results in most packets of a burst being sent to different destinations, thereby mimicking a system that uses a lot of multicasts or broadcasts.

In the simulation in Section 3.4 the buffer queues of the sending NIs could grow infinitely large (i.e., the size was only limited by the capacity of the hardware on which the simulation was executed). This meant that the average latency between packet generation and delivery would eventually grow infinitely large as well. Naturally, this is not possible for a hardware implementation. The FIFO queues in the BE generator blocks can only have a finite size—512 flits in the implemented system—which means that buffer overruns will eventually occur if traffic is continuously generated at a higher rate than what can be injected into the NoC. This means that the latency will not grow indefinitely—in contrast to the simulations—and instead will eventually reach a maximum. However, the latency will still show a steep increase around the point at which the traffic injection rate can no longer keep up with the generation rate. Furthermore, the buffer overruns serve as an additional indicator that the saturation rate is reached. Therefore, these buffer overruns are also monitored by the statistics collector module.

Configuration Decoder

The configuration decoder module decodes messages from the host-PC and configures the FGs in the generator blocks accordingly. Specifically, the seed, masks, and offsets in the RNGs are configured. Furthermore, the module writes the seed value to the destination pointer of the BE FG and enables or disables the individual generator blocks.

Bus Access Module (Arbiter)

The bus access module handles the read and write accesses to the NI via the wishbone bus. Furthermore, it arbitrates write requests to the NI from the different generator blocks and read requests from the NI triggered by the TDM and BE IRQs. The arbiter must ensure that the critical TDM traffic is served first and that buffer overruns in the receiving TDM endpoints are avoided. Therefore, TDM traffic must have precedence over BE traffic and reads from the NI must have precedence over writes to it[14]. The priority of the different accesses to the NI is as follows:

$$TDM_{read} > TDM_{write} > BE_{read} > BE_{write} \tag{4.1}$$

This has the consequence that incoming traffic can starve outgoing traffic. Hence, care must be taken when setting up a test in order to ensure that this does not happen continuously (i.e., it must be ensured that the sum of the incoming and generated traffic in one tile does not exceed the capacity of the wishbone bus).

Arbitration must also be implemented for write requests to the NI from different TDM generator blocks as well as read requests from different TDM endpoints of the NI. For both cases round-robin arbitration is used. This ensures that all incoming or outgoing TDM connections have the same priority and cannot block each other indefinitely.

Consumption Block

The consumption block evaluates the received packets and messages in order to extract the information of interest that is then recorded by the statistics collector module. For both TDM and packet switched traffic the latency is determined. This is done by adding the two's complement of the received timestamp to the current timestamp. The timestamp is a 32 bit value which means that latencies of up to $2^{32} - 1$ cycles are correctly determined. This is sufficient since *a)* longer tests are never run in the scope of this thesis and *b)* even if longer tests would be run, latencies of this magnitude would be extremely unlikely due to the finite buffer in the BE generator block.

The other extracted information is the endpoint on which a message has been received for TDM traffic and the source of a packet for the packet switched traffic.

[14] In the evaluation presented in Section 5 each tile will only ever generate and receive either TDM or BE traffic. However, the TGM is designed in a way that would allow both traffic types concurrently.

Statistics Collector Module

The statistics collector module records the information gathered during a test run. The following data is recorded:

- Number of generated TDM messages and BE packets
- Number of overruns that occurred in each generator block
- Number of received TDM messages and BE packets
- Accumulation of all latencies for both the received TDM messages and BE packets
- Lowest and highest TDM latency and endpoint the respective message has been received on
- Lowest and highest BE latency and source tile the respective packet has been sent by

Recording this data, naturally, requires a lot of register space. To save some register space, the latencies of the received packets and messages are simply accumulated. The average latency must then be determined on the host-PC in a post-processing step.

The statistics collector module has a special *start recording* input to trigger the collection of data. This is used to define a warm-up period in which the generator blocks are enabled but no data is being recorded. Since the amount of data collectively recorded by all TGMs is considerable—especially in the implemented 8x8 system—sending data to the host-PC during a run would exceed the capabilities of the debug sub-system (cf. Section 4.1.1). Hence, the recorded data is sent out only once after a test run is finished.

4.2.2.2 Test Setup Module

The Test Setup Module is the second additional module used for the evaluation system. It is implemented as OSD debug module and globally controls when a test run is started, when recording of the statistics is started, and when a test is stopped. This is done by controlling two signals that are connected to all TGMs. One simultaneously enables the traffic generation in all FGs, the other enables the recording in the statistics modules. To do so, the module uses a 32 bit wide counter and two 32 bit wide registers to store both the cycle in which recording starts and the cycle in which the test run ends. When the test ends both signals are de-asserted which triggers the transmission of the recorded data to the host-PC.

In addition to controlling the tests, the module also houses the central destination table that is used by the BE FGs.

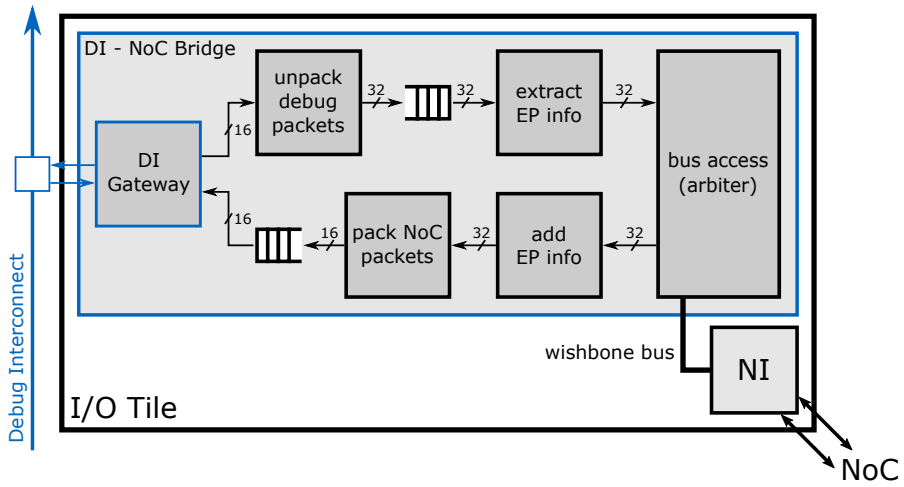


Figure 4.8: I/O tile overview

4.2.2.3 I/O Tile

OpTiMSoC does currently not provide a way to communicate with the software running on the processing tiles via the NoC. The software is loaded to the tile's local memory via the debug sub-system and then runs to completion, or indefinitely. The programs running on different tiles can interact with each other via the NoC but they can only be *observed* by the host-PC. To enable communication between the host-PC and the applications of the demonstrator system over the NoC a special I/O tile is used.

The I/O tile contains an OSD debug module which uses the debug sub-system for communication with the host-PC. This debug module is, in essence, a bridge module between the DI and the NoC (more specifically, the NI). From the NoC's point of view the module is an I/O tile that directly forwards the messages sent to it to the host-PC (and from where messages sent by the host-PC are injected into the NoC). From the host-PC's point of view it is a debug module that is accessed via the debug sub-system and is used to send TDM and BE messages to the tiles via the NoC (and receive them). Figure 4.8 shows the architecture of the I/O tile and its sub-modules.

As described in Section 4.1.1, the debug sub-system has its limitations. Most importantly, the DI is only 16 bit wide whereas the NoC and the wishbone interface to the NI are 32 bit wide. Moreover, the debug packets sent via the DI cannot be more than 12 flits long and three of these flits are used for the debug packet information. This means that *a)* each NoC flit must be packed into two debug flits and *b)* NoC messages or packets that do not fit in a single debug packet must be split into multiple debug packets. Both these tasks are implemented in separate sub-modules, one unpacking the messages and packets received via the DI, and one packing the messages and packets received from the NoC.

A special configuration flit is added to each debug packet that is sent to or from the I/O tile. This flit holds the information about the traffic type of the packet or message (TDM or packet switched) and which endpoint of the NI the packet or message should

be written to or has been received by. Additionally, this flit is also used for a special configuration message to enable and disable the individual endpoints of the NI in the I/O tile. In both directions—i.e., writing to and reading from the NI—this flit is extracted and added in separate sub-modules.

The arbiter works in a similar way as the arbiter in the TGM with TDM being prioritized over BE and reads being prioritized over writes.

Using the debug sub-system for the I/O tile and the communication between the host-PC and the software running on the system has two notable downsides (other than the necessary conversions). The DI is not designed for huge traffic loads and is easily congested if high traffic loads are sent continuously. This not only affects the maximum bandwidth available for the communication via the I/O tile but can also affect the communication with other debug modules. However, the bandwidth is sufficient for the demonstrator system described in Section 4.3.2.

The other downside is that by implementing the I/O tile with a debug module the strict isolation of critical and non-critical traffic is violated. This is caused by the fact that both traffic types use the same off-chip interface and both are sent over the DI. Even though TDM traffic is prioritized when writing to and reading from the NI, head-of-line blocking is a problem (i.e., TDM messages waiting to be written to the NI are blocked by BE packets that are ahead of them and cannot be served). This can happen if backpressure from the packet switched part of the NoC stalls traffic all the way to the receiving buffers of the I/O tile or even into the DI. Similarly, TDM traffic can be blocked by traffic from other debug modules. How this issue is handled in the demonstrator system is described in Section 4.3.2. In a deployed system, though, it would be necessary to ensure strict isolation of the two traffic types for the off-chip interface as well as any other involved paths and modules.

4.2.2.4 Fault Injection Module

The second additional module for the demonstrator system is the FIM. For testing and demonstration, it is necessary to reliably force faults on individual links on demand. The main purpose of the FIM—other than testing the protection switching—is to demonstrate how a faulty link affects the packet switched BE traffic, i.e., by causing misrouting or even a deadlock. However, when forcing a stuck at fault on a random bit of a link, or even flipping a random bit, the fault is masked surprisingly often. Therefore, the FIM flips a different bit in every cycle. This, of course, is not a fault that would typically occur in any system but the approach works well for the purpose of the demonstrator.

The FIM is a purely combinatorial module in order to not change the behavior the NoC. A simple shift register builds the core of the module. A single bit is shifted through the register and wrapped around from the MSB to the LSB. The bits from the shift register are gated with an *enable* signal (logical AND) and then XORed with the bits from the link in order to flip one bit. The FIM is only used in the interactive demonstrator system, where it is instantiated on every link.

4.2.2.5 Traffic Control Module

The last additional module used for the demonstrator system is a traffic control module that is instantiated in each tile of the system. The module is implemented as OSD debug module and has two functions. First, it passively observes the traffic that is written to and read from the NI by monitoring the bus activity. If a message or packet is written to or read from the NI the surveillance module records this activity and stores the destination or source for BE packets or the endpoint for TDM messages. Furthermore, the module also performs a simple fault detection to determine the number of faulty incoming BE packets. To enable this fault detection only the lower 16 bits of a packet are used for the payload and this payload is mirrored to the upper 16 bits (meaning fault detection is enabled through data redundancy.). Since faults are injected by the FIMs by flipping a single bit, faults can be detected by using a bit-wise XOR operation on the lower and upper 16 bits. It must be noted that this is purely for demonstration purposes. The recorded data is sent to the host-PC periodically (every 250ms). Tracing the number of sent, received, and faulty packets in hardware has the advantage that it is unintrusive, meaning the measurement does not affect the traffic generation or consumption.

The second function of the traffic control module is to control the traffic generation of the BE tiles in the demonstrator system. For this purpose, the module provides several configuration registers that are mapped into the address space of the CPU and are accessible via the local wishbone bus. These registers control the traffic generation by determining how often traffic bursts should be generated and how long they should be. For both parameters a range can be defined which is used to generate a random value for the parameter. A seed register is provided defining the seed for the software RNG. Furthermore, a register is provided with a bitmap that defines all valid destinations for the generated BE packets.

These registers can directly be configured via the debug sub-system, thereby bypassing the NoC. This approach has the advantages that the BE applications can still be controlled when they cannot be reached via the NoC due to congestion or faults. This also avoids possible head-of-line blocking in the I/O tile caused by blocked configuration packets sent to the BE applications (cf. Section 4.2.2.3).

When a new value is written to one of the memory mapped registers an IRQ is raised informing the application running on the CPU. The application can then read the updated values via the bus. The architecture of the traffic control module is shown in Figure 4.9.

4.3 System Architecture

Using OpTiMSoC and the additional modules described in the previous Section 4.2, two different manycore systems were designed and implemented in the scope of this thesis: an evaluation system used for the SSE of different system parameters and mapping strategies, and a demonstrator system used to showcase the capabilities of protection switching in NoC with an interactive example. The following sections describe these two systems in more detail.

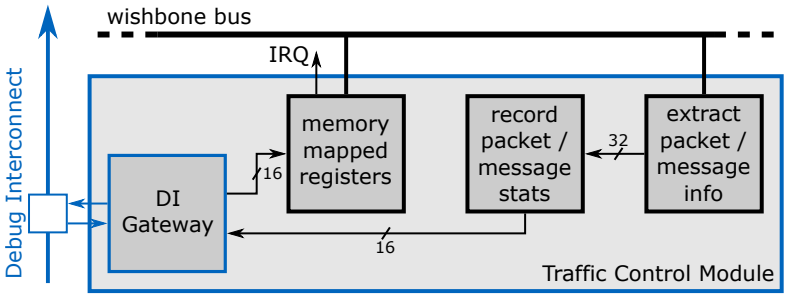


Figure 4.9: Traffic control module overview

4.3.1 Evaluation System

Just as for the simulated system described in Section 3.4.1 a size of 8x8 tiles is used for the evaluation system. The system is composed of 64 generator tiles connected by the hybrid NoC, the test setup module and the network control module (cf. sections 4.2.1.4, 4.2.2.1, 4.2.2.2). The generator tiles, the NoC, and the debug sub-system are all in the same clock domain and are clocked at 100MHz. Figure 4.10 shows the architecture of the evaluation system.

In order run tests on the evaluation system a software implemented in Python is used on the host-PC. This *test runner* reads a special configuration file defining the tests that are to be executed (i.e., application scenario and mappings, cf. sections 5.1.2 and 5.1.3). The file also defines for how many cycles each test is run, how long the warm-up period is, and how often each test is executed with different seeds. The test runner then connects to the evaluation system implemented on the FPGA and reads the slot table size of the system. Next, all mappings—generated by the mapping algorithms described in Section 3.5.3—that match the defined tests and the slot table size are gathered and the tests are executed. For each individual test run the test runner does the following:

- Reset the system
- Set up all TDM paths and enable the respective endpoints and links
- Configure all generator tiles and set different seeds for all RNGs
- Configure the central BE destination table in random order
- Start the test by configuring warm-up period and test duration
- Wait for recorded test data from generator tiles and NCM
- Do necessary post-processing
- Write recorded data to a file

The test runner is only responsible for setting up a test and storing the data gathered during a test. During the test execution there is no interaction between the test runner

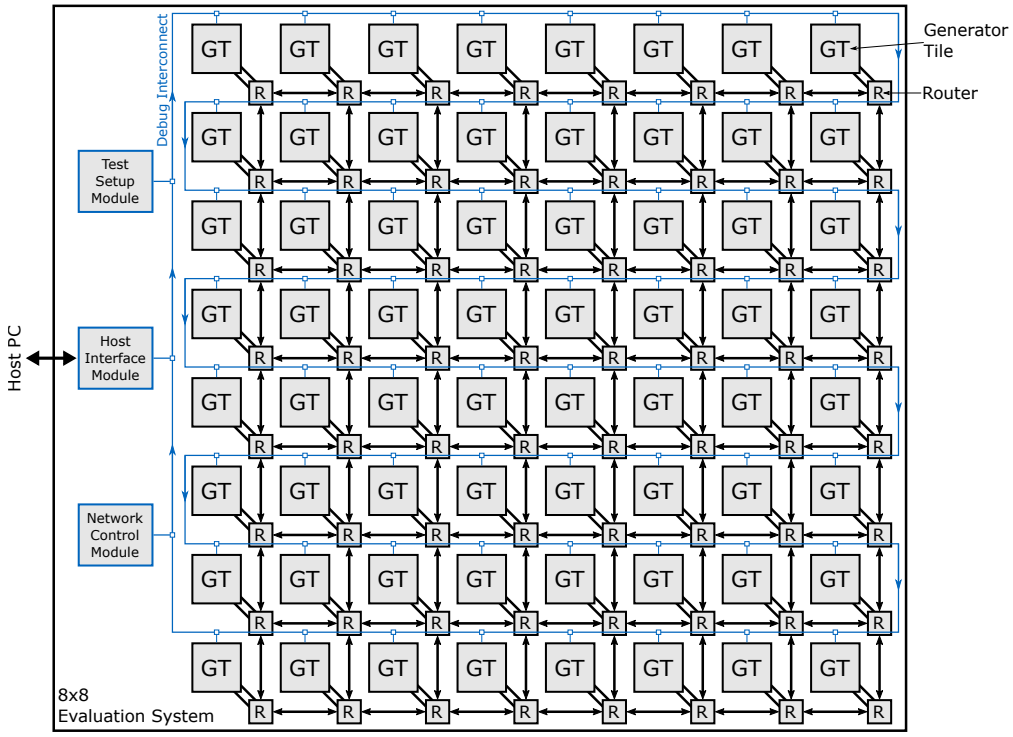


Figure 4.10: 8x8 Evaluation System Architecture

and the evaluation system. This is important in order to avoid the bottleneck of the debug sub-system and the off-chip interface during the test which would affect the test execution.

4.3.2 Demonstrator System

The purpose of the demonstrator system is to showcase the capabilities of protection switching in NoC with an example use case in an interactive system. Specifically, the demonstrator is designed to show three aspects of the concept: fault-tolerance, traffic isolation, and (hard) real-time capabilities. To do so it is necessary to have both critical and non-critical applications in the system. Furthermore, the system must be sufficiently large to allow path diversity for protection switching.

The demonstrator system is composed of 15 processing tiles and an I/O tile which are all connected by a hybrid 4x4 NoC mesh. Each processing tile contains an OpenRISC core, a local memory, and a NI. Furthermore, four OSD debug modules are instantiated in each processing tile to trace the software and the core registers, load the software into the local memory, and to trace the sent/received packets/messages and control the traffic generation. The architecture of the processing tile and its sub-modules are shown in Figure 4.11 OpTiMSoC allows to either use BRAM for the local memory or map the memory to external DDR memory in case the available BRAM is insufficient. In the demonstrator system, BRAM is used for the critical processing tiles—for better

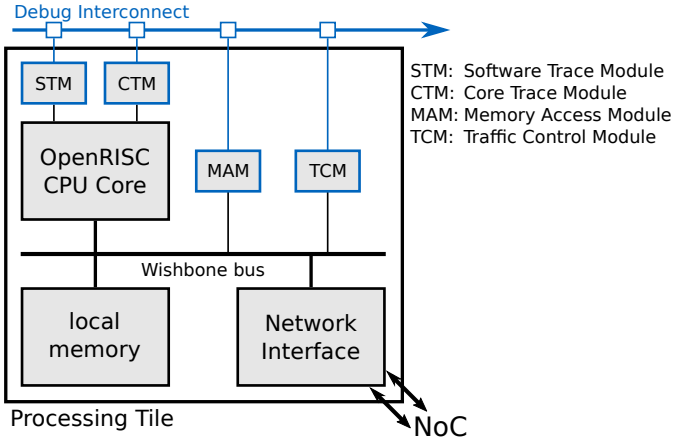


Figure 4.11: Processing tile of the demonstrator system

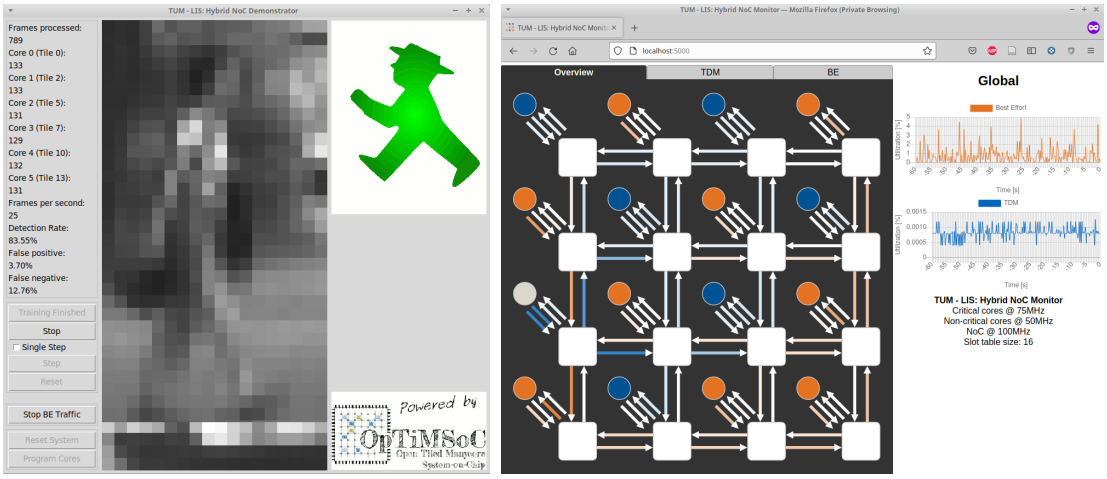
performance—while external memory is used for the BE tiles since not enough BRAM is available for all processing tiles. For the synthesis results in Section 4.4, however, all local memories are mapped to external DDR memory for better comparability.

A pedestrian detection algorithm (based on a simple support vector machine) is used to represent a critical application. Six of the processing tiles are defined to be critical tiles which each run the pedestrian detection algorithm. The host-PC sends individual frames to the critical tiles via the I/O tile and the NoC using TDM and protection switching. The algorithm then executes the pedestrian detection and sends the calculated result (i.e., pedestrian or not) back to the host-PC which displays the result and sends a new frame to the tile it got an answer from. The frames used for the demonstrator are taken from the Daimler Mono Pedestrian Classification Benchmark Dataset [161, 162]

The nine non-critical processing tiles produce and consume packet switched BE traffic as background traffic. By default, uniform random traffic is generated but each tile can also be configured to only send traffic to a sub-set of all other BE tiles. The software running on both the critical and non-critical tiles is implemented bare metal in C, i.e., without any underlying operating system.

A *demonstrator runner* is implemented in Python on the host-PC. The demonstrator runner connects to the demonstrator system—more precisely, to the debug sub-system—and displays a GUI that allows to configure the demonstrator system and control the pedestrian detection. Furthermore, a local server is started that can be accessed with a browser in order to display an interactive monitoring GUI. This GUI can be used to inspect the NoC—including individual links—force the injection of faults in the NoC, re-configure TDM paths, and configure the BE traffic generation. The two GUIs are shown in figure 4.12a and 4.12b respectively.

As described in Section 4.1.1, the debug sub-system has its limitations regarding the available bandwidth and can quickly become a bottleneck with growing system size and when large amounts of data are sent. In the demonstrator system there are basically two large traffic streams: the frames that are sent to the critical tiles via the I/O tile



(a) Pedestrian detection GUI **(b)** Monitoring GUI

Figure 4.12: Demonstrator GUI

(the answer that is sent back is very short) and the statistics and utilization data sent by the traffic control modules and the NCM to the host-PC. To avoid that these two major traffic streams block each other, the I/O tile is the first module on the debug ring *after* the off-chip gateway (the host interface module) and the NCM is the last module *before* the gateway.

The demonstrator system has three different clock domains to showcase the possible use in a GALS system. The non-critical tiles and the debug sub-system are clocked at 50MHz, the critical tiles are clocked at 75MHz, and the hybrid NoC is clocked at 100MHz. Due to the limitations of the debug sub-system and the performance of the OpenRISC cores, the achievable frames per second per core is limited to around 7. The consequence is that the amount of TDM traffic in the NoC is relatively low. Nevertheless, the demonstrator effectively shows that the critical communication is unaffected by any single fault and that even heavy congestion in the NoC caused by BE traffic has no effect on the critical traffic. Figure 4.13 shows an overview over the demonstrator system and the connected host-PC. A video describing the demonstrator and showing it in action is available online [163].

4.4 Synthesis Results

Both the evaluation and the demonstrator system were synthesized with different parameters to enable the SSE of different system parameters and mapping strategies, and to evaluate how the parameters affect the overall resource requirements. In both cases the data link width of the hybrid NoC is 32 bit (not counting parity bits and status flags, cf., Section 4.2.1.1). The evaluation system was synthesized with a slot table size of 4, 8, 16, 32, and 64 to enable the evaluation of a large number of different mappings. Furthermore, router input buffer sizes for the packet switched traffic of 8, 16, and 32 flits

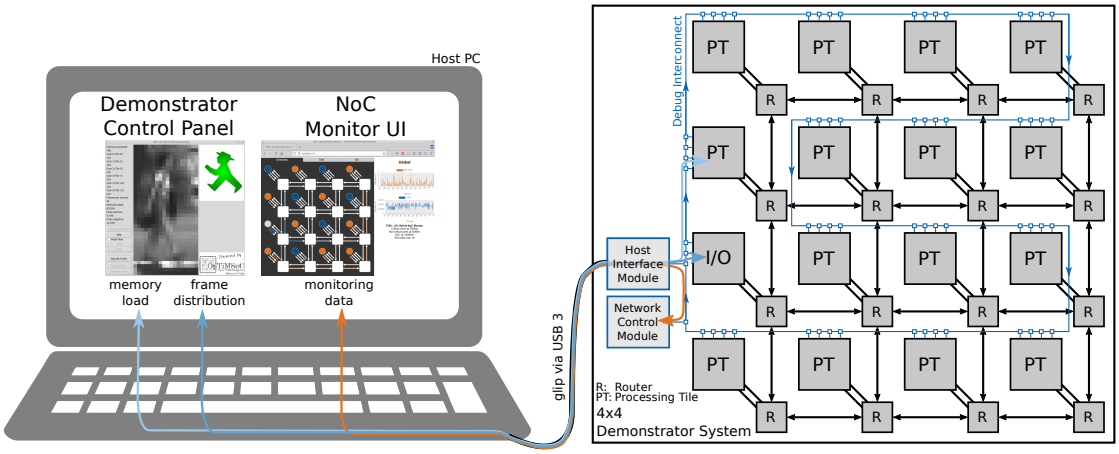


Figure 4.13: 4x4 Demonstrator System with host-PC

were synthesized. The evaluation system uses version 1 of the data path synchronization in the NI which requires two individual *in*-queues (cf. Section 4.2.1.2). The demonstrator system was synthesized with a slot table size of both 16 and 128 and router input buffer size of 8 flits. This system uses version 2 of the data path synchronization in the NI. In order to not distort the results for the hybrid NoC, the demonstrator system was synthesized without FIMs and FDMs in the routers for the comparison in this section. Both systems were synthesized with three TDM endpoints in the NIs with the exception of the NI in the I/O tile of the demonstrator system which has 15 TDM endpoints. An overview over the synthesis results is presented and discussed in the following paragraphs.

All syntheses were done with Xilinx Vivado for a Xilinx Virtex Ultrascale FPGA (XCVU095). One peculiarity of Vivado is that, while optimizing a design, it sometimes moves logic between different (sub-)modules or combines logic from different (sub-)modules in a single Look-Up Table (LUT). This is very convenient, of course, for decreasing the overall resource requirements of a design but makes it difficult to determine the exact size of a particular (sub-)module. Hence, the results presented in this section are a good first indicator but must be taken with a grain of salt. However, to reduce this uncertainty the average resource requirements are given for each module that is instantiated more than once (e.g., for all NoC routers, NIs, etc.). The average is always rounded up to the next integer (with the exception of BRAM where half a block can be used).

Table 4.1 shows an overview over the synthesis results for the demonstrator system with a slot table size of 16. The routers have two local links each in order to enable protection switching. The number of ports that are synthesized to connect a router to its neighbors depends on the position of the router: 2 for a corner router (*R2*), 3 for an edge router (*R3*), and 4 for a central router (*R4*). The table also lists the synthesis results for the BE and TDM endpoints (*EP BE* and *EP TDM*) and for a NI with three TDM endpoints. Furthermore, the table shows the synthesis results for the OpenRISC

Unit	R2	R3	R4	EP BE	EP TDM
#LUTs	819	1394	1960	409	387
#Registers	660	854	1048	400	668
#BRAM	-	-	-	1	1

Unit	NI	OR core	PT	NoC 4x4	Sys 4x4
#LUTs	2189	4517	10 900	22 256	226 737
#Registers	2564	2788	11517	13 661	244 452
#BRAM	4	10.5	14.5	-	293

Table 4.1: Demonstrator system synthesis results for a slot table size of 16

Unit	ST16	ST128	NoC 4x4 (ST128)	Sys 4x4 (ST128)
#LUTs	55	139	29 646	254 844
#Registers	55	394	40 779	287 848
#BRAM	-	-	-	293

Table 4.2: Synthesis results of a single router slot table with size 16 (ST16) and 128 (ST128), and both the 4x4 NoC and 4x4 system with slot table size 128

(OR) core, the processing tile (PT), the 4x4 NoC (not including the NIs), and the entire demonstrator system. It should be noted that the OpenRISC core is fairly small and that certain features such as a data cache are deactivated to further reduce the size. The hybrid NoC accounts for $\sim$ 9.8% of the overall LUT and $\sim$ 5.6% of the overall register requirements of the demonstrator system, not counting the NIs. Including the NIs, the NoC accounts for $\sim$ 27.9% of the overall LUT and $\sim$ 25.7% of the overall register requirements[15]. It should be noted, though, that particularly the design of the NI could be further improved—similar to [155]—in order to utilize the tile local memory for the buffers in the different endpoints. This way, the resource requirements of the NI could be further reduced. This, however, is out of the scope of this thesis. Overall, the demonstrator system uses $\sim$ 42% of the FPGA's LUTs and $\sim$ 23% of the FPGA's registers.

Table 4.2 shows the synthesis results for a single router slot table of the demonstrator system with size 16 and 128 respectively. Furthermore, it shows how a slot table size of 128 affects the overall size of both the 4x4 NoC and the entire demonstrator system. As can be seen, the larger slot table has a considerable effect and changes the contribution the NoC has to the overall resource requirements to $\sim$ 11.6% of the overall LUT and $\sim$ 14.2% of the overall register requirements, not including the NIs. However, even a slot table size of 128 is still affordable. Furthermore, it should be noted that a slot table size of more than 16 should rarely be necessary, as will be discussed in Section 5.2.1.

[15]The NI in the I/O tile has 15 TDM endpoints and is, hence, much larger than the NI listed in Table 4.1. This has been considered in the values given.

Unit	EP TDM	NI	GT	NoC 8x8	Sys 8x8
#LUTs	609	2695	4450	104 196	437 957
#Registers	694	2645	4328	68 079	370 206
#BRAM	2	7	8	-	513

Table 4.3: Evaluation system synthesis results for a slot table size of 16

Lastly, Table 4.3 shows the synthesis results for the evaluation system with a slot table size of 16 and a router input buffer size of 8 flits. Since the path synchronization is implemented differently in the NI (cf. Section 4.2.1.2) the TDM endpoint is slightly larger than in the demonstrator system. The NoC accounts for $\sim$ 23.8% of the overall LUT and $\sim$ 18.4% of the overall register requirements of the evaluation system, not including the NIs ($\sim$ 63.2% and $\sim$ 64.1% respectively including the NIs). This is of little surprise as the traffic generator module is much smaller than an OpenRISC core and the entire system is designed to evaluate the NoC. Overall, the evaluation system uses $\sim$ 81% of the FPGA's LUTs and $\sim$ 34% of the FPGA's registers.

4.4.1 Comparison with State of the Art

A direct comparison of different implementations is often difficult, not least because few publications provide synthesis results and the ones that do typically only consider either the NoC routers or the NIs but not both. In this section, the synthesis results of the hybrid NoC router are compared to two router implementations presented in [99]—which use packet switching with virtual channels to provide GS in a mixed-critical system and can be considered state of the art—for a comparison with a purely packet switched NoC.

A 2x2 NoC composed of four corner routers is used for the comparison. The hybrid 2x2 NoC uses a total of 3276 LUTs and 2640 registers. The smallest router implementation described in [99] has two VCs and uses 5132 LUTs and 3874 registers for a 2x2 NoC. A version with three VCs uses 6346 LUTs and 4875 registers. Hence, the hybrid TDM and packet switched router proposed in this thesis requires $\sim$ 36% fewer LUTs and $\sim$ 31% fewer registers than the purely packet switched router with two VCs and $\sim$ 48% fewer LUTs and $\sim$ 46% fewer registers than the version with three VCs. This is despite the additional local link for the TDM traffic which is required for the protection switching. The reason is that the implementation of the packet switching can be very basic and does not require VCs or QoS since TDM is used for critical traffic. However, the hybrid NoC comes at the expense of lager NIs which must handle the TDM channels and the protection switching.

4.5 Summary

An extensive SSE of different system parameters and mapping strategies for a given traffic scenario requires a prohibitive amount of time with cycle-accurate simulations. Instead, an evaluation system for the exploration was designed and implemented on

an FPGA. Furthermore, an interactive demonstrator system was developed and implemented for the ARAMiS II research project. This demonstrator showcases the capabilities of protection switching in NoC and the traffic isolation between the critical TDM traffic and the packet switched BE traffic.

The basis for the two implemented systems is OpTiMSoC, an open source framework for implementing tiled manycore systems. For the design of both the evaluation and the demonstrator system, the packet switched NoC of OpTiMSoC was replaced with the hybrid NoC described in Section 3.2. Furthermore, several additional modules were developed to enable testing with the evaluation system and interaction with the software running in the demonstrator system.

Synthesis shows that the NoC has reasonable resource requirements. A comparison with a state-of-the-art packet switched router shows that the hybrid NoC router requires $\sim$ 36%–48% fewer LUTs and $\sim$ 31%–46% fewer registers. In the following Chapter 5 the described evaluation system is used for the evaluation of the mapping strategies proposed in Section 3.5.1.

5 FPGA-Based Evaluation

The conclusion of the simulation-based evaluation of protection switching in NoC in Section 3.4 was that the concept works, that the adverse effect protection switching has on the BE traffic is affordable, and that it can even help to improve the overall network utilization. However, the evaluation only considered two different mapping scenarios. More tests with a greater number of mappings are necessary for a comprehensive evaluation. Furthermore, the mapping strategies described in Section 3.5.1—which are intended to maximize the available bandwidth for the BE applications—must be evaluated to determine if they work in the intended way and consistently produce *good* mappings. Both takes a prohibitive amount of time with the cycle-accurate simulation-based approach that has been used for the proof of concept.

In this chapter, an in-depth evaluation of protection switching in NoC and the mapping strategies described in Section 3.5.1 is presented. The evaluation is based on tests performed with the evaluation system described in Section 4.3.1 which is implemented on the "Xilinx Virtex UltraScale FPGA VCU108 Evaluation Kit" which contains a Xilinx Virtex Ultrascale FPGA (XCVU095). Goal of the FPGA-based evaluation is:

- to verify or falsify the preliminary results from the simulation-based evaluation presented in Section 3.4,
- to evaluate the different protection switching versions and compare them to each other based on a larger number of different mapping scenarios, and
- to evaluate the effectiveness of the mapping strategies proposed in Section 3.5.1 to consistently produce *good* mappings.

Evaluation results similar to the ones presented in this chapter have previously been published in [16, 17]. However, additional tests with different parameters have been added in this thesis and all test cases described in these publications have been re-run for consistency. This, naturally, causes slight differences in the results. The new and additional results, however, show the same trend as the ones presented in [16, 17] and, hence, further support the conclusions drawn in these publications.

5.1 Experimental Setup

Before the evaluation results are presented and discussed, this section describes the experimental setup, the system parameters of the evaluation system, and the application scenarios—or mapping scenarios—that are used.

5.1.1 System Definition

The evaluation system—both the hardware part implemented on the FPGA and the *test runner* implemented in software on the host-PC—is described in Section 4.3.1. The hardware part consists of 64 generator tiles that are connected by an 8x8 NoC. Similar to the proof of concept simulations in Section 3.4 different task graphs are mapped to the system to represent critical applications. Tiles that are part of a critical application produce and/or consume TDM traffic according to their task graph. The remaining tiles are BE tiles and produce/consume uniform random packet switched BE traffic (uniform random between the BE tiles of the mapping).

The BE FGs of the generator tiles are configured to generate traffic in bursts of 0–15 packets. Each packet has a fixed size of 15 flits. Different traffic generation rates are achieved by configuring different random intervals between bursts in the RNGs of the FGs (e.g., 307–818 cycles for a generation rate of 20%).

The TDM FGs are configured to produce traffic with a steady rate. This, of course, is typically only the case for streaming-like applications. However, hard real-time applications often have rigid intervals between messages. Furthermore, even if traffic is generated in a more bursty manner it is drained with a steady rate according to the amount of slots that are reserved for the connection. It is up to the developer to ensure that the reserved bandwidth—i.e., the number of reserved slots—is at all times sufficient for the application.

To evaluate both the effect of protection switching on the BE traffic in general and the effectiveness of the mapping strategies under different circumstances, the system was implemented in four different basic versions. The first version uses generator modules operating in batch mode, meaning that the destination is determined individually for each packet of a generated BE burst (cf. 4.2.2.1). This version is representative for systems in which the BE applications generate a lot of multicasts or broadcasts, e.g., caused by cache coherence protocols. The input buffers of the NoC routers have a size of 8 flits.

The second, third, and fourth version all use generator modules in burst mode, meaning that the destination is determined once for all packets of a burst. These versions are representative for systems in which the BE applications primarily generate singlecasts. Naturally, the average packet latency will be higher for these systems and the saturation rate will typically be lower than for the first version. To assess how much—if at all—a larger input buffer in the NoC routers can alleviate this effect, and how this affects the mapping strategies, three systems were implemented with an input buffer size of 8, 16, and 32 flits respectively. All four basic systems are listed in Table 5.1.

Each of these four systems was synthesized with five different slot table sizes—4, 8, 16, 32, and 64—to assess how the slot table size affects the quality of the generated mappings[1]. This results in tests being run on 20 different hardware configurations (combinations of slot table size and basic evaluation system version). Mappings for each of the five slot table sizes were created for different applications scenarios (cf. the follow-

[1] In general, small slot tables are favorable regarding the overall size and power consumption of the NoC but limit the amount of possible mappings since fewer TDM channels can share a single link.

Evaluation system	BE generator	NoC router buffer size
Version 1	batch mode	8
Version 2	burst mode	8
Version 3	burst mode	16
Version 4	burst mode	32

Table 5.1: Evaluation system - basic versions

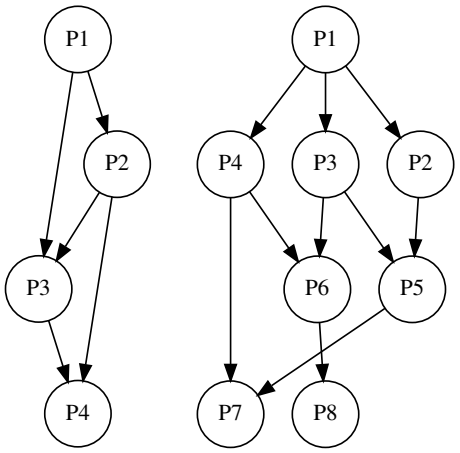


Figure 5.1: Task graphs A (left) and B

ing Section 5.1.2) to evaluate how well the different mapping strategies perform under different constraints. Each mapping was then tested on each of the four systems.

5.1.2 Application Scenarios

An important aspect of the evaluation are the application scenarios that are assumed. Real-life application scenarios and their traffic patterns vary greatly and it is impossible to consider every possible scenario. Nevertheless, the evaluation of the mapping strategies and protection switching in general should include a certain range of different traffic scenarios. To achieve this, two different task graphs are defined to represent critical applications: a smaller one with 4 nodes and 5 edges (graph A) and a larger one with exactly twice the amount of nodes and edges (graph B). Both graphs are shown in Figure 5.1.

For the evaluation it is assumed that no more than half of the processing tiles in a mixed-critical system will run safety-critical applications in order to leave enough spare resources for both BE applications but also resources to potentially migrate to in case of a fault in one of the processing tiles (the latter, however, is out of the scope of this thesis but will be briefly discussed as future work in Section 6.1). Using the mapping algorithm described in Section 3.5.3 graph A was mapped 2, 4, 6, and 8 times to the 8x8

system and graph B was mapped 1, 2, 3, and 4 times. This allows to evaluate how well the mapping strategies perform with different amounts of critical tiles—specifically with 8, 16, 24, and 32 critical tiles—and whether the size of the graph has a significant impact on the quality of the mapping (i.e., the achieved saturation rate). Furthermore, for each of these combinations of task graphs, mappings with different bandwidth requirements were created: for a TDM traffic generation rate of 5%, 10%, 15%, 20%, and 25% per sending node. This results in a total of 40 different example application scenarios. The generation rate is to be understood as the net generation rate of each sending node of a task graph, meaning, e.g., with a generation rate of 10% P1 to P3 of graph A generate 10% traffic each, evenly divided between the outgoing edges. Note that the flit injection rate of the critical tiles is twice their generation rate when 1+1 protection is used.

5.1.3 Mappings

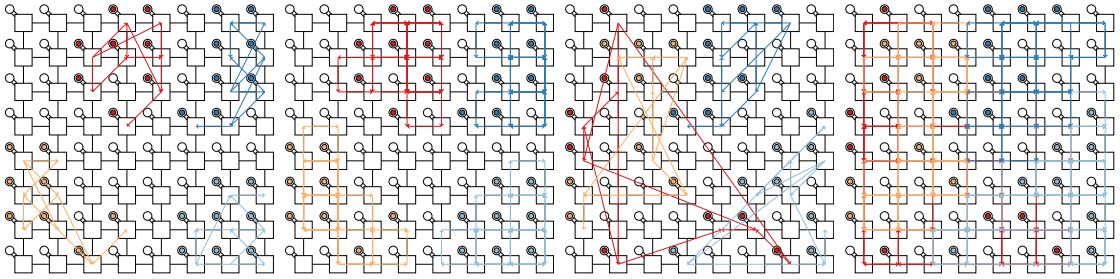
For each of the 40 example application scenarios mappings were created for the five different slot table sizes: 4, 8, 16, 32, and 64. The mappings were created with two active paths for 1+1 protection and one active path for 1:1/1:n protection since they both have the same effect on the BE traffic. This resulted in a total of 400 different example application scenario, slot table size, and protection switching combinations as the basis for comparison.

With the algorithm described in Section 3.5.3 mappings were generated for all of the 400 combinations with a multi-objective optimization of the optimization objectives O1 to O4 described in Section 3.5.1. All optimizations were run 5 times independently for 1500 iterations and then combined to a single 4-dimensional Pareto front. The seven different mapping strategies listed in Table 3.2, each following a given optimization direction, were used to select mappings from this Pareto front. Strategies S1 to S4 respectively follow the direction of the single objectives O1 to O4. Strategies S5 to S7 follow the direction defined by the normalized weighted sum of multiple objectives: O1 & O2 for S5, O2 & O4 for S6, and O1, O2, & O4 for S7. The used weights are assumed to be equal.

For each strategy five samples were selected following the optimization direction: the best one (sample 1), the worst one (sample 5), and linearly three samples in between. This allows to determine if there is a correlation between a given mapping strategy and the effect on the BE traffic performance.

In addition to the mappings generated with the strategies S1 to S7, the saturation rates for systems with 8x7, 8x6, 8x5, and 8x4 BE tiles was evaluated. This allows a comparison of the mapping strategies to the common approach of virtually splitting the system into a critical and a non-critical application domain (hereafter called ‘split region’ systems). Furthermore, the saturation rate of a system with 8x8 BE tiles was evaluated for a baseline comparison.

For each of the mappings generated by the strategies S1 to S7 tests were run both without TDM traffic—to determine the baseline saturation rate of that mapping—and with TDM traffic. Furthermore, an additional test was run in which, in addition to the already reserved slots, as many additional slots as possible are reserved for each TDM



(a) S1 - connections **(b)** S1 - used paths **(c)** S7 - connections **(d)** S7 - used paths

Figure 5.2: Different mappings of four times graph B with a slot table size of 16, a TDM generation rate of 25%, and 1+1 protection

channel (naturally, how many additional slots can be reserved, if any, varies between mappings and channels). The intention of this *extended* TDM "mapping"[2], hereafter referred to as "TDM$^+$ traffic" is to determine if it makes sense to reserve as many slots as possible to a critical connection in order to reduce the time in which the packet switched BE traffic is blocked, or if this has a worse effect on the BE traffic.

Two examples of different mappings generated by using different strategies are shown in Figure 5.2. Figures 5.2a and 5.2b respectively show the logical connections and the used paths for a mapping generated by using strategy S1, and figures 5.2c and 5.2d show the same for a mapping generated by using strategy S7. Both mappings map four instances of graph B to a NoC with a slot table size of 16 and for a TDM traffic generation rate of 25% and 1+1 protection. As can be seen, the generated mappings differ greatly both in how the graphs are clustered or spread out and in the amount of links that are used.

For each mapping, as well as the split region systems and the 8x8 reference runs, the saturation rate was determined by running tests with increasing BE traffic generation rate. Each configuration was run 10 times with different random seeds in the FGs and the average value is used for the evaluation. Each test was run for 10 000 000 cycles after a warm-up period of 100 000 cycles. Longer runs did not change the results significantly. BE traffic generation rates between 1% and 35% were tested in 1% steps. However, a larger step size was used for lower generation rates that did not cause the NoC to saturate and the 1% step size was used around the saturation rate (with the exception of the split region and the reference tests for which all generation rates were tested). This way, on average only 8–9 different BE generation rates had to be tested for each mapping—instead of 35—which significantly reduced the number of test runs. Overall, a total of 13 737 910 individual tests were run. Due to the FPGA implementation, an average of ~ 4.5 tests could be run per second.

An important question is *when* the NoC can be considered to be in saturation. One possibility would be to define an average latency for the packet switched traffic as indicator. However, this makes it difficult to compare the different basic systems listed in

[2]Technically it's not a different mapping, just the same mapping but with additional resources reserved.

Table 5.1 since the systems with the BE FGs operating in burst mode and larger NoC router input buffers naturally have a higher average latency. Hence, for the evaluation presented in this thesis the NoC is considered to be saturated when the traffic injection rate cannot keep up with the traffic generation rate in the FGs. This is indicated by buffer overruns in the generators (cf. Section 4.2.2.1).

For the evaluation in this chapter the threshold after which the NoC is considered to be in saturation is when, on average, each BE tile had an overrun of one packet. The FIFO buffer of the BE FGs can hold 64 packets and the burst queue holds 8 bursts of length 1–15, meaning it can be considered an additional buffer with a size of 8–225 packets. Furthermore, the output buffer of the BE endpoint in the NI can hold 17 packets, meaning that 89–306 BE packets are buffered in a generator tile before a buffer overrun would occur. This, however, is a relatively low amount of packets compared to the total number of packets generated during a run. For example, at a traffic generation rate of 25% each BE tile generates, on average, $10\,100\,000/4 = 2\,525\,000$ flits during a run which results in $168\,333.\overline{3}$ packets. This means that, at this rate, the NoC would be considered to be saturated when the traffic injection rate is $\sim 0.01\%$–0.05% lower than the generation rate.

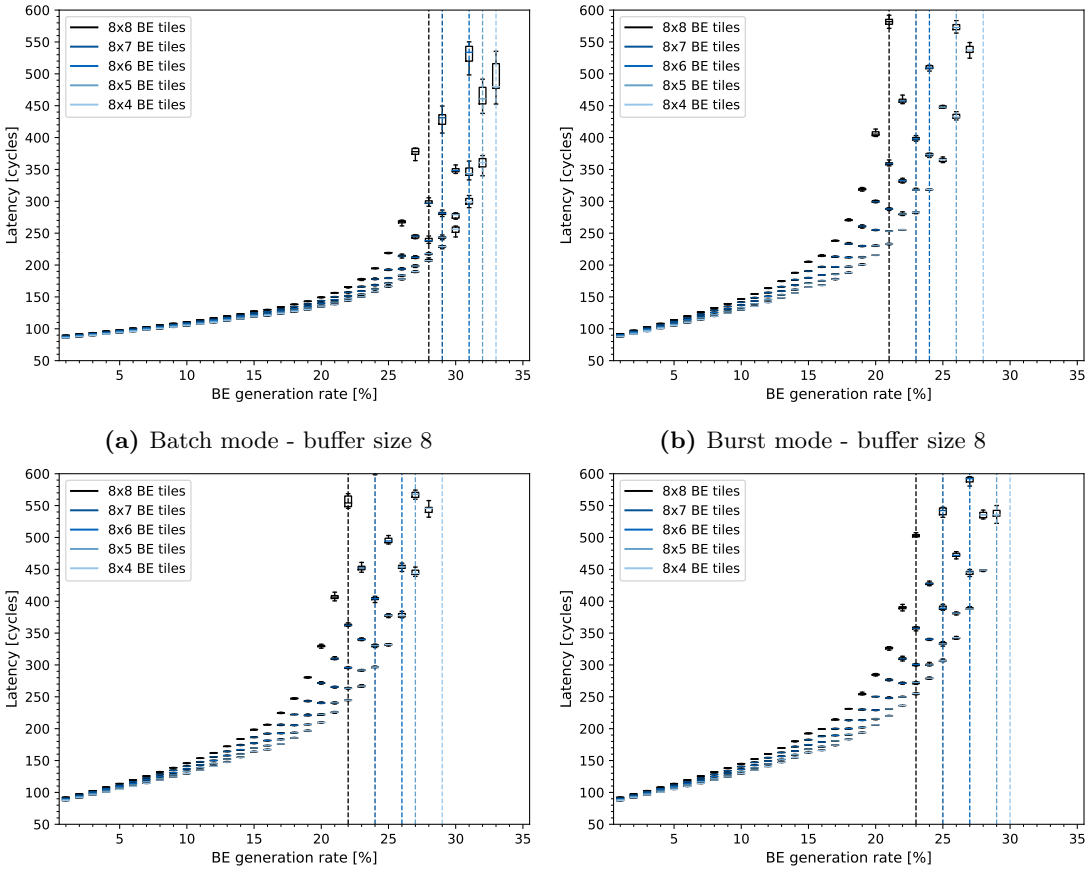
5.2 Mapping Strategy Evaluation

Figure 5.3 shows the measured latencies—ten runs for each generation rate, each with different random seeds—of both the reference system with 64 tiles generating BE traffic and the different split region systems for all four basic versions of the evaluation system. The dotted vertical lines depict the saturation rate, at which the average number of buffer overruns in the generator tiles exceeds 1 per BE tile. The results show that the spread of the latencies for different random seeds is mostly fairly low and only gets larger towards the saturation rate. The results also show that the system with the FGs operating in batch mode has the highest saturation rates which is of little surprise since the BE is, naturally, spread out more. The three systems operating in burst mode all achieve lower saturation rates but the rates consistently increase with growing NoC router buffer size. In all four systems, the reference runs with 8x8 BE tiles achieve the lowest saturation rate whereas the smallest split region—8x4 BE tiles—achieves the highest saturation rate. The reason is that in the smaller systems packets have, on average, a shorter path to travel and, therefore, less interference with other packets.

In the following sections 5.2.1 and 5.2.2 the mapping strategies will first be evaluated for both 1+1 protection on one hand and 1:1/1:n protection on the other hand. Afterwards, and based on the evaluation results, the protection switching versions are compared in Section 5.3.

5.2.1 Mapping Strategies for 1+1 Protection

As a first step, a fidelity check of the mapping strategies is done by comparing the results of the samples of each strategy individually. For each parameter combination of task graph (A or B), number of graphs, TDM traffic generation rate, and slot table size,



(a) Batch mode - buffer size 8

(b) Burst mode - buffer size 8

(c) Burst mode - buffer size 16

(d) Burst mode - buffer size 32

Figure 5.3: Latencies of the 8x8 reference and split region systems for the four basic system versions

the five samples of a strategy are ranked. The higher the saturation rate of a sample and the lower the average latency at that rate—in case two samples have the same saturation rate—the better. Figure 5.4 shows a histogram for each sample number, showing the number of times a sample got ranked 1 to 5 (best to worst). In case of a correlation between a mapping strategy and the BE performance, the histogram for sample n should have a peak at rank n. The histograms shows the results from all four basic system versions as they all show a similar trend[3]. The same is done for the remaining figures in this chapter, unless stated otherwise. The individual histograms can be seen in figures A.2 and A.3 in the Appendix.

When looking at the single-objective mapping strategies S1 to S4 the histograms suggest a correlation for S2 and S4 but no correlation for S1 and S3. In case of S1 the middle sample 2–4 seem to be ranked higher than the supposedly best and worst

[3]The differences between the mapping strategies tend to be larger when using batch mode, but the trend is the same across all basic system versions.

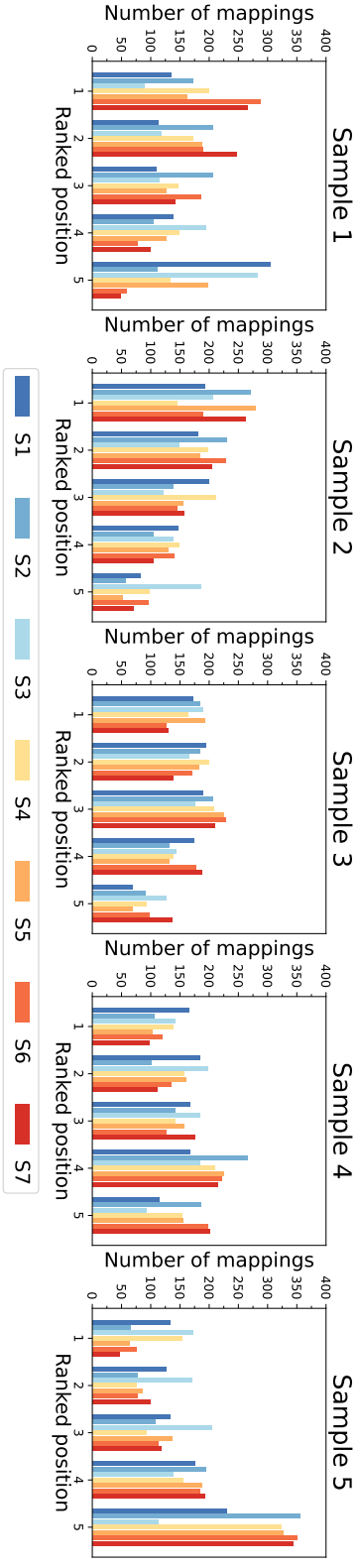


Figure 5.4: Ranking histogram for 1+1 mapping strategy samples - all basic system versions combined

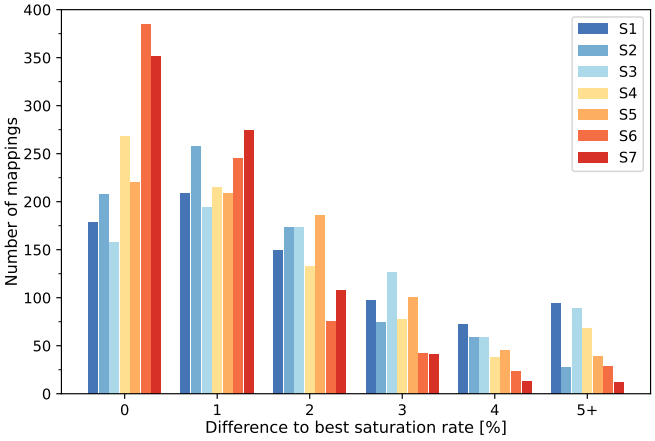


Figure 5.5: Overall comparison of 1+1 mapping strategies

samples 1 and 5. This suggests that the extremes—both minimizing or maximizing the number of reserved slots—have a negative effect on the BE traffic. S3, on the other hand, does not seem to be very consistent at all. The three multi-objective strategies were defined based on the results of the single-objective mapping strategies. Since S3 does not show a correlation and S2 arguably shows the best correlation out of the four, all multi-objective optimizations that include objective O2 and exclude objective O3 are defined as mapping strategies. As can be seen, in contrast to the single-objective strategies, the multi-objective strategies S5 to S7 all show a strong correlation between the sample number and the BE traffic performance, particularly S6 and S7.

For the remainder of this section only sample 1—the allegedly best mapping of each strategy—is used to directly compare the strategies to each other. For this comparison, only the saturation rate of each mapping is compared, without additionally considering the average latency at the saturation rate. Figure 5.5 shows a histogram of the difference in percent to the highest saturation rate that is achieved for each parameter combination and for all strategies (the individual histograms can be seen in Figure A.1 in the Appendix).

Several observations can be made from this histogram. First, it shows that, overall, the majority of the mappings are relatively close to the best mapping. Second, there is no single best strategy that guarantees to always produce the best or at least a near optimal mapping. This could be interpreted as an indicator that the proposed mapping strategies only have a minor influence on the achievable saturation rate. However, the results clearly show that both S6 and S7 in particular have a high tendency to be at least close to the best saturation rate and only a low tendency to have a difference to the highest saturation rate of more than 3%. S2, S4, and S5 all still show fairly decent results and only S1 and S3 have a high tendency to be among the worst mappings. This is consistent with the observations made in Figure 5.4.

As a next step, a more fine-grained comparison of the mapping strategies is done with subsets of the overall results in order to determine for which traffic scenarios the

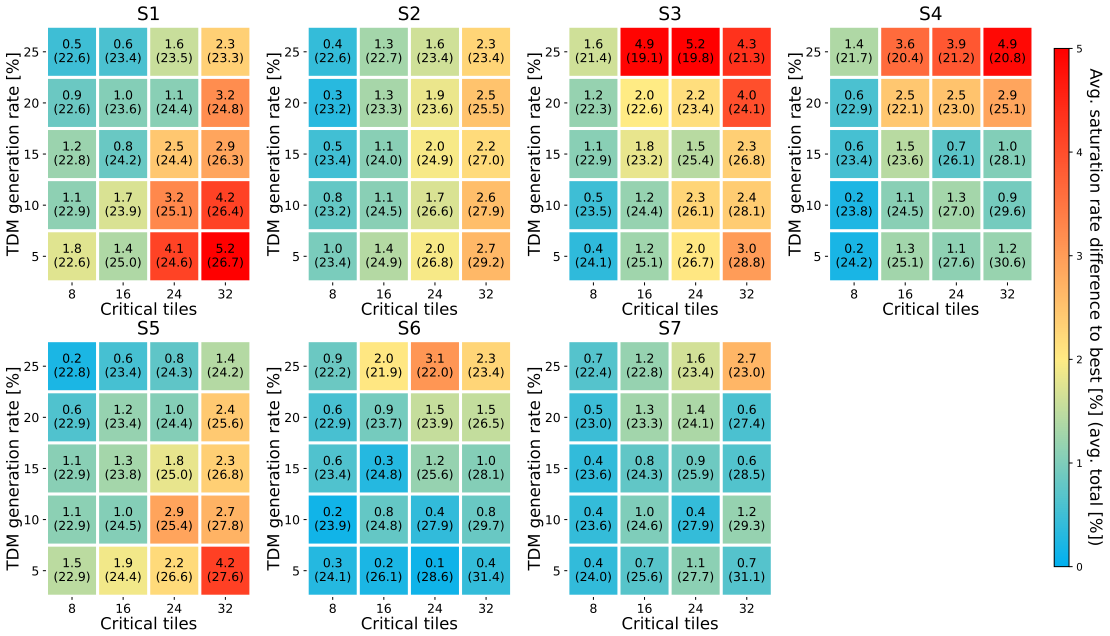


Figure 5.6: Detailed comparison of 1+1 mapping strategies

mapping strategies work best. This is done by grouping the mappings according to the number of critical tiles and the TDM generation rate. Figure 5.6 shows a matrix for each of the mapping strategies S1 to S7. The first value in each cell (subset) shows the average difference between the saturation rate of a mapping created with a given strategy and the highest saturation rate of all strategies. A strategy that always achieves the highest saturation rate (for a subset) will have an average difference of 0.0. This value allows a relative comparison of the strategies for a certain subset of the mappings (i.e., a certain number of critical tiles and amount of TDM traffic). The second value (in brackets) shows the average absolute saturation rate of the mappings generated with each strategy for a parameter subset. This value allows a comparison of the mapping strategies and the reference and split region runs shown in Figure 5.3. Furthermore, this value allows to determine the overall maintainable traffic generation rate, i.e., BE traffic and TDM traffic combined. As can be seen, the mapping strategies show large differences in their performance across the parameter space.

Figure 5.6 shows that, between the single-objective strategies S1 to S4, S4 performs the best for TDM traffic generation rates up to 15%. For these rates, S4 shows little sensitivity to the number of critical tiles. S2 shows good results for 8 and 16 critical tiles and, in comparison to the other strategies, especially for high amounts of TDM traffic. S1 actually produces relatively good mappings for high amounts of TDM traffic. This is in contrast to the overall comparison of the strategies in Figure 5.5 where S1 performed relatively poorly. S3, however, shows no particular strengths and seems to only yield reasonable mappings for low amounts of TDM traffic and few critical tiles (parameter subsets for which the differences between the mappings are smaller to begin with).

The multi-objective mapping strategies S5 to S7 show that it can be very beneficial to combine strategies. S5 combines the strengths of S1 and S2 and, in some cases, even amplifies them, particularly for high amounts of TDM traffic. This is surprising since S1 aims to minimize the number of reserved TDM slots—which typically leads to clustering the critical applications—while S2 aims to use the same amount of TDM slots on each link, which tends to spread out the critical applications and requires more reserved slots, thereby standing in conflict to S1. However, as the evaluation of the samples in Figure 5.4 showed, both maximizing and minimizing the number of reserved TDM slots proved to have a negative effect on the BE traffic. In combination, S2 seems to counteract the tendency of S1 to pick the extreme case of minimizing the reserved slots.

Both S6 and S7 show very good results for TDM traffic generation rates of up to 20%. For both strategies the number of critical tiles seems to have little effect on the quality of the mappings for these generation rates. S7 even produces reasonable results for 25% TDM traffic generation rate and up to 24 critical tiles and, thereby, proves to be the most consistent strategy for a large variety of parameter combinations, which is also reflected in Figure 5.5. In fact, between the two example mappings shown in Figure 5.2 the mapping generated with S7 consistently achieves a higher saturation rate than then one generated with S1, even though way more slots are reserved and the TDM paths are much longer. In the worst case (burst mode with a router buffer size of 8) the S7 mapping achieves a 1% higher saturation rate (22% vs 21%) but in the best case (batch mode with router buffer size 8) the S7 mapping achieves a 4% higher saturation rate (29% vs 25%).

Comparison to Split Regions

When comparing the saturation rates of the mappings generated by strategies S1 to S7 to the saturation rates of the split region systems shown in Figure 5.3 it is obvious that for many subsets the split regions achieve a higher saturation rate[4]. This is not surprising since, due to the 1+1 protection, the amount of TDM traffic injected into the NoC is twice as high as the generated TDM traffic. This means, for TDM generation rates of 15% and higher, the critical tiles inject more traffic into the NoC than is maintainable for most of the BE only systems. Only the 8x6, 8x5, and 8x4 split system in batch mode achieve saturation rates higher than 30%. The degradation of the maintainable BE generation rate is the price for giving fault-tolerance and hard real-time guarantees to the critical applications. However, Figure 5.6 also shows that, for some application scenarios, the proposed strategies can even achieve higher saturation rates than the split region systems. Particularly S6 and S7 achieve very competitive results for a high number of critical tiles (24 and 32) and a low TDM traffic generation rate (5% or 10%).

Another interesting comparison is between the test runs without TDM traffic and the split region systems. Figure 5.7 shows the same matrix as Figure 5.6, only for the runs without TDM traffic. The figure shows that optimization goal O4 seems to work in the

[4]For better comparability, the interested reader can find a detailed comparison of the 1+1 mapping strategies for each basic system individually in the Appendix (figures A.4, A.5, A.6, and A.7).

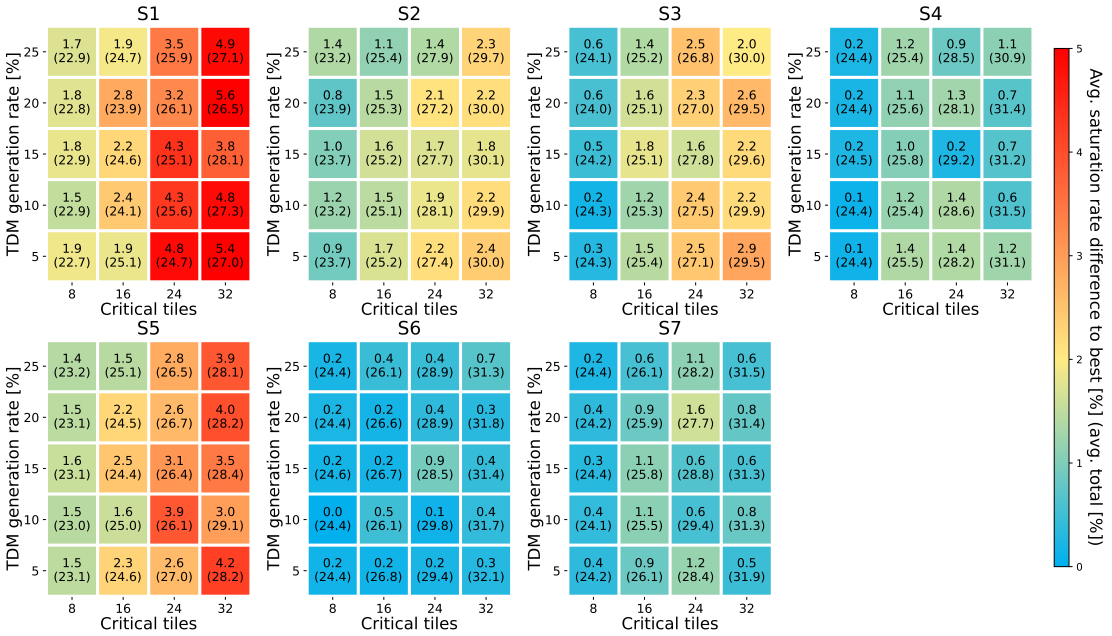


Figure 5.7: Detailed comparison of 1+1 mapping strategies - baseline runs without TDM traffic

intended way which is directly affecting the mapping of the BE tiles (cf. Section 3.5.1). Furthermore, the combination with objective O2 further improves the generated mappings, as shown by the results of S6 and S7. This is interesting in case the critical applications only produce low amounts of traffic overall but require very low latencies, meaning the TDM channels must be designed for a high bandwidth (cf. Section 3.3). In such a system, the BE applications can efficiently make use of the available network resources in periods where no TDM traffic is being sent.

One important aspect when comparing the mappings generated with the mapping strategies to the split region systems is that, due to the 1+1 protection, splitting the system into two application domains is often not possible. For instance, the mapping algorithm could not find any feasible mappings for split regions, a TDM traffic generation rate of 25%, and systems with a slot table size of 4 or 8. Even for a generation rate of 20% and a slot table size of 16 the algorithm typically needed several attempts and a long time to find a feasible mapping. Furthermore, it is generally not possible to split the system in an 8x7 and an 8x1 domain for 8 or fewer critical tiles because this would make it impossible to use disjoint paths with no path traversing the BE domain. Other shapes of split regions would lead to BE traffic traversing the critical regions if XY routing is used. Hence, it can be beneficial and is in some cases even necessary to avoid a strict separation of critical and BE regions.

Effect of Slot Table and Task Graph Sizes

To evaluate the effect of the slot table size on the quality of the generated mappings the histogram in Figure 5.5 is split into five histograms, one for each slot table size. The resulting histograms are shown in Figure 5.8. As can be seen, the overall trend of the different mapping strategies is the same for all slot table sizes. However, the differences between the strategies is less pronounced for the slot table size 4. The reason is that with a small slot table the mapping algorithm has less flexibility to move TDM paths and, therefore, less potential of optimizing a mapping according to the defined strategy. Nevertheless, both mapping strategies S6 and S7 remain good choices for all different slot table sizes.

In general, a smaller slot table size is preferable from chip area and power consumption perspective. However, it should be large enough to give the mapping algorithm sufficient freedom for optimization. It should also be noted that generating optimized mappings for a slot table of size 4 took about 2–10 times as long (in some cases up to ~ 40 hours) as the ones for a slot table size of 16 or more[5]. During the experiments that were done in the scope of this thesis a slot table size of 16 proofed to be a good trade-off and larger slot table sizes were never necessary, including the split region mappings with a TDM traffic generation rate of 25% and 1+1 protection.

To evaluate the effect of the task graph size on the saturation rate of a mapping, the mappings of the smaller task graphs (A) are compared to the larger task graphs (B). Overall, in $\sim 46.3\%$ of all cases the mappings of graph A achieved a higher saturation rate, in $\sim 26.1\%$ graph B achieved the higher saturation rate, and in $\sim 27.6\%$ they achieved the same saturation rate. These percentages change slightly for different mapping strategies and parameter combinations but the overall trend remains the same. This suggests that smaller graphs can be mapped more easily and give more freedom to the mapping algorithm but do not affect the mapping strategy otherwise.

TDM$^+$ Traffic

As a last step, the effect of the TDM$^+$ traffic is evaluated, for which—in addition to the already reserved slots of a mapping—as many additional slots as possible are reserved for each TDM channel. This is done by comparing the saturation rate differences of both the runs with TDM and TDM$^+$ traffic to the reference runs of the same mapping without TDM traffic. Figure 5.9 shows a histogram with the saturation rate differences. As can be seen, the differences are, overall, relatively small. However, the *normal* TDM traffic runs are typically slightly closer to the reference runs, meaning they achieve a higher saturation rate than the TDM$^+$ traffic runs. In $\sim 79.8\%$ of all cases both TDM and TDM$^+$ traffic achieve the same saturation rate. However, in $\sim 15.2\%$ of all cases TDM achieves the higher saturation rate whereas the same happens for TDM$^+$ traffic in only $\sim 5.0\%$ of all cases. These percentages do not change significantly for different

[5] This is arguably not a big issue for the target system and use case defined in Section 3.1 since mappings are only generated offline and infrequently, but is an interesting observation that should be kept in mind if mappings are to be generated at runtime.

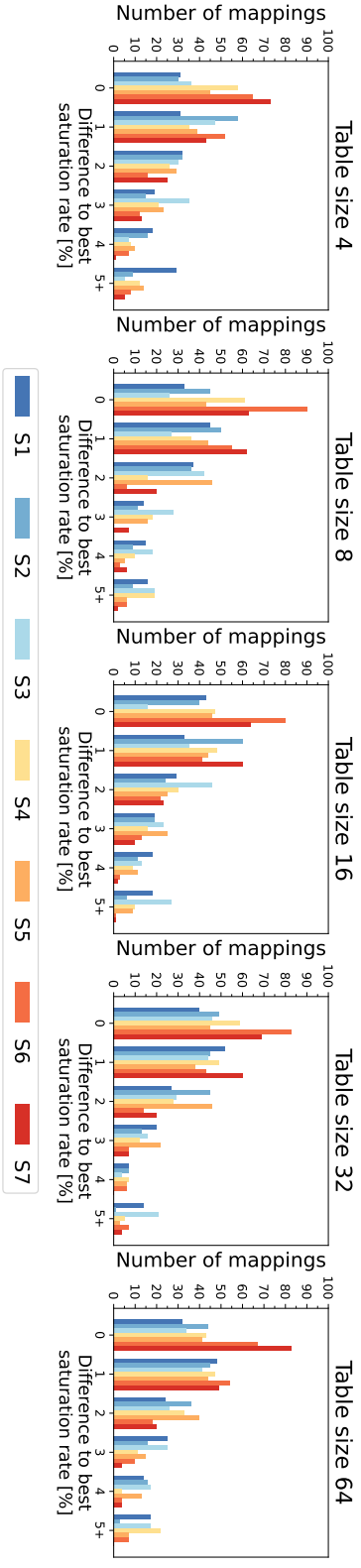


Figure 5.8: Overall comparison of 1+1 mapping strategies for each slot table size

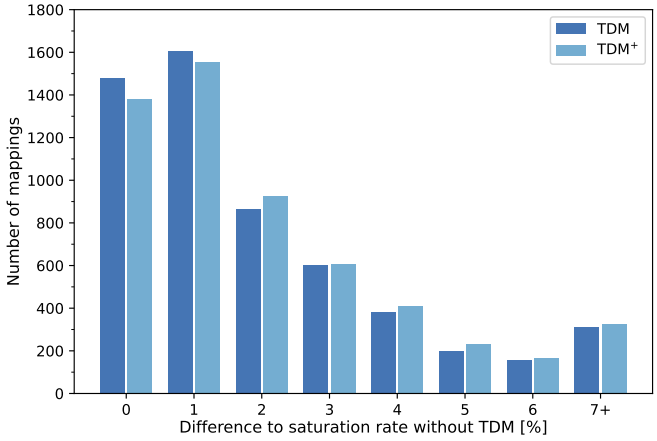


Figure 5.9: Difference of TDM and TDM$^+$ saturation rates to reference (w/o TDM) for 1+1 mappings

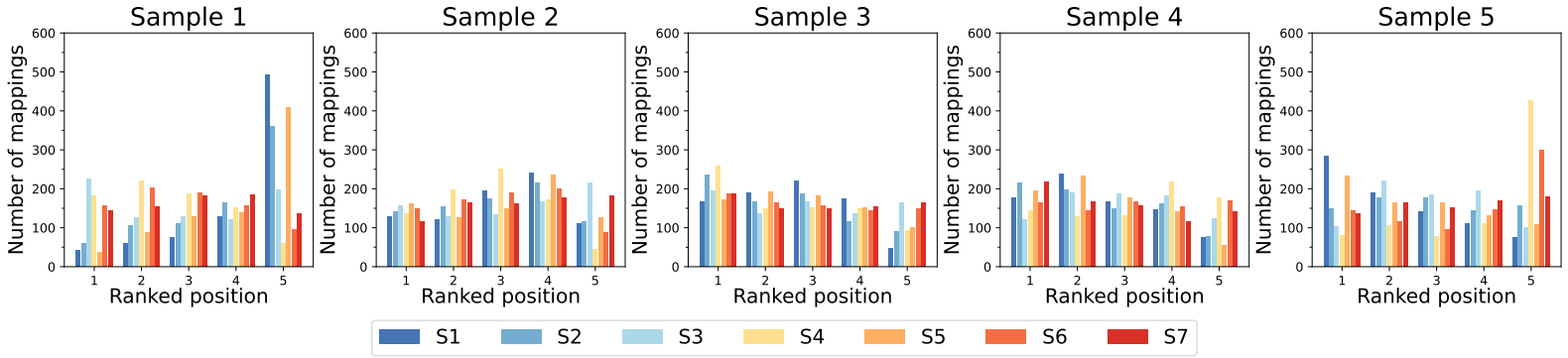
mapping strategies, parameter combinations, or basic system versions. Hence, it can be concluded that reserving more slots than necessary should be avoided since it typically has no benefits for the critical applications but has an adverse effect on the BE traffic performance significantly more often than it has a positive effect.

5.2.2 Mapping Strategies for 1:1 / 1:n Protection

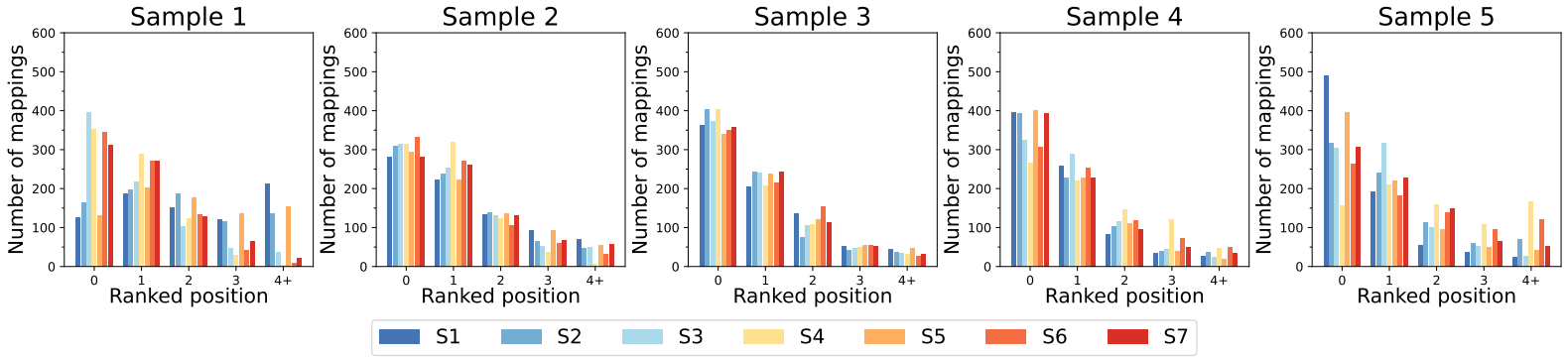
Similar to the 1+1 protection mappings, the samples of the 1:1/1:n protection mappings are compared as a first step. Figure 5.10a shows the histograms with the rankings of the different samples. The results differ greatly from the results for the 1+1 protection mapping and don't show a clear correlation for most of the strategies.

One distinct feature is the peak at position five of sample one of strategies S1, S2, and S5, which shows that the supposedly best sample was ranked last in most cases. Looking at S1, the histograms actually show a clear negative correlation and also suggest a negative correlation for S2. Combined, this also explains the results for S5, the combination of S1 and S2. For the other mapping strategies, no clear correlation can be seen. However, another distinct feature is the peak at position five of sample five of the strategies S4 and S6, which shows that the supposedly worst sample was ranked last in most cases. Overall, S4 still seems to work reasonably well considering that samples 1–3 all have a (small) peak in the first three positions and samples 4 and 5 both have their peak at the expected position.

An explanation for the histograms—particularly the ones for S4 and S6—could be that, with the exception of the worst mappings (i.e., sample 5), the samples all achieve very similar, if not the same, saturation rates. In this case they are ranked according to the average latency at the saturation rate and very small differences can distort the histograms. This is supported by the histograms shown in Figure 5.10b which show the saturation rate difference of each sample to the best sample. As can be seen, for both



(a) Ranking histogram for 1:1/1:n mapping strategy samples



(b) Saturation rate differences for 1:1/1:n mapping strategy samples

Figure 5.10: Histograms for 1:1/1:n mapping strategy samples - all basic system versions combined

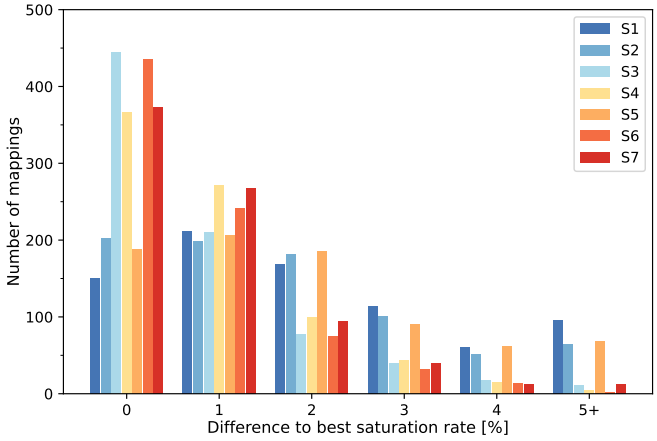


Figure 5.11: Overall comparison of 1:1/1:n mapping strategies

S4 and S6 sample 0 most often achieves the same saturation rate as the best sample or is at least close. For samples 2–5 the number of times the sample has a more than 2% lower saturation rate than the best increases. Interestingly, a similar observation can be made for strategies S3 and S7. Only for S1, S2, and S5 this trend is reversed which confirm the observations from Figure 5.10a.

For the remainder of this section, and similar to the 1+1 mapping evaluation, only sample 1—the allegedly best mapping of each strategy—is used to directly compare the strategies to each other. Figure 5.11 shows a histogram of the difference in percent to the highest saturation rate that is achieved for each parameter combination and for all strategies (the individual histograms for all basic system versions can be seen in Figure A.8 in the Appendix). The first observation that can be made here is that both S4 and particularly S6 achieve saturation rates that most often either match or come close to the highest saturation rate and only rarely have a difference of more than 3%. This is consistent with the observations from figures 5.10a and 5.10b. Furthermore, S7 also achieves fairly good saturation rates. The big surprise, however, is that S3 suddenly consistently produces good mappings with only a small chance of having a difference to the highest saturation rate of more than 3%. Overall, the histogram shows that the differences of the achieved saturation rates for strategies S3, S4, S6, and S7 are generally smaller than for the 1+1 mappings (cf. Figure 5.5). For strategies S1, S2, and S5, on the other hand, the histogram shows that they do not seem to work very well for 1:1/1:n protection and the generated mappings have a high chance of having a much lower saturation rate than the best mapping. This too is consistent with the observations from figures 5.10a and 5.10b.

Figure 5.12 shows a more fine-grained comparison of the mapping strategies for subsets of the overall results. This comparison confirms the observations from the histograms. Strategies S3, S4, S6, and S7 all consistently generate relatively good mappings. S3 only seems to struggle with high TDM traffic rates and 32 critical tiles. S6 and S7 once again show that it can be very beneficial to combine strategies as they both are most

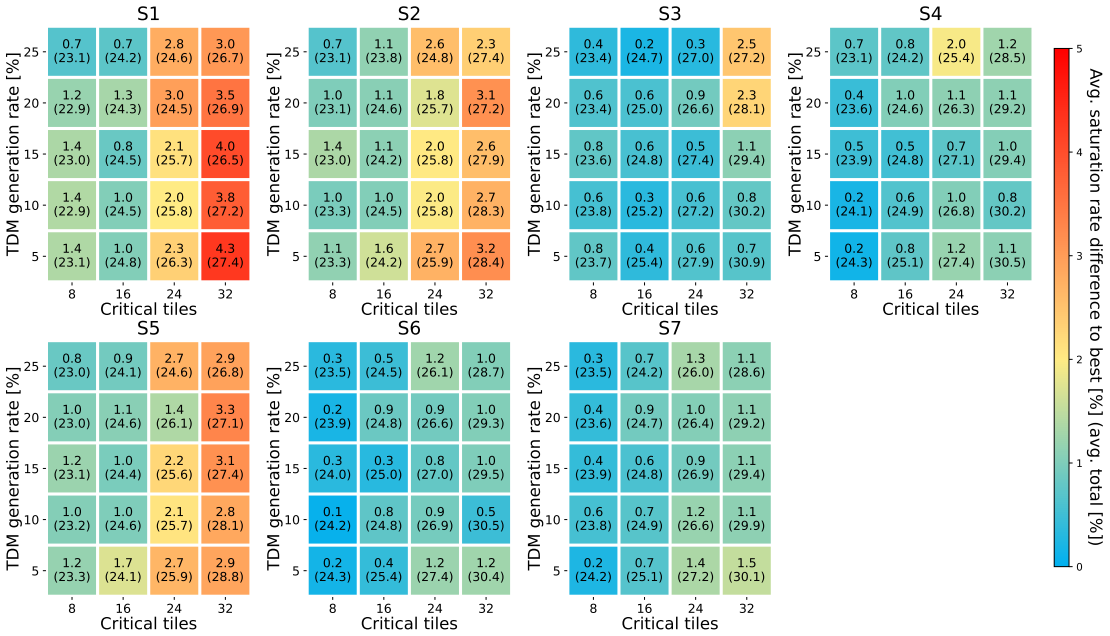


Figure 5.12: Detailed comparison of 1:1/1:n mapping strategies

consistent across all subsets of parameters. Strategies S1, S2, and S5, on the other hand, do not seem to produce good mappings, particularly for a larger number of critical tiles. In conclusion it can be said that, while S7 seems to have a slight edge over S6 for 1+1 mappings the opposite seems to be the case for 1:1/1:n mappings. Nevertheless, both strategies are good options and are the most consistent between strategies S1–S7.

A general observation that can be made when comparing figures 5.12 and 5.6 is that the 1:1/1:n mappings mostly achieve a higher saturation rate than the corresponding 1+1 mappings. The only exception are some of the mappings with a low TDM traffic generation rate. Compared to the 1+1 mappings the achievable saturation rate also decreases less with increasing TDM traffic generation rates. This is of little surprise since the amount of TDM traffic that can interfere with the BE traffic is cut in half.

Comparison to Split Regions

The comparison to the split region systems is very similar to the comparison of the 1+1 mappings to the split regions which is why an extensive but repetitive comparison is omitted here. In general, the achievable saturation rates of the 1:1/1:n mappings are typically lower than the ones of the split region systems. However, the saturation rates are naturally closer than for the 1+1 mappings, especially for high amounts of TDM traffic. For better comparability, the interested reader can find a detailed comparison of the 1:1/1:n mapping strategies for each basic system individually in the Appendix (figures A.9, A.10, A.11, and A.12). Overall, similar conclusions can be drawn as for the 1+1 mappings: Spreading out the critical applications has an adverse effect on the

BE traffic but also allows BE traffic to efficiently utilize resources in times where they are not used for critical communication. Furthermore, in some cases it is not possible to find feasible TDM channel mappings when splitting the system into different application domains.

Effect of Slot Table and Task Graph Sizes

Analogous to the evaluation of the 1+1 mapping strategies, Figure 5.13 shows five histograms with the results of the mapping strategies for the different slot table sizes. Overall, similar observations can be made here. The mapping algorithm has more difficulties optimizing mappings for a slot table size of 4. This can especially be seen at the example of S7. However, in contrast to the 1+1 mappings a slot table size of 8 already seems to give sufficient freedom to the mapping algorithms in order to optimize the mappings according to the defined strategy, especially when disregarding strategies S1, S2, and S5.

The effect of the task graph size on the saturation rate of a mapping is similar to the 1+1 mappings but even more pronounced in favor of graph A. Overall, in $\sim 57.0\%$ of all cases the mappings of graph A achieved a higher saturation rate, in $\sim 14.2\%$ graph B achieved the higher saturation rate, and in $\sim 28.8\%$ they achieved the same saturation rate. Again, these percentages change slightly for different mapping strategies and parameter combinations but the overall trend remains the same and confirms the findings from the 1+1 mappings.

TDM$^+$ Traffic

As a last step, the effect of the TDM$^+$ traffic is evaluated for the 1:1/1:n mappings. Again, the results are similar to the 1+1 mappings. However, since the amount of TDM traffic is only half that of the 1+1 mappings the differences to the reference runs without TDM traffic are much lower, as can be seen in Figure 5.14. Nevertheless, the *normal* TDM traffic typically achieves a slightly higher saturation rate than the TDM$^+$ traffic. Overall, in $\sim 84.5\%$ of all cases both TDM and TDM$^+$ traffic achieve the same saturation rate, in $\sim 9.5\%$ TDM achieves the higher saturation rate and in $\sim 6.0\%$ TDM$^+$ traffic achieves the higher saturation rate. Again, these percentages do not change significantly for different mapping strategies, parameter combinations, or basic system versions. Although the ratio is different than for the 1+1 mappings the TDM$^+$ traffic still has an adverse effect on the BE traffic significantly more often than it has a positive effect. Hence, the same conclusion can be drawn as for the 1+1 mappings: reserving more slots than necessary for the critical traffic should be avoided.

5.3 Protection Switching Comparison

In this section the protection switching versions 1:n, 1:1, and 1+1 are compared based on the results of the FPGA-based evaluation in the previous Section 5.2 in order to reevaluate the assessments made in Section 3.4.2. The protection switching versions

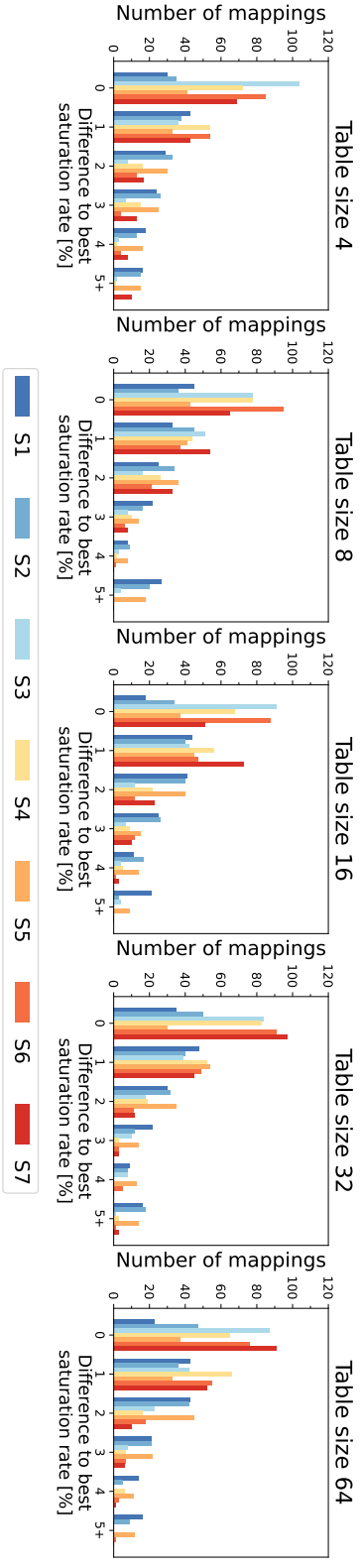


Figure 5.13: Overall comparison of 1:1/1:n mapping strategies for each slot table size

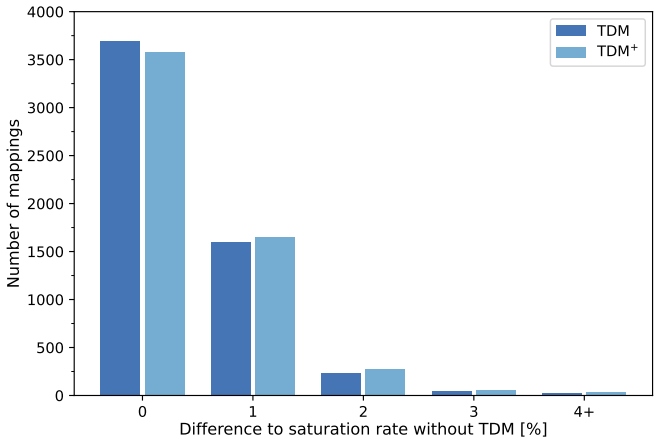


Figure 5.14: Difference of TDM and TDM$^+$ saturation rates to reference (w/o TDM) for 1:1/1:n mappings

are compared based on their effect on the BE traffic in Section 5.3.1, and then their respective latency, hardware, and power overhead are discussed in Section 5.3.2.

5.3.1 Effect on BE Traffic

For the comparison of the effect that the protection switching versions—1+1 on the one hand and 1:1/1:n on the other hand—have on the BE traffic, the different mapping strategies are not considered as they have already been evaluated and discussed in sections 5.2.1 and 5.2.2 respectively. Instead, for each traffic scenario the mapping with the highest saturation rate for 1+1 protection is compared to the mapping with the highest saturation rate for 1:1/1:n protection. The greater the difference between the saturation rates is, the stronger the adverse effect of the 1+1 protection on the BE traffic.

Overall, 1+1 protection was compared to 1:1/1:n protection for 200 different application scenario and constraint combinations for all four basic system versions, resulting in a total of 800 comparisons. Figure 5.15 shows the histogram with the saturation rate differences from 1+1 protection to 1:1/1:n protection. As expected, mappings with 1+1 protection typically have a lower saturation rate than mappings with 1:1/1:n protection[6]. However, in most cases the differences are relatively low. Across all comparisons, the 1+1 mappings saturated, on average, only at a $\sim$ 1.0% lower BE generation rate than the 1:1/1:n mappings. The worst case was a 8% lower saturation rate (one occasion) and in only 14 cases the difference was more than 4%. In $\sim$ 94.6% of all cases 1+1 protection had no more than a 3% lower saturation rate and in $\sim$ 86.9% of all cases no more than 2%. This shows that the adverse effect that 1+1 protection has on the BE

[6] On rare occasions the 1+1 mappings actually achieved a higher saturation rate. However, these occasions were counted as a difference of '0' as the 1+1 mapping could also be used for 1:1/1:n protection by simply only using one path at a time.

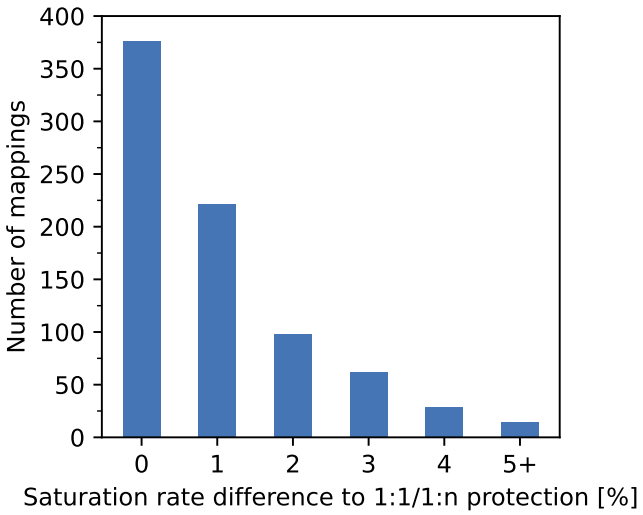


Figure 5.15: Saturation rate difference between 1+1 mappings and 1:1/1:n mappings

traffic can in most cases effectively be mitigated and in many cases even be nullified by good mapping strategies.

For a more fine-grained evaluation, the mappings are compared for different parameter subsets, similar to the evaluation in sections 5.2.1 and 5.2.2. Figure 5.16 shows the average total saturation rate for both 1+1 protection (upper number) and 1:1/1:n protection (lower number) for each combination of number of critical tiles and TDM traffic generation rate. The color coding of the cells show the average difference between the saturation rates achieved by the mappings for the different switching versions. The figure shows that the switching versions achieve almost the same saturation rates for most of the parameter subsets—particularly for low amounts of TDM traffic and/or few critical tiles. 1+1 protection only affects the the BE traffic more heavily with both a high number of critical tiles and a high TDM traffic generation rate. However, many safety-critical hard real-time applications do not exchange or produce large amounts of traffic but instead only sent relatively short messages (e.g., control messages or periodic sensor data). This means that for many use cases the difference of the saturation rates of both 1+1 and 1:1/1:n protection is negligible.

An interesting observation that can be made from Figure 5.16 is that BE traffic seems to interfere more with itself than TDM traffic interferes with BE traffic. This can be seen when comparing the saturation rates for a TDM traffic generation rate of both 20% and 25% and an increasing number of critical tiles. With more critical tiles, fewer tiles generate and inject BE traffic which increases the available bandwidth for each BE tile. For low TDM traffic generation rates this is of little surprise as less traffic is injected into the NoC overall. However, with 1+1 protection the TDM traffic injection rate is twice the generation rate which means that for a generation rate of 20% and 25% each critical tile that generates traffic has an injection rate 40% and 50% respectively. Yet, the achievable BE injection rates increase with growing number of critical tiles even for

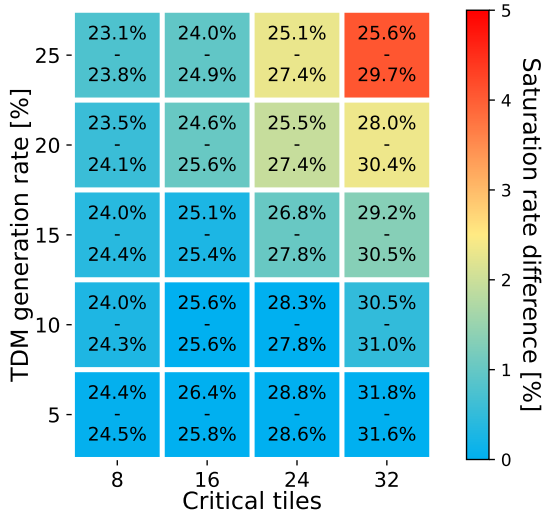


Figure 5.16: Detailed comparison of the protection switching versions. Average total saturation rate of 1+1 protection (top number) and 1:1/1:n protection (bottom number).

1+1 protection and a TDM traffic generation rate of 20% and 25%. The reason is that the TDM traffic typically only reduces the available bandwidth for the BE traffic on a link, but never blocks it entirely. This reduces backpressure and causes fewer BE packets to be blocked throughout the NoC.

Overall, it can be concluded that the FPGA-based evaluation supports the preliminary findings of the proof of concept in Section 3.4: The performance *cost* of the BE traffic when using protection switching is more than outweighed by the additional bandwidth for the critical TDM traffic—especially when using good mapping strategies—and that even the additional adverse effect of 1+1 protection on the BE traffic is well affordable.

5.3.2 Latency, Hardware, and Power Overhead

In this last part of the comparison, the additional latency, hardware, and power overheads that can be expected for each protection switching version are discussed.

The worst-case latencies of the three protection switching versions have been analyzed and compared in sections 3.3 and 3.3.1 respectively. In essence, 1+1 protection has the lowest worst-case latency which is virtually equal to the worst-case latency of a TDM channel without protection switching (and assuming no faults can occur on that unprotected channel). The worst-case latencies of both 1:1 and 1:n protection can easily be more than twice as high, depending on the checkpoint distance and the additional latencies for both the feedback channel to the sending NI and for the potentially necessary configuration of the secondary path in case of 1:n protection. Especially the last two parameters—latency caused by the feedback channel and the configuration of the secondary path—depend on the implementation of the overlay network. Overall, it

can be assumed, though, that 1+1 protection has the lowest worst-case latency and 1:n protection hat the highest worst-case latency.

Both 1:1 and 1:n protection require an overlay network which, as discussed in Section 3.2.3, must be fault-tolerant itself. For both versions, it might be possible to save some resources in the NIs—in comparison to the NI architecture presented in Section 4.2.1.2 which supports 1+1 protection—by using a single in- and out-queue, however, both versions require larger buffers in the NIs to store messages that have been sent, until it can be assumed that they have been received without faults. 1+1 protection, on the other hand, does not require such an overlay network and additional buffers. Hence, without considering a specific implementation of the overlay network, it can be assumed that the hardware overhead is lower for 1+1 protection.

On the other hand, the power overhead of 1+1 protection in comparison with 1:1 and 1:n protection can be expected to be at least 100% since every flit is sent twice (100% if both paths have the same length, otherwise it will be more). However, this overhead only affects the critical TDM traffic in the NoC. Compared to the packet switched BE traffic the power consumption of the TDM traffic is much lower to begin with since no buffer queues are used in the routers, just single register stages. Furthermore, it can be assumed that the overlay network that is necessary for 1:1 and 1:/n protection also causes additional power dissipation, even if it is dormant for the majority of time. Hence, it can be assumed that the power overhead of 1+1 protection is not considerably higher than the one of 1:1 and 1:n protection, if at all.

5.4 Discussion

In this final part a brief summarizing discussion on the different protection switching versions and their advantages and disadvantages is presented together with a comparison to different related work which also addresses the challenge of implementing GS in NoCs for mixed-critical systems.

The FPGA-based evaluation has shown that, overall, all three protection switching versions can achieve good saturation rates and are suitable candidates to implement fault-tolerant and hard real-time capable communication in mixed-critical MPSoCs. The question is: which of the three protection switching versions is most suited?

Both 1:1 and 1:n protection have the same effect on the BE traffic. However, the worst-case latency of 1:n protection is (potentially much) higher than the one of 1:1 protection and with the additional step of configuring the secondary path there are more steps in which something can go wrong. Furthermore, since finding mappings for 1:1 (and 1+1) protection is not difficult, at least for slot tables of size 8 and higher, there is no need for different critical connections to share the same backup path. Hence, 1:1 is to be favored over 1:n protection.

Between 1:1 and 1+1 protection, 1:1 protection has a lower adverse effect on the BE traffic and, potentially, a lower power overhead. On the other hand 1+1 protection has a lower worst-case latency and hardware overhead. The question is, which is more important? It is the author's opinion that 1+1 protection is the better choice for multiple

Approach	NoC Size	Saturation Rate
QoSinNoC [102]	4x4	24%–36%
Forward Pressure [99]	8x8	22.2%–26.7%
CDG-Driven Strong Isolation[50]	4x4	10%–17.5%
DCFNoC [103]	3x3 / 6x6	12% / 3%
1+1 protection	8x8	23.1%–31.8%

Table 5.2: Different approaches of implementing GS in NoCs for mixed-critical systems

reasons. First, the evaluation of the proposed mapping strategies showed that good mapping strategies can mitigate or even nullify the additional adverse effect that 1+1 protection has on the BE traffic. Particularly the multi-objective strategies S6 and S7 consistently produce good mappings. Furthermore, it is important to keep in mind that protection switching is intended to be used in safety-critical systems where fault-tolerance and hard real-time guarantees are of the highest priority. Since with 1+1 protection no feedback channel and re-sending in case of a fault is needed there is yet one less step during which faults can occur. Hence, even *if* 1+1 should have a higher power (or even hardware) overhead it should still be preferred over 1:1 protection.

For a similar reason the critical applications and traffic should be spread out across the system even *if* splitting the system into critical and non-critical domain would be possible and spreading out the critical applications has an adverse effect on the BE traffic. With the critical communication spread out, there is a much lower chance that a single fault affects multiple TDM paths at once, which increases the overall resilience of the system and would make it easier to search for new alternative paths at runtime (cf. Section 3.2.4).

Overall, even the saturation rates that are achieved by the 1+1 mappings are quite competitive when comparing them to other approaches of implementing GS in NoCs for mixed-critical systems. Simulations of a 4x4 NoC presented in [102] result in saturation rates between 24%–36%. In [99], simulations of an 8x8 NoC with two overlapping critical data streams and a combined link utilization of 50% on these overlapping links result in saturation rates between 22.2%–26.7%. The approach proposed in [50] was evaluated with simulations of a 4x4 NoC and reached saturation rates of 10%–17.5%, depending on the traffic scenario and the number of application domains. And in [103], results obtained with an RTL simulator showed saturation rates of 3%–12%, depending on the NoC size (12% in a 3x3 NoC, 3% in a 6x6 NoC). The results, including the ones for 1+1 protection, are summarized in Table 5.2[7].

[7] The 1+1 protection results are taken from Figure 5.16. The values vary between the four basic system versions and go from 20.8%–29.3% in the worst case to 26.8%–35.0% in the best case (cf. Figure A.13 in the Appendix).

5.5 Summary

Since a comprehensive validation of the preliminary findings in Section 3.4 and evaluation of the mapping strategies proposed in Section 3.5.1 on the basis of a large number of different application scenarios would require a prohibitive amount of time in simulation, an FPGA-based evaluation is done. Four different basic system versions are implemented—with different router buffer sizes and BE traffic behavior—and each system is synthesized with five different slot table sizes: 4, 8, 16, 32, and 64.

Two task graphs are defined to represent critical applications and are combined in a total of 40 different example application scenarios. For each of these scenarios, mappings are created according to the mapping strategies S1–S7 for both 1+1 protection on the one hand and 1:1/1:n protection on the other hand. For each mapping, the saturation rate—i.e., the rate at which the BE flit injection rate cannot keep up with the generation rate—is determined by running tests with increasing BE generation rates. Furthermore, the saturation rate of a system with 8x8 BE tiles as well as the saturation rates of different split region systems with 8x4, 8x5, 8x6, and 8x7 BE tiles are determined for comparison.

The mapping strategies are first evaluated for 1+1 protection and 1:1/1:n protection individually and afterwards compared to each other. Also, the effect that both the slot table size and the task graph size of the critical applications have on the quality of the mappings is evaluated. The results show that especially the multi-objective strategies S6 and S7 consistently produce good mappings with a relatively high saturation rate. Furthermore, the results show that the 1:1/1:n mappings generally achieve higher saturation rates—which is unsurprising since less TDM traffic is being injected into the NoC—but that good mappings can help to mitigate and in many cases even nullify the adverse effect that 1+1 protection has on the BE traffic.

It is concluded that 1+1 protection, despite its slightly lower saturation rates and potentially larger power overhead, is the protection switching version that is best suited to implement fault-tolerant and hard real-time capable communication in mixed-critical MPSoCs. A final comparison with related work that addresses the implementation of GS in NoCs for mixed-critical systems shows that even 1+1 protection can achieve very competitive saturation rates.

6 Conclusion

In recent years, the demand for computational power in safety-critical systems—such as in the avionics and, particularly, the automotive domain—has increased drastically. In case of automotive, this demand is mostly driven by an increasing amount of ADAS and the movement towards autonomous driving. This increased demand motivated the ARAMiS (and ARAMiS II) research project with the goal of answering the question *if* and *how* MPSoCs could safely be used in safety-critical environments and, potentially, even in mixed-critical systems which implement both safety-critical and non-critical applications on the same chip. The work in this thesis was motivated by the question of which features a NoC for such a mixed-critical MPSoC would have to provide, how such a NoC could be implemented, and how the resources of such a NoC could be utilized most efficiently.

Today, a vast amount of research and related work exists on the topic of NoCs in general. A large part of this research is dedicated to either maximizing the performance or minimizing the cost of NoCs. Less work, however, exists on the topic of fault-tolerance and mixed-critical communication in NoCs—particularly addressing hard real-time guarantees—and, to the best of the author's knowledge, no work yet exists that considers both in a holistic approach. The work presented in this thesis closes this gap.

The target system that is assumed in the scope of this thesis is a mixed-critical MPSoC. The critical applications are assumed to be known either at system design time or at compile time of the software and static in nature, meaning they are not dynamically started and stopped during runtime and their behavior is known. The BE applications, on the other hand, can be started and stopped dynamically and their behavior is not known beforehand. For such a system, the requirements that a NoC must meet are determined. Most notably, the NoC must support multiple concurrent critical traffic streams while providing isolation between the different traffic streams and BE traffic, provide fault-tolerance to the critical communication, guarantee in-order delivery, and give QoS guarantees to critical communication.

The approach presented in this thesis is based on a hybrid TDM and packet switched NoC. TDM is used for critical traffic while packet switching is used for non-critical BE traffic. In contrast to other works that use TDM in NoC, the TDM channels are *semi-static*, meaning they are configured at system start time and then only adjusted when necessary, e.g., due to a fault. This way, both QoS guarantees and traffic isolation can be provided to the critical communication. Using packet switching for the BE traffic, on the other hand, allows the BE traffic to utilize the bandwidth that is not or cannot be used by the TDM traffic and gives maximum flexibility to the BE communication.

The main contribution of this thesis is the adoption of protection switching to NoCs in order to implement fault-tolerant communication. In essence, two disjoint paths are

defined for each communication channel and in case of a fault on one of the paths the other is still fully functional. Protection switching is exclusively used for the critical TDM traffic. Three different protection switching versions are considered: 1:n, 1:1, and 1+1 protection. The necessary adjustments are introduced and discussed, most notably a second local link connecting each NI to its neighboring NoC router, and checkpoints that are used for path synchronization. A worst-case timing analysis of the three protection switching versions is provided which is the basis for a first comparison.

A proof of concept based on cycle accurate simulations with an 8x8 NoC and two different application scenarios showed that the approach works and enabled a first evaluation of the protection switching versions and their effect on the packet switched BE traffic. All three protection switching versions have an adverse effect on the performance (i.e., the available bandwidth and average latency) of the BE traffic which was to be expected since more traffic is generated overall. This adverse effect is worst in case of 1+1 protection since every critical flit is sent twice. However, the simulations also showed that the achievable traffic injection rate of both TDM and BE traffic combined is higher than for BE traffic alone.

The second contribution is the design of the architecture of the hybrid NoC as well as two systems using the hybrid NoC: an evaluation system used to further evaluate the protection switching versions as well as different mapping strategies, and a demonstrator system to showcase the capabilities of protection switching in NoC with an example use case in an interactive system. Both systems are implemented on an FPGA.

The evaluation system uses special generator tiles that can be configured to generate both TDM and BE traffic with a certain rate, either statically or dynamically based on random numbers. These generator tiles not only allow tests to run without direct interaction of a host-PC—which would make precise timing virtually impossible—but also enable the evaluation of an 8x8 NoC in hardware since they are smaller than processing elements which would prohibit such a large design due to the limited resources of the FPGA. The demonstrator system consists of a 4x4 NoC with 15 processing tiles and an I/O tile which is used for the communication with a host-PC. The host-PC sends individual images to some of the processing tiles which run a pedestrian detection algorithm to represent safety-critical applications. Furthermore, the host-PC provides a GUI which allows users to inject faults in the NoC to test the fault-tolerance of the critical communication, reconfigure TDM paths at runtime, and control the generation of BE background traffic to demonstrate the traffic isolation.

For both systems synthesis results are presented. The results show that the NoC is relatively small despite providing both TDM and packet switched traffic. In comparison to a state-of-the-art purely packet switched NoC that is designed to provide QoS guarantees the routers of the hybrid NoC with protection switching require $\sim$ 36%–48% fewer LUTs and $\sim$ 31%–46% fewer registers (depending on whether the hybrid router is compared to the packet switched router version with 2 or with 3 VCs). The reason is that the implementation of the packet switched traffic can be kept very basic and, e.g., does not require different VCs since no guarantees must be given to the packet switched traffic.

The third contribution is the definition and, particularly, the evaluation of seven different mapping strategies—especially the ones using multi-objective optimizations—that are designed to maximize the available bandwidth for the BE applications for a given set of safety-critical applications in order to use the hybrid NoC most efficiently. The mapping strategies are based on four different optimization goals which are used to define four single-objective and three multi-objective mapping strategies. In order to evaluate these strategies, a larger number of tests than the ones for the proof of concept and with a larger number of different application scenarios is necessary. Since this would take a prohibitive amount of time with the cycle-accurate simulations that were done for the proof of concept, these tests were done with the evaluation system implemented on FPGA.

The FPGA-based evaluation of the the mapping strategies showed that evenly spreading out the critical applications across all rows and columns of a tiled MPSoC while at the same time evenly spreading out the critical TDM traffic across all links of the NoC—potentially while additionally minimizing the number of slots that are overall reserved for the TDM traffic—consistently results in mappings that are most beneficial for the packet switched BE traffic. This strategy works well for both 1+1 protection on the one hand and 1:1/1:n protection on the other hand, although differences to worse mapping strategies are more pronounced for 1+1 protection due to the higher amount of TDM traffic. The results show that the slot table size has only little influence on the quality of the mapping as long as it is large enough to enable optimizations to be made (a slot table size of 8–16 seems to be a good trade-off).

A comparison of the different protection switching versions confirms that 1+1 protection has a worse adverse effect on the BE traffic than both 1:1 and 1:n protection have, but that this effect can be mitigated and in many cases even be nullified by using a good mapping strategy. A comparison to related work shows that even 1+1 protection allows for very competitive BE traffic injection rates of up to $\sim$ 23%–31% in an 8x8 NoC, depending on the traffic scenario. The conclusion is that 1+1 protection is the protection switching version that is best suited to be used in mixed-critical MPSoCs since it does not require a special overlay network—which would pose an additional challenge and potential point of failure—and since it has a (much) lower worst-case latency than both 1:1 and 1:n protection.

6.1 Outlook

This thesis describes, to the best of the author's knowledge, the first NoC that provides both fault-tolerance and hard real-time communication to critical applications in a mixed-critical MPSoC in a holistic approach. Nevertheless, there are several different directions for potential future research on the topic.

In this thesis, only faults that corrupt the flits of a TDM channel are considered and faults that affect the status flags, slot tables, or cycle counters of the slot tables are left aside. This aspect should be addressed in future work. For the status flags, a simple and proven approach would be to use two wires with a differential signal for each flag, for

error detection, or to use TMR for error correction. However, it might also be possible to encode the different possible flag combinations that can occur with fewer signals and use, e.g., hamming codes for fault-detection and correction in order to lower the overall hardware overhead. A similar approach could be used to detect and maybe also correct faults that occur in slot tables. Corruption of the cycle counters that point to the current slot in the slot tables is a major threat as this permanently affects all connections that are configured in the slot table. However, since each slot table has its own cycle counter there already is a high degree of redundancy in each NoC router and NI. A possible approach would be some kind of voting system in each router and NI that could be used to automatically correct faulty cycle counters, thereby implementing a self-healing property.

Naturally, the NoC is only one part of the system that can be affected by faults. While fault-tolerance of other resources, specifically processing elements, is a whole other topic on its own, it would be interesting to consider runtime task migration from one processing tile to another to react to faults in a tile. By extending the approach of protection switching, specific backup tiles could be determined at compile time together with backup paths that would be used for the migration itself and the normal operation afterwards. These backup tiles could be used normally by BE applications until a migration of a critical task becomes necessary. Interesting considerations would be the consequences for the worst-case latencies in case of a fault and possible adjustments to the mapping algorithm in order to find feasible mappings. A possible way of implementing such a task migration without running out of slots in the slot tables could be to use a secondary "shadow" slot table that can be switched to and that holds connection configurations that are only needed temporarily. Such shadow slot tables could generally be used for changes in modes of operation.

Another interesting topic would be a lightweight version of the mapping algorithm and an implementation of a central NoC manager which can find and configure backup paths at runtime in order to return from a non-fault-tolerant state to a fault-tolerant state after a fault occurred.

So far, the protection switching only considers disjoint paths between a single sender and receiver. An investigation of possibilities to use protection switching with multicasts and broadcasts would be compelling. Also, the necessary extension of the mapping algorithm and reevaluation of the mapping strategies in multicast and broadcast scenarios would be an interesting challenge. Similarly, protection switching in NoCs with different topologies such as torus or star could be investigated, or how protection switching effects packet switched traffic that uses adaptive routing algorithms.

One aspect that remains a challenge for TDM traffic in general is the access to heavily shared resources such as DDR memory interfaces or I/O tiles. This is typically addressed by using larger slot tables which, naturally, comes with a larger hardware overhead. Other works use multiple interfaces in parallel. Overall, this is an area with a lot of potential for improvement. A possible approach could be to use larger slot tables in the NIs and smaller ones in routers. This would limit the possible mappings of the TDM channels but could help reduce the hardware cost.

Lastly, the mapping strategies could be further improved and refined to lower the chance of a *bad* mapping being produced. This could, e.g., be done by actively considering the routing algorithm of the packet switched traffic and analyzing potential bottlenecks depending on the location of the BE tiles.

Bibliography

[1] Deloitte. Automotive electronics cost as a percentage of total car cost worldwide from 1970 to 2030. Accessed: 2022-07-30. URL: https://www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide/#statisticContainer.

[2] Yano Research Institute Ltd.. Global Market of ADAS & Autonomous Driving Systems - Key Research Findings 2019. Accessed: 2022-07-31. URL: https://www.yanoresearch.com/en/press-release/show/press_id/2134.

[3] N. A. V. Doan, M. Koenen, T. Wild, and A. Herkersdorf. Multi-Objective Optimization of Channel Mapping for Fail-Operational Hybrid TDM NoCs. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pages 201–207, Nov 2019.

[4] AutoScout24. Unser Auto von morgen - 2015. Accessed: 2022-07-31. URL: https://doczz.net/doc/5960743/unser-auto-von-morgen-2015.

[5] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[6] R. H. DENNARD, F. H. GAENSSLEN, H.-N. YU, V. LEO RIDEOVT, E. BASSOUS, and A. R. LEBLANC. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, Oct 1974.

[7] H. Sutter. The Free Lunch Is Over, March 2005. Accessed: 2022-08-04. URL: http://www.gotw.ca/publications/concurrency-ddj.htm.

[8] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, NY, USA, 1967. Association for Computing Machinery. URL: https://doi.org/10.1145/1465482.1465560, doi:10.1145/1465482.1465560.

[9] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011.

[10] H. Sutter. Welcome to the Jungle, November 2011. Accessed: 2022-08-04. URL: https://herbsutter.com/welcome-to-the-jungle/.

[11] Apple at Work M1 Overview. Technical report, Apple Inc., 2021. Accessed: 2022-08-04. URL: `https://www.apple.com/euro/business/mac/pdf/Apple-at-Work-M1-Overview.pdf`.

[12] ARAMiS II Research Project. Supported by the German BMBF, funding ID 01 IS 16025. Accessed: 2022-08-10. URL: `https://www.aramis2.com`.

[13] ISO 26262 Road vehicles — Functional safety. Technical report, International Organization for Standardization, 2018.

[14] M. Koenen, N. A. V. Doan, T. Wild, and A. Herkersdorf. A Hybrid NoC Enabling Fail-Operational and Hard Real-Time Communication in MPSoC. In *ARCS 2019 - 32nd International Conference on Architecture of Computing Systems*, May 2019.

[15] M. Koenen, N. A. V. Doan, T. Wild, and A. Herkersdorf. Channel Mapping Strategies for Effective Protection Switching in Fail-Operational Hard Real-Time NoCs. In *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '19, 2019. `doi:10.1145/3313231.3352372`.

[16] M. Koenen, N. A. V. Doan, T. Wild, and A. Herkersdorf. Exploring Task and Channel Mapping Strategies in Fail-Operational and Hard Real-Time NoCs. In *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7, 2020. `doi:10.1109/NorCAS51424.2020.9264993`.

[17] M. Koenen, N. A. V. Doan, T. Wild, and A. Herkersdorf. Protection Switching Schemes and Mapping Strategies for Fail-Operational Hard Real-Time NoCs. *Microprocessors and Microsystems*, 87:104385, 2021. `doi:https://doi.org/10.1016/j.micpro.2021.104385`.

[18] S. Pasricha and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., 2008.

[19] L. B. Giovanni De Micheli. *Networks on Chips - Technology and Tools*. Morgan Kaufmann Publishers Inc., 2006. `doi:https://doi.org/10.1016/B978-0-12-370521-1.X5000-0`.

[20] C.-T. Hsieh and M. Pedram. Architectural energy optimization by bus splitting. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(4):408–414, 2002. `doi:10.1109/43.992764`.

[21] R. Lu and C.-K. Koh. A high performance bus communication architecture through bus splitting. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, pages 751–755, 2004. `doi:10.1109/ASPDAC.2004.1337693`.

[22] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 684–689, 2001. `doi:10.1109/DAC.2001.156225`.

[23] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro*, 32(2):20–27, 2012. doi:10.1109/MM.2012.12.

[24] D. L. Mulnix. Intel® Xeon® Processor Scalable Family Technical Overview. Technical report, Intel Corporation, Oct 2019. Accessed: 2022-04-05. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html.

[25] G. Dimitrakopoulos, A. Psarras, and I. Seitanidis. *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*. Springer Publishing Company, Incorporated, 2014.

[26] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 890–895 Vol.2, Feb 2004. doi:10.1109/DATE.2004.1269001.

[27] N. Kavaldjiev, G. Smit, and P. Jansen. A virtual channel router for on-chip networks. In *IEEE International SOC Conference, 2004. Proceedings.*, pages 289–293, 2004. doi:10.1109/SOCC.2004.1362438.

[28] A. Burns, J. Harbin, and L. S. Indrusiak. A Wormhole NoC Protocol for Mixed Criticality Systems. In *2014 IEEE Real-Time Systems Symposium*, pages 184–195, 2014. doi:10.1109/RTSS.2014.13.

[29] M. Panic, C. Hernandez, E. Quinones, J. Abella, and F. J. Cazorla. Modeling high-performance wormhole nocs for critical real-time embedded systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, 2016. doi:10.1109/RTAS.2016.7461342.

[30] T. Moscibroda and O. Mutlu. A Case for Bufferless Routing in On-Chip Networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 196–207, New York, NY, USA, 2009. Association for Computing Machinery. URL: https://doi.org/10.1145/1555754.1555781, doi:10.1145/1555754.1555781.

[31] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato. A bufferless switching technique for NoCs. In *Wina*, 2008.

[32] M. Hayenga, N. Enright Jerger, and M. Lipasti. SCARAB: A single cycle adaptive routing and bufferless network. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 244–254, 2009. doi:10.1145/1669112.1669144.

[33] R. M. Metcalfe. Packet communication. Technical report, USA, 1973.

[34] S. Liu, A. Jantsch, and Z. Lu. Parallel probe based dynamic connection setup in TDM NoCs. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014. doi:10.7873/DATE.2014.252.

[35] D. Wiklund and D. Liu. SoCBUS: switched network on chip for hard real time embedded systems. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 8 pp.–, 2003. doi:10.1109/IPDPS.2003.1213180.

[36] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 155a–155a, April 2005. doi:10.1109/IPDPS.2005.95.

[37] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Design, Automation and Test in Europe*, pages 1226–1231 Vol. 2, March 2005. doi:10.1109/DATE.2005.36.

[38] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. PhaseNoC: TDM scheduling at the virtual-channel level for efficient network traffic isolation. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1090–1095, March 2015. doi:10.7873/DATE.2015.0418.

[39] S. Liu, A. Jantsch, and Z. Lu. Analysis and evaluation of circuit switched noc and packet switched noc. In *2013 Euromicro Conference on Digital System Design*, pages 21–28, Sept 2013. doi:10.1109/DSD.2013.13.

[40] K. Goossens, J. Dielissen, and A. Radulescu. AEthereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers*, 22(5):414–421, Sept 2005.

[41] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 350–355, 2003. doi:10.1109/DATE.2003.1253633.

[42] A. Radulescu, J. Dielissen, S. Pestana, O. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):4–17, 2005. doi:10.1109/TCAD.2004.839493.

[43] E. Kasapaki and J. Sparsø. Argo: A Time-Elastic Time-Division-Multiplexed NOC Using Asynchronous Routers. In *2014 20th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 45–52, May 2014. doi:10.1109/ASYNC.2014.14.

[44] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. Müller, K. Goossens, and J. Sparsø. Argo: A Real-Time Network-on-Chip Architecture With an Efficient GALS Implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(2):479–492, 2016. doi:10.1109/TVLSI.2015.2405614.

[45] E. Kasapaki and J. Sparsø. The argo noc: Combining tdm and gals. In *2015 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, Aug 2015. doi:10.1109/ECCTD.2015.7300101.

[46] R. A. Stefan, A. Molnos, and K. Goossens. daelite: A tdm noc supporting qos, multicast, and fast connection set-up. *IEEE Transactions on Computers*, 63(3):583–594, March 2014. doi:10.1109/TC.2012.117.

[47] A. K. Lusala and J. D. Legat. A hybrid noc combining sdm-tdm based circuit-switching with packet-switching for real-time applications. In *10th IEEE International NEWCAS Conference*, pages 17–20, June 2012. doi:10.1109/NEWCAS.2012.6328945.

[48] F. Pakdaman, A. Mazloumi, and M. Modarressi. Integrated circuit-packet switching noc with efficient circuit setup mechanism. *The Journal of Supercomputing*, 71:2787–2807, 11 2014. doi:10.1007/s11227-014-1337-0.

[49] J. Yin, P. Zhou, S. S. Sapatnekar, and A. Zhai. Energy-Efficient Time-Division Multiplexed Hybrid-Switched NoC for Heterogeneous Multicore Systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 293–303, May 2014.

[50] M. Gorgues Alonso, J. Flich, M. Turki, and D. Bertozzi. A low-latency and flexible tdm noc for strong isolation in security-critical systems. In *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 149–156, 2019. doi:10.1109/MCSoC.2019.00029.

[51] C. Reinbrecht, A. Susin, L. Bossuet, and J. Sepúlveda. Gossip NoC – Avoiding Timing Side-Channel Attacks through Traffic Management. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 601–606, 2016. doi:10.1109/ISVLSI.2016.25.

[52] W. J. Dally and B. P. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2004.

[53] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. Das. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings. 42nd Design Automation Conference, 2005.*, pages 559–564, 2005. doi:10.1109/DAC.2005.193873.

[54] E. Fleury and P. Fraigniaud. A general theory for deadlock avoidance in wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems*, 9(7):626–638, 1998. doi:10.1109/71.707539.

[55] I. Cidon, J. Jaffe, and M. Sidi. Distributed Store-and-Forward Deadlock Detection and Resolution Algorithms. *IEEE Transactions on Communications*, 35(11):1139–1145, 1987. doi:10.1109/TCOM.1987.1096699.

[56] P. Lopez, J. Martinez, and J. Duato. A very efficient distributed deadlock detection mechanism for wormhole networks. In *Proceedings 1998 Fourth International Symposium on High-Performance Computer Architecture*, pages 57–66, 1998. doi:10.1109/HPCA.1998.650546.

[57] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *SIGARCH Comput. Archit. News*, 20(2):278–287, Apr 1992. URL: https://doi.org/10.1145/146628.140384, doi:10.1145/146628.140384.

[58] W. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, 1993. doi:10.1109/71.219761.

[59] Y. H. Song and T. Pinkston. A progressive approach to handling message-dependent deadlock in parallel computer systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):259–275, 2003. doi:10.1109/TPDS.2003.1189584.

[60] A. Hansson, G. Kees, and R. Andrei. Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip. *VLSI Design*, 2007, 04 2007. doi:10.1155/2007/95859.

[61] A. J. Lankes. *A Selective Packet Discard Technique for Efficient Deadlock Recovery in Networks-on-Chip*. Dissertation, Technische Universität München, München, 2015.

[62] A. S. Tanenbaum and D. Wetherall. *Computer Networks*. Prentice Hall, Boston, 5 edition, 2011. URL: https://www.safaribooksonline.com/library/view/computer-networks-fifth/9780133485936/.

[63] M. Vonbun, A. Schiechel, N. A. V. Doan, T. Wild, and A. Herkersdorf. Apec: Improved acknowledgement prioritization through erasure coding in bufferless nocs. In *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '19, New York, NY, USA, 2019. Association for Computing Machinery. URL: https://doi.org/10.1145/3313231.3352366, doi:10.1145/3313231.3352366.

[64] P. Baran. On Distributed Communications Networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, 1964. doi:10.1109/TCOM.1964.1088883.

[65] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of On-Chip Networks Using Deflection Routing. In *Proceedings of the 16th ACM Great Lakes Symposium on VLSI*, GLSVLSI '06, page 296–301, New York, NY, USA, 2006. Association for Computing Machinery. URL: https://doi.org/10.1145/1127908.1127977, doi:10.1145/1127908.1127977.

[66] E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 1126–1127, 2003. doi:10.1109/DATE.2003.1253765.

[67] T. T. Ye, L. Benini, and G. D. Micheli. Packetization and routing analysis of on-chip multiprocessor networks. *Journal of Systems Architecture*, 50(2):81–104, 2004. Special issue on networks on chip. URL: https://www.sciencedirect.com/science/article/pii/S1383762103001425, doi:https://doi.org/10.1016/j.sysarc.2003.07.005.

[68] H. Zhang and D. Ferrari. Rate-controlled static-priority queueing. In *IEEE INFOCOM '93 The Conference on Computer Communications, Proceedings*, pages 227–236 vol.1, 1993. doi:10.1109/INFCOM.1993.253355.

[69] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1396, 1995. doi:10.1109/5.469298.

[70] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen. An event-based network-on-chip monitoring service. In *Proceedings. Ninth IEEE International High-Level Design Validation and Test Workshop (IEEE Cat. No.04EX940)*, pages 149–154, 2004. doi:10.1109/HLDVT.2004.1431260.

[71] J. van den Brand, C. Ciordas, K. Goossens, and T. Basten. Congestion-Controlled Best-Effort Communication for Networks-on-Chip. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2007. doi:10.1109/DATE.2007.364415.

[72] A. Laffely, J. Liang, P. Jain, W. Burleson, and R. Tessier. Adaptive systems on a chip (aSoC) for low-power signal processing. In *Conference Record of Thirty-Fifth Asilomar Conference on Signals, Systems and Computers (Cat.No.01CH37256)*, volume 2, pages 1217–1221 vol.2, 2001. doi:10.1109/ACSSC.2001.987684.

[73] A. Kostrzewa, S. Saidi, and R. Ernst. Dynamic Control for Mixed-Critical Networks-on-Chip. In *2015 IEEE Real-Time Systems Symposium*, pages 317–326, Dec 2015. doi:10.1109/RTSS.2015.37.

[74] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, pages 250–256, 2000. doi:10.1109/DATE.2000.840047.

[75] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2):105–128, 2004. Special issue on networks on chip. URL: https://www.sciencedirect.com/science/article/pii/S1383762103001139, doi:https://doi.org/10.1016/j.sysarc.2003.07.004.

[76] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A Survey of QoS Architectures. *Multimedia Syst.*, 6(3):138–151, may 1998. URL: https://doi-org.eaccess.ub.tum.de/10.1007/s005300050083, doi:10.1007/s005300050083.

[77] A. Radulescu and K. Goossens. *Interconnect and Memory Organization in SOCs for Advanced Set-Top Boxes and TV: Evolutin, Analysis and Trends*, pages 399–423. Springer, Germany, 2004. doi:10.1007/1-4020-7836-6_15.

[78] T. Bjerregaard. *The MANGO clockless network-on-chip: Concepts and implementation.* PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005. Supervised by Assoc. Prof. Jens Sparsø, IMM. URL: http://www2.compute.dtu.dk/pubdb/pubs/4025-full.html.

[79] E. Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proceedings of the IEEE*, 89(5):665–692, 2001. doi:10.1109/5.929649.

[80] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D. Lundqvist. Lowering power consumption in clock by using globally asynchronous locally synchronous design style. In *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, pages 873–878, 1999. doi:10.1109/DAC.1999.782202.

[81] T. Villiger, H. Kaslin, F. Gurkaynak, S. Oetiker, and W. Fichtner. Self-timed ring for globally-asynchronous locally-synchronous systems. In *Ninth International Symposium on Asynchronous Circuits and Systems, 2003. Proceedings.*, pages 141–150, 2003. doi:10.1109/ASYNC.2003.1199174.

[82] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch. Methods for Fault Tolerance in Networks-on-Chip. *ACM Comput. Surv.*, 46(1), Jul 2013. doi:10.1145/2522968.2522976.

[83] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19, 2003. doi:10.1109/MM.2003.1225959.

[84] P. Dodd and L. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on Nuclear Science*, 50(3):583–602, 2003. doi:10.1109/TNS.2003.813129.

[85] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005. doi:10.1109/MM.2005.110.

[86] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, and S. Borkar. Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-/spl mu/m to 90-nm generation. In *IEEE International Electron Devices Meeting 2003*, pages 21.5.1–21.5.4, 2003. doi:10.1109/IEDM.2003.1269336.

[87] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings International Conference on Dependable Systems and Networks*, pages 389–398, 2002. doi:10.1109/DSN.2002.1028924.

[88] M. Zhang and N. Shanbhag. Soft-Error-Rate-Analysis (SERA) Methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2140–2155, 2006. doi:10.1109/TCAD.2005.862738.

[89] M. Cuviello, S. Dey, X. Bai, and Y. Zhao. Fault modeling and simulation for crosstalk in system-on-chip interconnects. In *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, pages 297–303, 1999. doi:10.1109/ICCAD.1999.810665.

[90] S. Sayil, V. K. Boorla, and S. R. Yeddula. Modeling Single Event Crosstalk in Nanometer Technologies. *IEEE Transactions on Nuclear Science*, 58(5):2493–2502, 2011. doi:10.1109/TNS.2011.2165295.

[91] Y. Yang. *Issues of ESD protection in nano-scale CMOS*. PhD thesis, George Mason University, Fairfax, Virginia, USA, 2010.

[92] E. Takeda, C. Y. Yang, and A. Miura-Hamada. Hot-carrier effects in MOS devices. 1995.

[93] J. Keane and C. H. Kim. An odomoeter for CPUs. *IEEE Spectrum*, 48(5):28–33, 2011. doi:10.1109/MSPEC.2011.5753241.

[94] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, S. Wei-kai, S. Sivakumar, G. Taylor, P. VanDerVoorn, and K. Zawadzki. Managing Process Variation in Intel's 45nm CMOS Technology. *Intel Technology Journal*, 12(2):93 – 109, 2008. URL: https://search-ebscohost-com.eaccess.ub.tum.de/login.aspx?direct=true&db=bth&AN=32925829&site=ehost-live.

[95] S. K. Saha. Modeling Process Variability in Scaled CMOS Technology. *IEEE Design & Test of Computers*, 27(2):8–16, 2010. doi:10.1109/MDT.2010.50.

[96] H. Ahmadian, R. Obermaisser, and M. Abuteir. Time-triggered and rate-constrained on-chip communication in mixed-criticality systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 117–124, Sept 2016. doi:10.1109/MCSoC.2016.58.

[97] H. Ahmadian and R. Obermaisser. Temporal Partitioning in Mixed-Criticality NoCs Using Timely Blocking. In *2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 98–105, 2017. doi:10.1109/MCSoC.2017.30.

[98] M. Dridi, S. Rubini, M. Lallali, M. J. S. Florez, F. Singhoff, and J. Diguet. DAS: An Efficient NoC Router for Mixed-Criticality Real-Time Systems. In *2017 IEEE*

International Conference on Computer Design (ICCD), pages 229–232, 2017. doi:10.1109/ICCD.2017.42.

[99] S. Tobuschat and R. Ernst. Providing throughput guarantees in mixed-criticality networks-on-chip. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 292–297, 2017. doi:10.1109/SOCC.2017.8226064.

[100] A. Kostrzewa, R. Ernst, and S. Saidi. Multi-path scheduling for multimedia traffic in safety critical on-chip network. In *2016 14th ACM/IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia)*, pages 1–10, Oct 2016.

[101] A. Kostrzewa, S. Saidi, L. Ecco, and R. Ernst. Ensuring safety and efficiency in networks-on-chip. *Integration, the VLSI Journal*, 58(Supplement C):571 – 582, 2017. URL: http://www.sciencedirect.com/science/article/pii/S0167926016300918, doi:https://doi.org/10.1016/j.vlsi.2016.10.015.

[102] S. Avramenko, S. P. Azad, S. Esposito, B. Niazmand, M. Violante, J. Raik, and M. Jenihhin. QoSinNoC: Analysis of QoS-Aware NoC Architectures for Mixed-Criticality Applications. In *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 67–72, 2018. doi:10.1109/DDECS.2018.00-10.

[103] T. Picornell, J. Flich, C. Hernández, and J. Duato. DCFNoC: A Delayed Conflict-Free Time Division Multiplexing Network on Chip. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[104] S. Wasly, R. Pellizzoni, and N. Kapre. HopliteRT: An efficient FPGA NoC for real-time applications. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 64–71, 2017. doi:10.1109/FPT.2017.8280122.

[105] Y. Ribot González and G. Nelissen. HopliteRT*: Real-Time NoC for FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3650–3661, 2020. doi:10.1109/TCAD.2020.3012748.

[106] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. herwood. SurfNoC: A Low Latency and Provably Non-interfering Approach to Secure Networks-on-chip. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 583–594, New York, NY, USA, 2013. ACM. doi:10.1145/2485922.2485972.

[107] A. Psarras, J. Lee, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos. Phasenoc: Versatile network traffic isolation through tdm-scheduled virtual channels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):844–857, May 2016. doi:10.1109/TCAD.2015.2488490.

[108] K. Goossens and A. Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Design Automation Conference*, pages 306–311, June 2010. doi:10.1145/1837274.1837353.

[109] A. Hansson, M. Subburaman, and K. Goossens. Aelite: A flit-synchronous Network on Chip with composable and predictable services. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 250–255, April 2009. doi:10.1109/DATE.2009.5090666.

[110] R. B. Sorensen, L. Pezzarossa, and J. Sparso. An area-efficient tdm noc supporting reconfiguration for mode changes. In *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 1–4, Aug 2016. doi:10.1109/NOCS.2016.7579324.

[111] T. D. A. Nguyen and A. Kumar. XNoC: A non-intrusive TDM circuit-switched Network-on-Chip. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–11, Aug 2016. doi:10.1109/FPL.2016.7577378.

[112] Y. Chen, E. Matus, and G. P. Fettweis. Register-Exchange Based Connection Allocator for Circuit Switching NoCs. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 559–566, March 2017. doi:10.1109/PDP.2017.14.

[113] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. A. E. Ghany. Survey on real-time networks-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1500–1517, May 2017. doi:10.1109/TPDS.2016.2623619.

[114] A. P. Frantz, M. Cassel, F. L. Kastensmidt, E. Cota, and L. Carro. Crosstalk- and SEU-Aware Networks on Chips. *IEEE Design & Test of Computers*, 24(4):340–350, 2007. doi:10.1109/MDT.2007.128.

[115] A. Eghbal, P. M. Yaghini, H. Pedram, and H. R. Zarandi. Designing fault-tolerant network-on-chip router architecture. *International Journal of Electronics*, 97(10):1181–1192, 2010. doi:10.1080/00207217.2010.512016.

[116] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers*, 22(5):434–442, 2005. doi:10.1109/MDT.2005.104.

[117] B. Fu and P. Ampadu. On Hamming Product Codes With Type-II Hybrid ARQ for On-Chip Interconnects. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(9):2042–2054, 2009. doi:10.1109/TCSI.2009.2026679.

[118] A. Dutta and N. A. Touba. Reliable Network-on-Chip Using a Low Cost Unequal Error Protection Code. In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, pages 3–11, 2007. doi:10.1109/DFT.2007.20.

[119] Q. Yu and P. Ampadu. Adaptive Error Control for NoC Switch-to-Switch Links in a Variable Noise Environment. In *2008 IEEE International Symposium on Defect*

and Fault Tolerance of VLSI Systems, pages 352–360, 2008. doi:10.1109/DFT.2008.40.

[120] Q. Yu, M. Zhang, and P. Ampadu. Exploiting inherent information redundancy to manage transient errors in NoC routing arbitration. In *Proceedings of the Fifth ACM/IEEE International Symposium*, pages 105–112, 2011.

[121] P. P. Pande, A. Ganguly, B. Feero, B. Belzer, and C. Grecu. Design of Low power & Reliable Networks on Chip through joint crosstalk avoidance and forward error correction coding. In *2006 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 466–476, 2006. doi:10.1109/DFT.2006.22.

[122] T. Lehtonen, P. Liljeberg, and J. Plosila. Online Reconfigurable Self-Timed Links for Fault Tolerant NoC. *VLSI Design*, 2007, 05 2007. doi:10.1155/2007/94676.

[123] M. R. Kakoee, V. Bertacco, and L. Benini. ReliNoC: A reliable network for priority-based on-chip communication. In *2011 Design, Automation Test in Europe*, pages 1–6, 2011. doi:10.1109/DATE.2011.5763112.

[124] P. Bogdan, T. Dumitraş, and R. Marculescu. Stochastic Communication: A New Paradigm for Fault-Tolerant Networks-on-Chip. *VLSI Design*, 2007, Apr 2007. URL: https://doi.org/10.1155/2007/95348, doi:10.1155/2007/95348.

[125] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *IEEE Computer Society Annual Symposium on VLSI*, pages 46–51, 2004. doi:10.1109/ISVLSI.2004.1339507.

[126] T. Lehtonen, P. Liljeberg, and J. Plosila. Analysis of Forward Error Correction Methods for Nanoscale Networks-On-Chip. 09 2007. doi:10.4108/ICST.NANONET2007.2035.

[127] S. Murali, D. Atienza, L. Benini, and G. De Micheli. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 845–848, 2006. doi:10.1145/1146909.1147124.

[128] L. Fiorin, L. Micconi, and M. Sami. Design of Fault Tolerant Network Interfaces for NoCs. In *2011 14th Euromicro Conference on Digital System Design*, pages 393–400, 2011. doi:10.1109/DSD.2011.54.

[129] F. K. Koupaei, A. Khademzadeh, and M. Janidarmian. Fault-tolerant application-specific network-on-chip. In *Proceedings of the World Congress on Engineering and Computer Science*, 2011.

[130] C. Grecu, P. Pande, B. Wang, A. Ivanov, and R. Saleh. Methodologies and algorithms for testing switch-based NoC interconnects. In *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, pages 238–246, 2005. doi:10.1109/DFTVS.2005.45.

[131] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. BIST for network-on-chip interconnect infrastructures. In *24th IEEE VLSI Test Symposium*, pages 6 pp.–35, 2006. doi:10.1109/VTS.2006.22.

[132] Z. Zhang, A. Greiner, and M. Benabdenbi. Fully distributed initialization procedure for a 2D-Mesh NoC, including off-line BIST and partial deactivation of faulty components. In *2010 IEEE 16th International On-Line Testing Symposium*, pages 194–196, 2010. doi:10.1109/IOLTS.2010.5560209.

[133] Tang Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Euromicro Symposium on Digital System Design, 2003. Proceedings.*, pages 180–187, Sep. 2003.

[134] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes. Dynamic Task Mapping for MPSoCs. *IEEE Design Test of Computers*, 27(5):26–35, Sep. 2010.

[135] Shih-Shen Lu, Chun-Hsien Lu, and Pao-Ann Hsiung. Congestion- and energy-aware run-time mapping for tile-based network-on-chip architecture. In *IET International Conference on Frontier Computing. Theory, Technologies and Applications*, pages 300–305, Aug 2010.

[136] L. Zhang, J. Yang, C. Xue, Y. Ma, and S. Cao. A two-stage variation-aware task mapping scheme for fault-tolerant multi-core Network-on-Chips. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017.

[137] Z. Lu and A. Jantsch. TDM Virtual-Circuit Configuration for Network-on-Chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(8):1021–1034, Aug 2008. doi:10.1109/TVLSI.2008.2000673.

[138] R. A. Stefan. *Resource Allocation in Time-Division-Multiplexed Networks on Chip*. Dissertation, Technische Universiteit Delft, 2012. URL: http://resolver.tudelft.nl/uuid:63251f1a-8c42-4153-94f4-c85c92a8e950.

[139] Y. Chen, E. Matus, and G. P. Fettweis. Centralized parallel multi-path multi-slot allocation approach for tdm nocs. In *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–5, May 2016. doi:10.1109/CCECE.2016.7726659.

[140] K. Hironaka, N. A. V. Doan, and H. Amano. Towards an Optimized Multi FPGA Architecture with STDM Network: A Preliminary Study. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, pages 142–150. Springer International Publishing, 2018.

[141] ITU-T G.808.1: Generic protection switching – Linear trail and subnetwork protection. Technical report, International Telecommunication Union, May 2014.

[142] ITU-T G.841: Types and characteristics of SDH network protection architectures. Technical report, International Telecommunication Union, Oct 1998.

[143] Y. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, 1996. doi:10.1109/AERO.1996.495891.

[144] E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst. FMEA-based analysis of a Network-on-Chip for mixed-critical systems. In *2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, pages 33–40, 2014. doi:10.1109/NOCS.2014.7008759.

[145] J. Sepúlveda, A. Zankl, D. Flórez, and G. Sigl. Towards Protected MP-SoC Communication for Information Protection against a Malicious NoC. *Procedia Computer Science*, 108:1103–1112, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland. URL: https://www.sciencedirect.com/science/article/pii/S1877050917307068, doi:https://doi.org/10.1016/j.procs.2017.05.139.

[146] Cocotb manual. Accessed: 2021-10-12. URL: https://www.optimsoc.org/.

[147] T. Hollstein, S. P. Azad, T. Kogge, and B. Niazmand. Mixed-criticality NoC partitioning based on the NoCDepend dependability technique. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, June 2015.

[148] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi:10.1109/4235.996017.

[149] OpTiMSoC. Accessed: 2021-04-23. URL: https://www.optimsoc.org/.

[150] OpenRISC. Accessed: 2021-04-23. URL: https://openrisc.io/.

[151] VCU108 Evaluation Kit. Accessed: 2022-03-30. URL: https://www.xilinx.com/products/boards-and-kits/ek-u1-vcu108-g.html.

[152] Open SoC Debug. Accessed: 2022-03-30. URL: https://opensocdebug.readthedocs.io/en/latest/.

[153] P. Wagner. *DiaSys: A Method and Tool for Non-Intrusive Runtime Diagnosis of Embedded Software*. PhD thesis, Technical University of Munich, 2019.

[154] GLIP. Accessed: 2021-11-30. URL: https://www.glip.io/.

[155] J. Sparsø, E. Kasapaki, and M. Schoeberl. An area-efficient network interface for a TDM-based Network-on-Chip. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1044–1047, 2013. doi:10.7873/DATE.2013.217.

[156] M. Ehm. Design and implementation of a network interface supporting remote dma for a fault-tolerant hybrid network on chip. Master's thesis, Chair of Integrated Systems, Technical University of Munich, 10 2020.

[157] M. Bakiri, C. Guyeux, J.-F. Couchot, and A. K. Oudjida. Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses. *Computer Science Review*, 27:135–153, 2018. URL: https://www.sciencedirect.com/science/article/pii/S1574013716302271, doi:https://doi.org/10.1016/j.cosrev.2018.01.002.

[158] K. Tsoi, K. Leung, and P. Leong. Compact FPGA-based true and pseudo random number generators. In *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, pages 51–61, 2003. doi:10.1109/FPGA.2003.1227241.

[159] P. Kohlbrenner and K. Gaj. An Embedded True Random Number Generator for FPGAs. FPGA '04, page 71–78, New York, NY, USA, 2004. Association for Computing Machinery. URL: https://doi.org/10.1145/968280.968292, doi:10.1145/968280.968292.

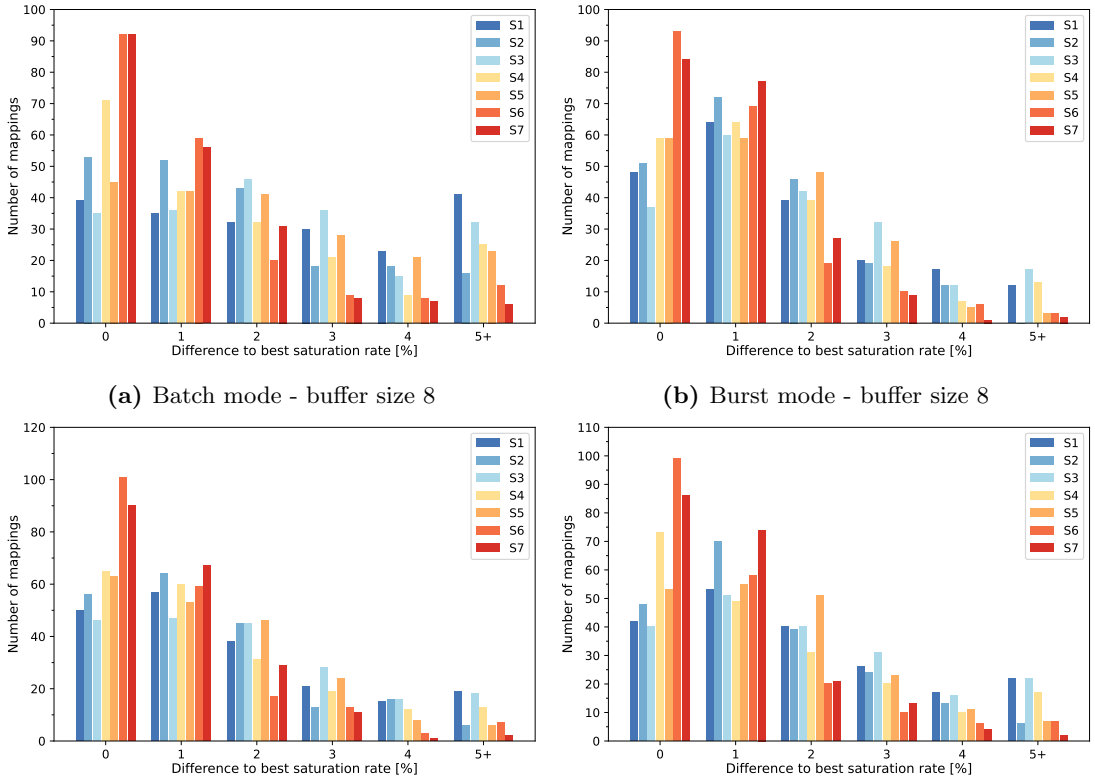
[160] A. Panda, P. Rajput, and B. Shukla. Fpga implementation of 8, 16 and 32 bit lfsr with maximum length feedback polynomial using vhdl. 05 2012. doi:10.1109/CSNT.2012.168.

[161] S. Munder and D. Gavrila. An Experimental Study on Pedestrian Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1863–1868, 2006. doi:10.1109/TPAMI.2006.217.

[162] Daimler Mono Pedestrian Classification Benchmark Dataset. Accessed: 2022-07-27. URL: http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/Daimler_Mono_Ped__Class__Bench/daimler_mono_ped__class__bench.html.

[163] A 4x4 mixed-critical many-core SoC demonstrator system with fault-tolerant NoC. Accessed: 2022-07-27. URL: https://youtu.be/KAo5qaEv-n4.

A Appendix



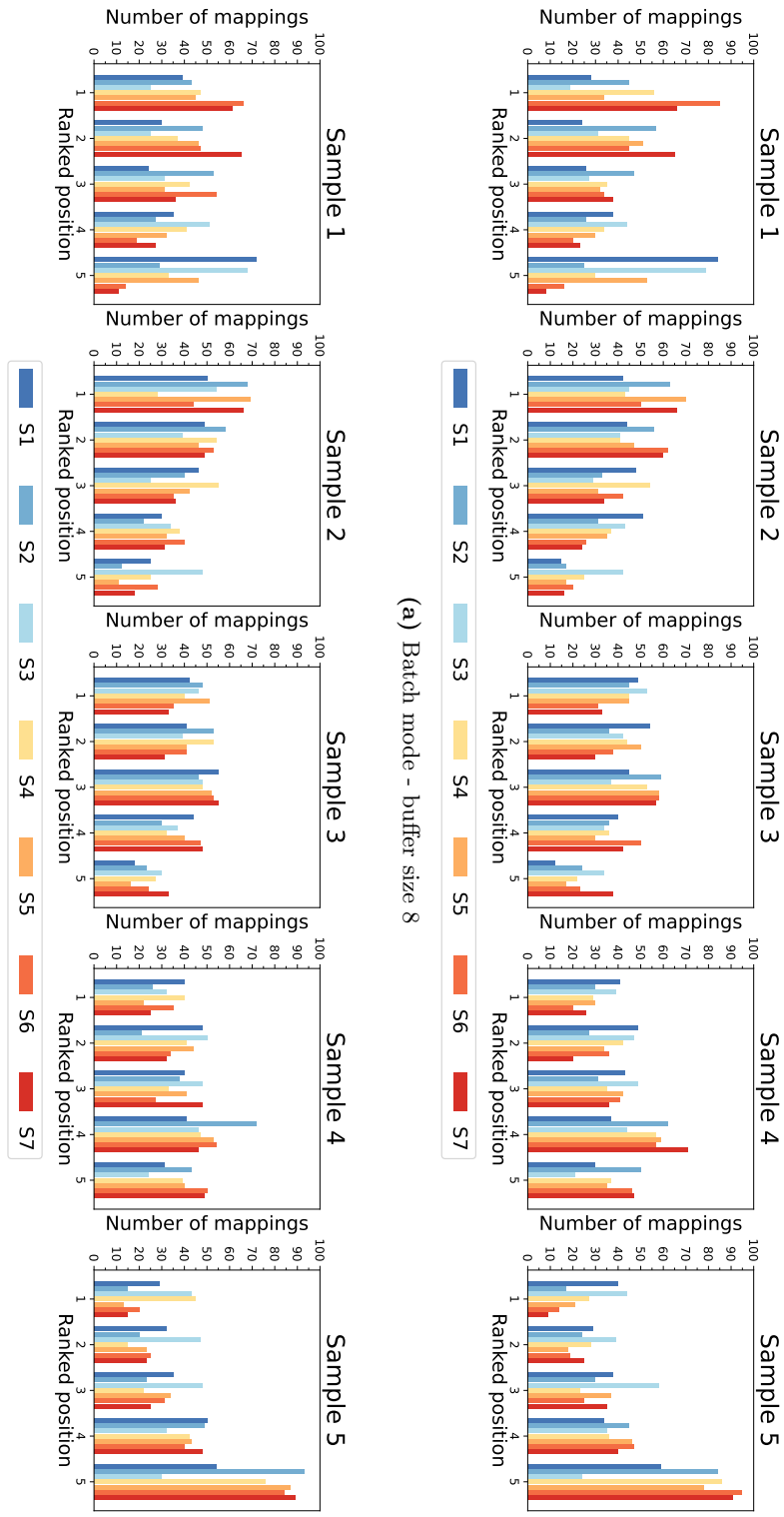
(a) Batch mode - buffer size 8

(b) Burst mode - buffer size 8

(c) Burst mode - buffer size 16

(d) Burst mode - buffer size 32

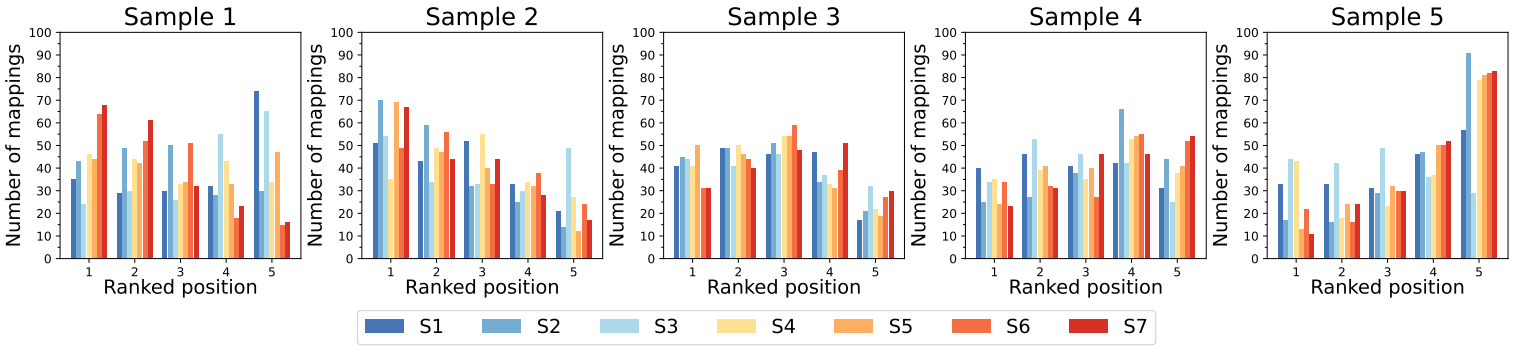
Figure A.1: Overall comparison of 1+1 mapping strategies - all basic system versions



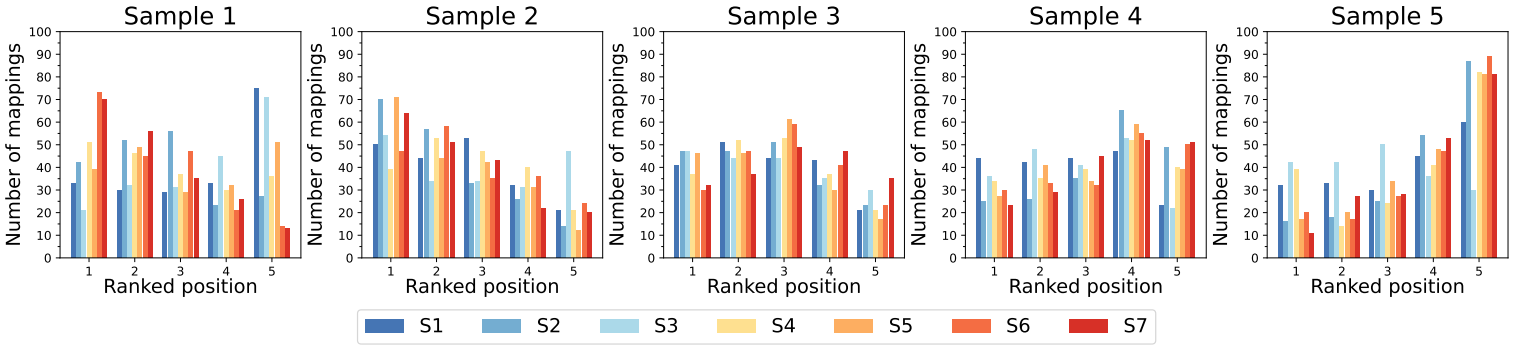
(a) Batch mode - buffer size 8

(b) Burst mode - buffer size 8

Figure A.2: Ranking histograms for 1+1 mapping strategy samples - all basic system versions



(a) Burst mode - buffer size 16



(b) Burst mode - buffer size 32

Figure A.3: Ranking histograms for 1+1 mapping strategy samples - all basic system versions (cont.)

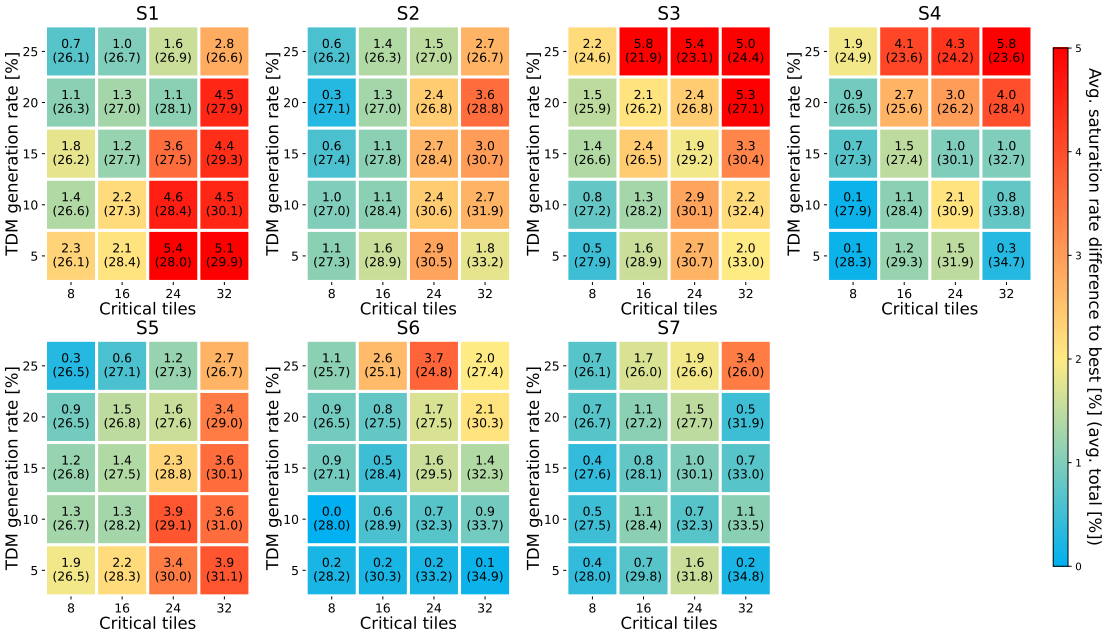


Figure A.4: Detailed comparison of 1+1 mapping strategies - batch mode, buffer size 8

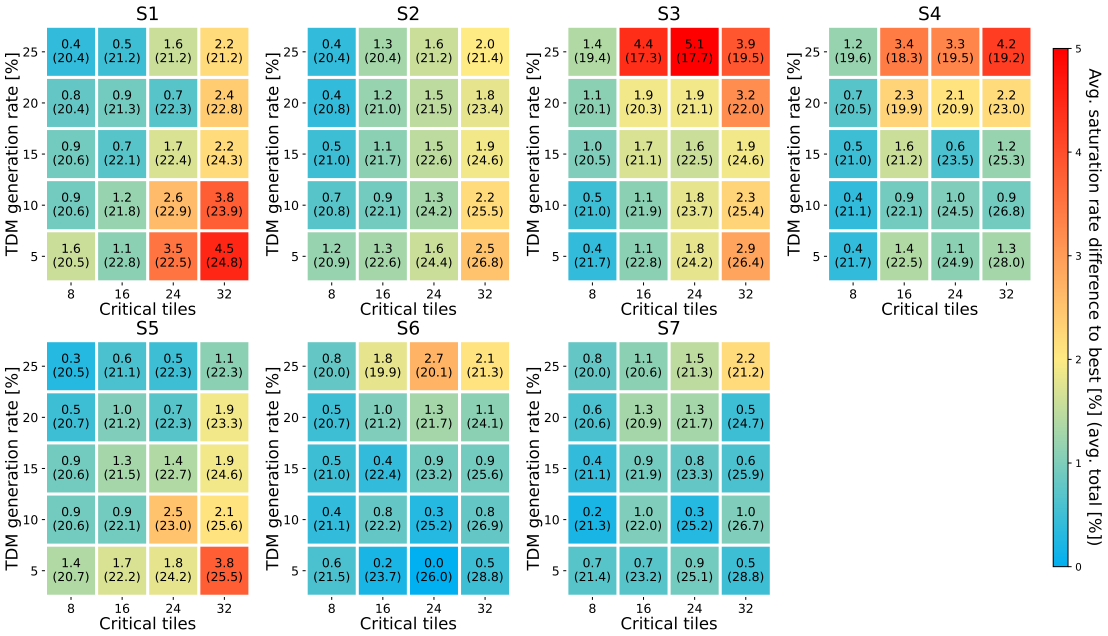


Figure A.5: Detailed comparison of 1+1 mapping strategies - burst mode, buffer size 8

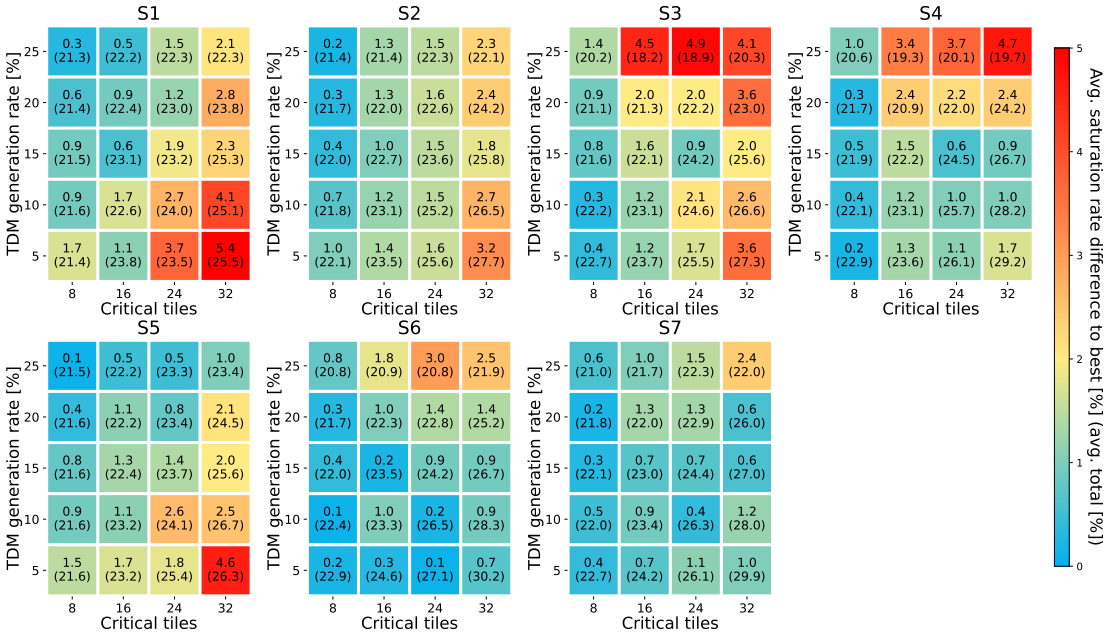


Figure A.6: Detailed comparison of 1+1 mapping strategies - burst mode, buffer size 16

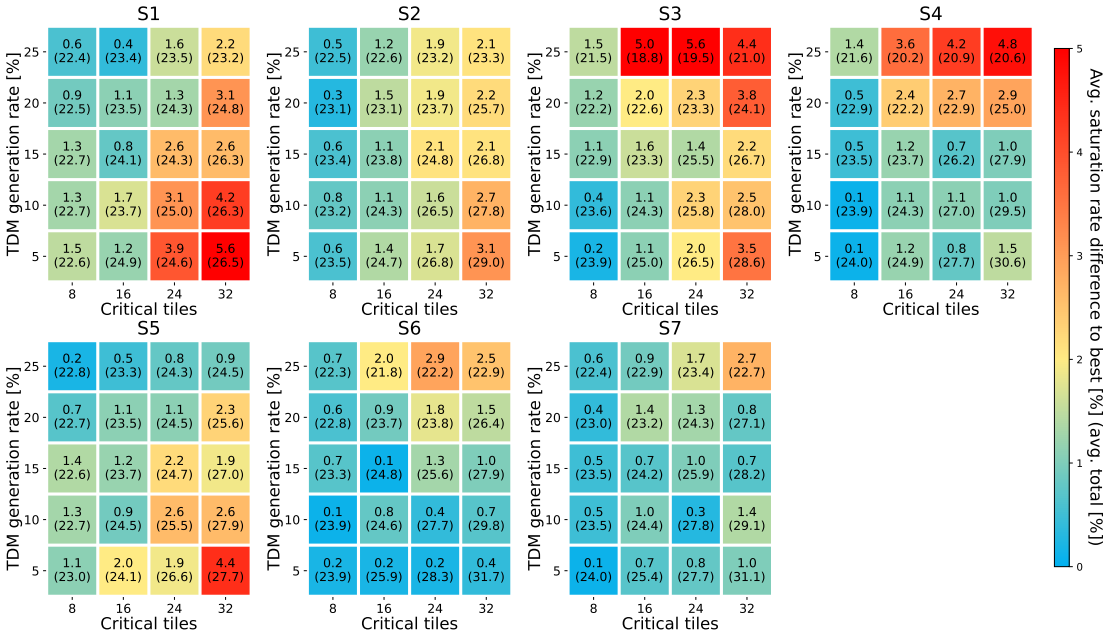
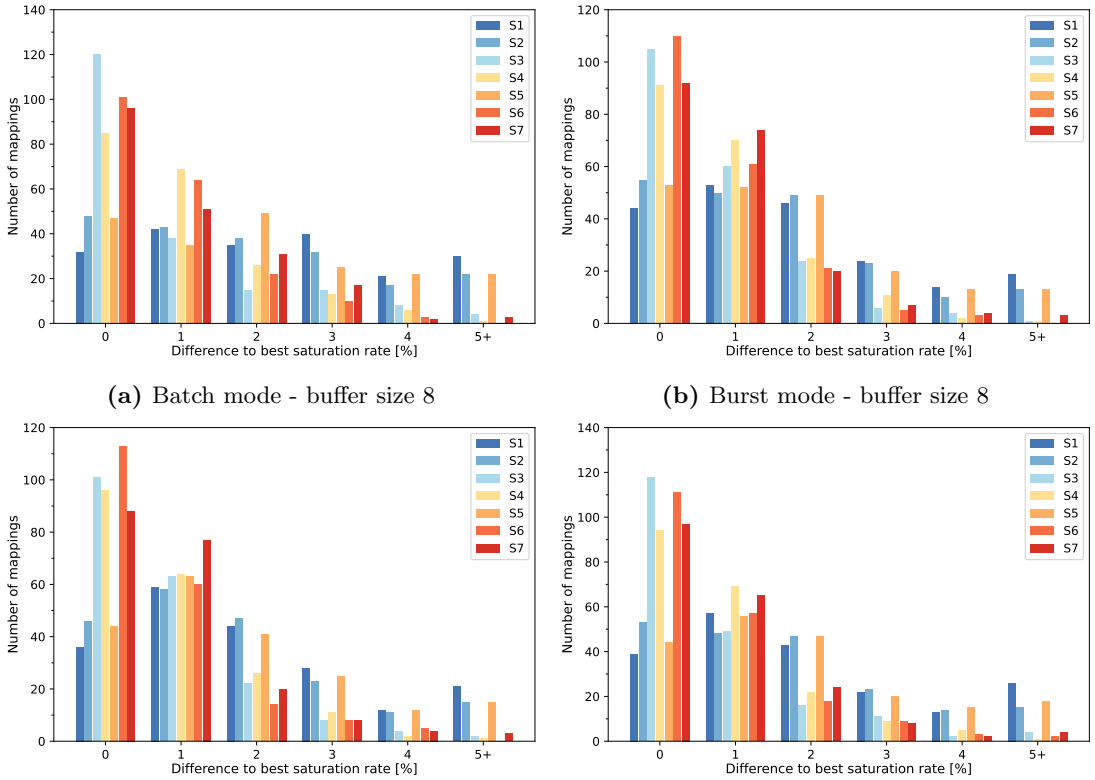


Figure A.7: Detailed comparison of 1+1 mapping strategies - burst mode, buffer size 32



(a) Batch mode - buffer size 8

(b) Burst mode - buffer size 8

(c) Burst mode - buffer size 16

(d) Burst mode - buffer size 32

Figure A.8: Overall comparison of 1:1/1:n mapping strategies - all basic system versions

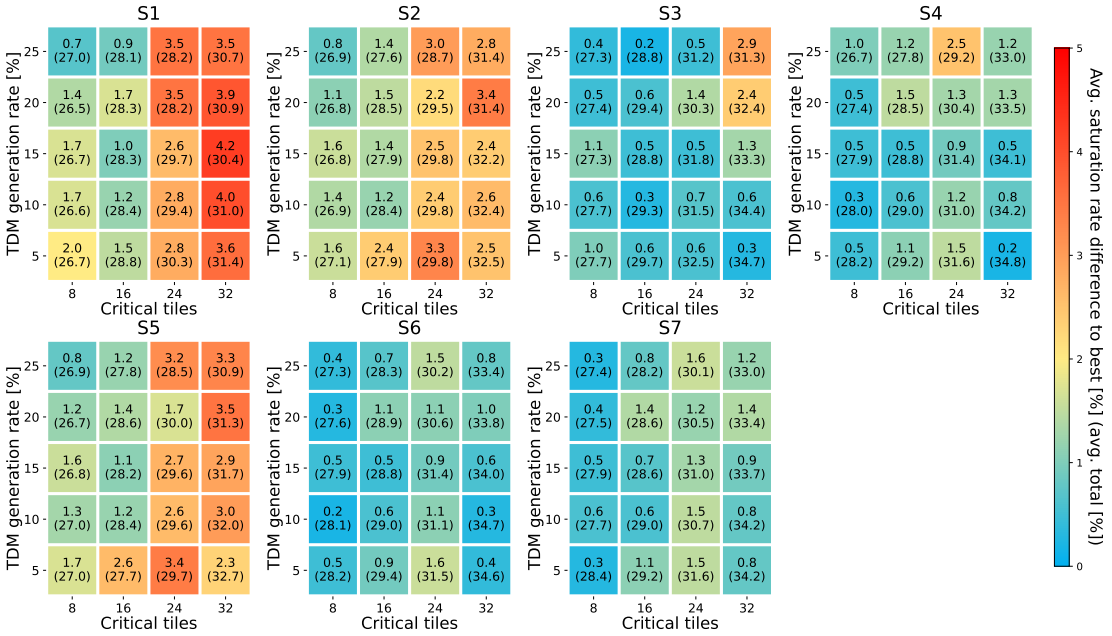


Figure A.9: Detailed comparison of 1:1/1:n mapping strategies - batch mode, buffer size 8

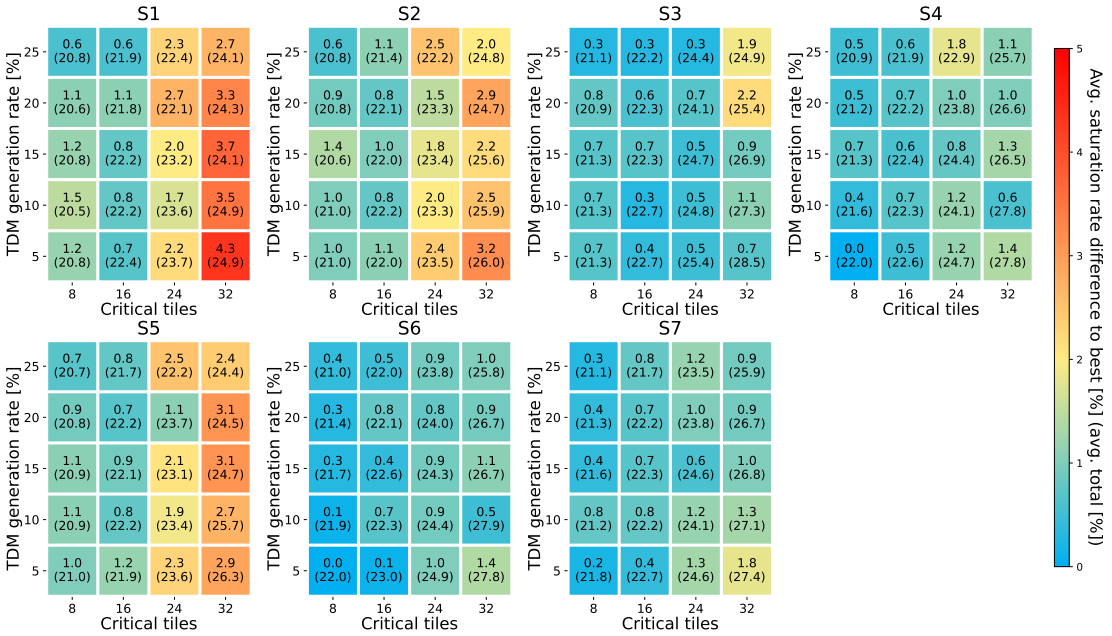


Figure A.10: Detailed comparison of 1:1/1:n mapping strategies - burst mode, buffer size 8

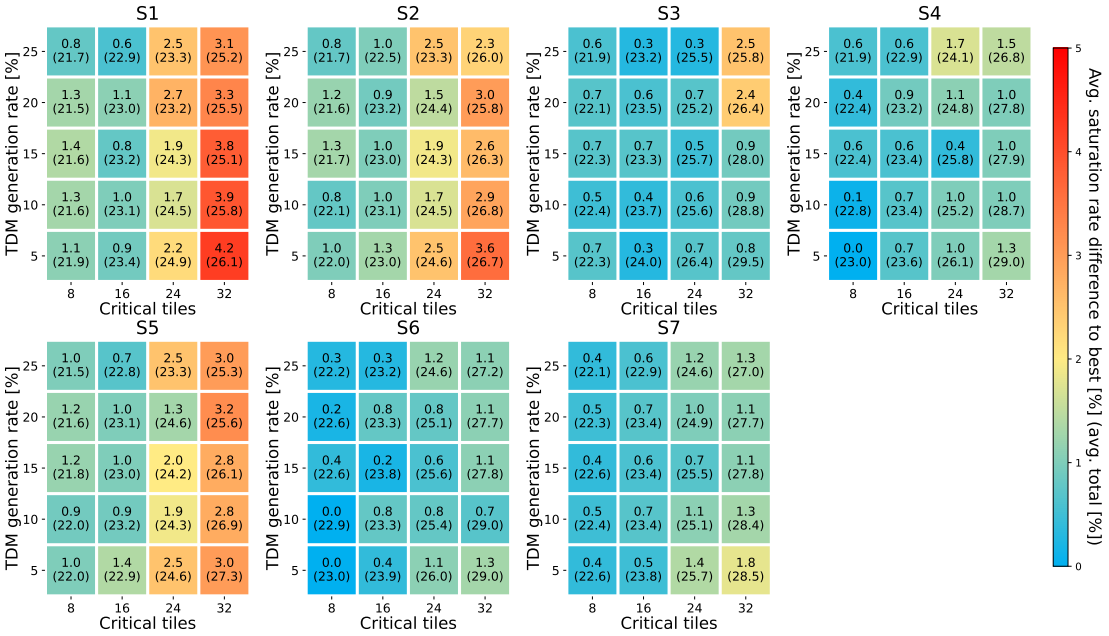


Figure A.11: Detailed comparison of 1:1/1:n mapping strategies - burst mode, buffer size 16

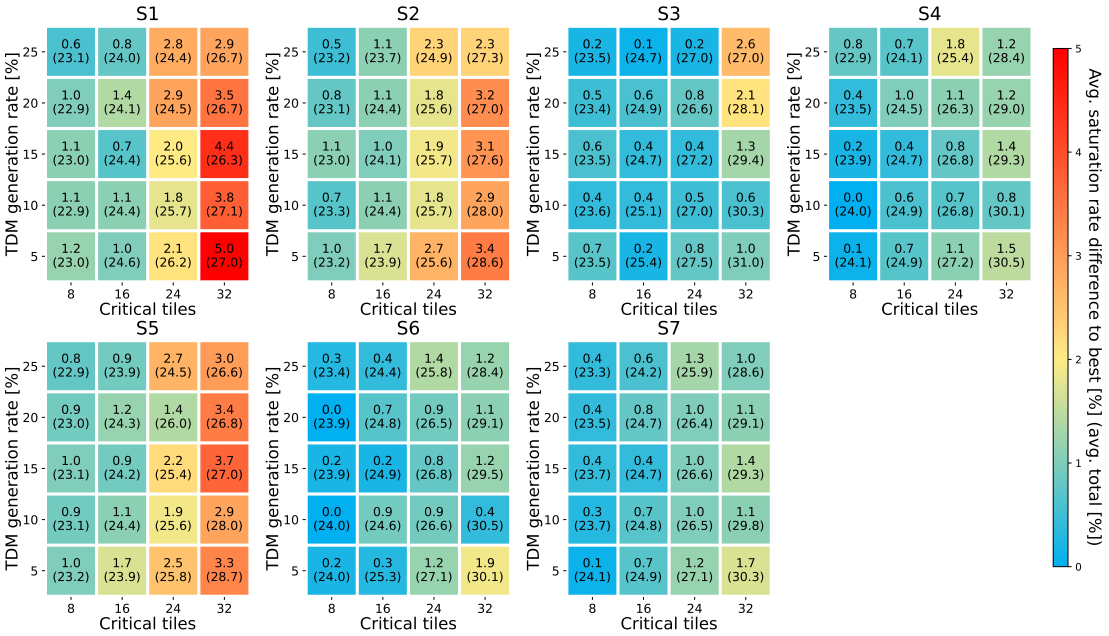
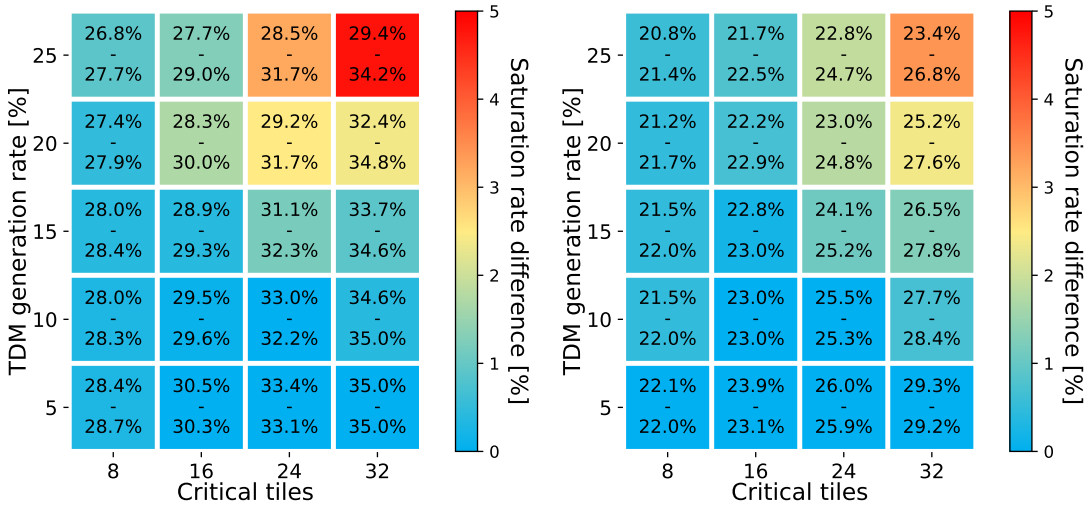


Figure A.12: Detailed comparison of 1:1/1:n mapping strategies - burst mode, buffer size 32



(a) Batch mode - buffer size 8

(b) Burst mode - buffer size 8

(c) Burst mode - buffer size 16

(d) Burst mode - buffer size 32

Figure A.13: Detailed comparison of the protection switching versions - all basic system versions