Monitoring-Aware Network-on-Chip Design

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commisie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 15 december 2008 om 16.00 uur

door

Calin Ciordas

geboren te Oradea, Roemenië

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. H. Corporaal

Copromotor:
dr.ir. A.A. Basten

Printed by: Universiteitsdrukkerij Technische Universiteit Eindhoven
ISBN 978-90-386-1475-5

Monitoring Aware Network-on-Chip Design

Calin Ciordas

Samenstelling promotiecommissie:

Prof. dr. ir. A.C.P.M. Backx	(voorzitter)
Prof. dr. Henk Corporaal	(promotor, TU Eindhoven)
Dr. ir. Twan Basten	(copromotor, TU Eindhoven)
Prof. dr. Kees Goossens	(TU Delft, NXP)
Prof. dr. ir. Ralph Otten	(TU Eindhoven)
Dr. ir. Diederik Verkest	(VU Brussel, IMEC)
Prof. dr. ir. Gerard Smit	(Universiteit Twente)

Printed by: Universiteitsdrukkerij Technische Universiteit Eindhoven
ISBN 978-90-386-1475-5

Summary

We elaborate in this thesis on the benefits and use of monitoring the internal network-on chip (NoC) communication. One salient characteristic of monitoring the internal NoC communication is that it corresponds to monitoring all inter-IP interaction, thus showing a complete picture of what is happening in the chip. The observed behavior can be interpreted in the form of information supporting debugging or in the form of performance information related to interconnect or system on chip supporting Quality of Service techniques. We argue that monitoring has to eventually take the form of a dedicated monitoring service. Our approach is to focus onto a generic monitoring service that has to be part of the NoC itself. The proposed NoC monitoring service is generic and can be instantiated for the monitoring task at hand, for example debugging or performance monitoring. We show the feasibility of our approach via instances for both these application domains.

The monitoring service is composed of multiple, spatially distributed monitors supporting NoC components (routers and network interfaces) but supporting also IP monitors if provided by a third party. The monitors support multiple levels of abstraction and feature a modular design composed of a sniffer, an event generator and a monitoring network interface. The capturing of data in the NoC-based System on Chip is non-intrusive. The NoC monitoring service includes one or more monitoring service access points, which configure the monitors at run-time and receive the monitored data. A centralized or a distributed version of the monitoring service can be instantiated. The monitoring service assumes an event-based model for the monitored data, allowing for on-chip data abstraction.

NoCs are the result of sophisticated design flows. They usually contain several steps: topology generation, mapping of cores to network interface ports, path selection and slot allocation. These steps can be done serially or in an integrated manner. Integration of monitors in the system has an influence on the design flow. To integrate multiple monitors in the design flow, the following steps are proposed: placement of the monitors, dimensioning of their communication requirements and activation. We investigate three possibilities of interconnecting the monitors together with the required design flow modifications. One of the three options, the one using the same NoC for both application data and monitoring data, is

further worked out in detail as it provides the most area-efficient solution. When using the same network for application data and monitoring data, we have two interdependent problems: the one of functional dimensioning of the NoC and mapping of cores while accounting for their communication requirements, and the other of monitor placement and monitoring bandwidth specification. If these two problems are solved sequentially, the monitoring communication requirements can be pre-computed. However, if the communication requirements of the monitors do not fit directly on the generated application NoC, a new NoC must be generated, e.g., by increasing the topology and repeating the process. However, by increasing the topology, the number of NoC routers increases. In turn, the mapping, path selection and allocation of resources may change and the number of required monitors may increase as well (e.g. if probing all routers is required) and their communication requirements may change. Therefore, in the mentioned cases, the monitoring problem must be solved within or at least tightly coupled with the NoC design process. We show that we can solve both problems in synergy, this being beneficial for the NoC monitoring service cost.

Summarizing, our main contributions are: a generic NoC monitoring service, proof of feasibility via two instances of the NoC monitoring service corresponding to two driver application areas (debug, quality of service management), a monitoring-aware NoC design flow able to take into account the monitoring requirements at any step in the NoC design flow, and experimental evaluation and cost quantification of all of the above.

Contents

Acknowledgements

First of all, I would like to thank Twan Basten who gave me the opportunity to do a PhD and to work with him. He was throughout the time my main backer and supporter. I appreciate his guidance, enthusiasm, the many open discussions with him, and of course the not so easy task of proofreading my (actually our) papers. He is a true researcher and I can only hope that part of his research drive has passed to me. He definitely made me wish to follow a research path. Thanks Twan!

I would also like to thank Kees Goossens for giving me the chance to work with the Æthereal project at Philips Research (now NXP Research); it was a great experience. I appreciate his always positive attitude and his trying to see the hidden gems (if any) in the work a PhD student occasionally produces. Thanks are going to the rest of the Æthereal project team, and in particular to Andrei Radulescu.

My special thanks goes to Rian van Gaalen who was always there to help and to provide advice, not only regarding the many hassles one has to face moving into a new country.

Life would have not been that easy at TU/e without the academic coffee breaks; but Amir, Sander, Dominik, Marc, Valentin, me and sometimes other people also, we have done everything in our power to enforce them. It was great!

I thank Jef van Meerbergen for being the link between TU/e and Philips research and continuously supporting my research. I would like to thank Henk Corporaal for taking up the task of being my first promotor when Jef could no longer act as promotor. I know I took a lot of his time, but his comments greatly helped improving this thesis.

I thank Jan Willem, Andreas, Andre and Milan for their contributions in our joint works. I also thank the members of the graduation committee, for reading and commenting on this thesis.

Last but not least, the thanks goes to my family for their permanent and unconditional support ...

Calin Ciordas
December 2008

Chapter 1

Introduction

This introductory chapter opens the way to the main contribution chapters of this thesis, being a mere prelude. It starts with an overview by making an inventory of current practices and trends in the complex embedded world looking at the shrinking technology, innovative networks on chip architectures, the validation dominated electronic system design flows and the emerging applications characteristics. It further details the monitoring problem within this resource constrained world. An innovative communication centric approach to solve the identified problem is briefly described. Based on the mentioned approach the description of this thesis' main contributions including the resulting scientific papers follows. A complete thesis overview concludes this chapter.

1.1 Current Trends

Moore's law, multi-billion transistor ICs, lead towards multi-core, heterogeneous, increasingly programmable SoCs through IP re-use and platform-based design

Ever increasing advances in semiconductor technology in the form of decreasing feature sizes combined with increasing customer demand for more and more functionality have enabled very complex large scale system on a chip (SoC) designs. The design teams are facing now the challenges of integrating the increased functionality on the growing number of transistors.

Currently, design teams are designing large scale integrated circuits (ICs) heavily relying on existing intellectual property (IP) reuse technologies [80], where existing (often third party) IPs are glued together by system integrators in a complex SoC. According to [81], the potential for IP reuse productivity gains is estimated to be at least 200%. One step beyond IP reuse is the concept of platform-based design [54]. Industry has adopted the concepts of platforms (e.g. Nexperia [28] from Philips/NXP, Nomadik [85] from ST, or Omap [90] from TI) where application domains (e.g. multimedia, automotive or mobile terminals) gracefully match

a specific architecture template consisting of reusable groups of cores [81]. In this case, a concrete SoC design comes alive through platform instantiation.

Systems on chip are high-value high-complexity ICs that form the heart and soul of a large diversity of products such as television sets, set-top boxes, and cellular phones. As an example, we can mention the Viper2 set-top box SoC [74, 31] as one instance of the Nexperia multimedia platform [28], with more than 60 different cores and 50 million transistors. As another example, we can mention the SAF7780 [48, 93] as one component in the in-car digital entertainment platform of NXP Semiconductors.

Existing SoC designs integrate more and more IPs and offer increased functionality, inherently resulting in multi-core designs. As SoCs are usually targeted to a specific domain, they usually comprise specialized IPs in the form of co-processors, which can very efficiently solve tasks within the targeted application domain, as well as programmable cores, making the multi-core designs heterogeneous. With the addition of programmable cores (e.g. ARM, MIPS, Trimedia), the multiprocessor SoC designs (MPSoCs) are becoming increasingly programmable.

The need for programmable and scalable interconnects is materialized in NoCs supported by state of the art NoC design flows

With the number of IPs in SoC designs increasing, in traditional architectures encompassing interconnects like busses, communication becomes a bottleneck [10, 82]. The system architecture methodologies also move towards a communication centric design [82]. This shows that the importance of interconnect design increasingly matches the importance of computation subsystems, and the need for a new communication architecture to avoid scalability problems.

Networks on chip (NoCs) [10, 25, 41, 45, 55, 60, 52, 13] have been proposed as a future-proof interconnect. They have emerged as a scalable, structured and modular interconnect solution [10, 82, 25], and tend to become preferred for large scale MPSoCs. NoCs decouple computation from communication supporting IP reuse. With the addition of NoCs, the interconnect sophistication reaches a new level, allowing for increased run-time communication programmability.

NoCs are the result of sophisticated design flows to aid in design time decisions [46, 66, 14, 40]. The NoC design flows are resulting in area-efficient NoC designs while meeting performance and power constraints. These synthesis flows normally consist of several steps like NoC topology generation, mapping of functional IP cores to the NoC, routing and scheduling of IP communication requirements.

Application specific standard products (ASSPs) are a preferred implementation for the ICs which have their functionality highly tuned to the application domain (like chips in TVs, such as Viper). Multi-use case mapping addresses this issue in the NoC context in [65, 64], where each use case resembles one application.

Validation effort of complex systems spans from pre-silicon and the silicon bring-up period to silicon life-time

With the increasing complexity of multi-core, heterogeneous, increasingly programmable SoCs and run-time programmable NoCs, the step of getting the design working properly becomes increasingly difficult. From the total engineering effort of current SoCs, it is estimated that the entire system verification and validation effort consumes up to 70% with the subsequent debug effort accounting for 30-50% [86]. "Design conception and implementation are becoming mere preludes to the main activity of verification" (quoted from [81]).

In the pre-silicon life of complex systems, slow system simulations are an important current practice both in academia and in industry, using for example SystemC [44]. One tendency is to run those simulations at higher levels of abstraction to cope with the speed problem. Simulations are coupled with transaction level models of the system, such as in the Æthereal flit accurate NoC simulator that is coupled with IP cores abstracted by traffic generators [37]. The simulation-based methods are further complemented by the emerging formal [29, 81] and (run-time) assertion-based methods [73]. While these methods arguably result in better quality designs and higher rates of first time right silicon [86] there is a clear need for an in-silicon verification flow.

The quest for better debug solutions points to the need for providing the necessary controllability, and in particular the observability of internal operations of complex SoCs, as it is very difficult to fix what you cannot see [86]. Observability of current SoCs is becoming a major bottleneck as the amount of embedded cores and critical internal signals per I/O pin ratio increases. This has led to the addition of dedicated on-chip resources which support functional analysis in order to increase SoC observability.

Historically, the silicon bring-up problems have been addressed by reusing the industry standard boundary scan-chain infrastructure and the JTAG port for both observability and controllability [101]. Today's designs have to meet strict time to market requirements as well as to reduce the number of redesigns and silicon spins required for a successful IC. As scan-chains alone are not enough [96] to handle this, this must be complemented by a structured debug strategy [100]. This debug strategy spans also to the silicon life-time.

Dedicated debug instrumentation has become common at core level [5], and bus-level [32]. Increasingly, a system-centric debug infrastructure, supporting multi-core system-level diagnosis and analysis, like ARM's Coresight [3] and First Silicon's OCI [56], is gaining momentum. Computation and communication observability in current SoCs are a recognized must, and its importance is growing.

Furthermore, designers have to live with the idea that debugging is still necessary after the chip bring-up period, during the life-time period, due to several reasons. Platform-based design enables the separation of one-time platform design and multiple-times platform instantiation, and the much larger use of the same platform instance. Debugging is required at the platform design, instantia-

tion, as well as at platform use, followed by software debugging for programmable platforms.

Dynamic, complex, scalable and QoS-aware applications from converging application domains are the driving force for SoCs

Modern multimedia applications are emerging. Such applications are becoming very complex as both size and scope of such applications are exploding. One potential factor for the increasing application size is the increasing complexity of continuously added features, for instance, supported features in the multi-window television like picture-in-picture and picture-and-picture. The scope of such application has been broadened from mainly genuine audio-video for simple TV sets to DVD players, recorders, digital radios, automotive, telecommunications or networking.

Furthermore, the convergence of previously unrelated application domains can be witnessed, e.g. looking at the transformation of TV sets into complex multimedia terminals. Another good example to emphasize the convergence of the application domains is the mobile phone. The mobile phone has evolved from a simple communication device to further include a web browser and web services, high resolution photo and video camera, radio, navigator, mp3 player, organizer and other PC-like applications emphasizing also the trend of PC penetration in the consumer electronics market.

With the addition of, for example, object-based MPEG4 applications, the applications in the audio/video domain become very dynamic as the video streams pack video objects instead of frames. Therefore, the processing shifts from (fixed number of) pixels/frame video processing to object-based video processing, where objects can vary in size, shape and contents; the number of objects can vary as well. This means that the required video processing resources are amenable to the input video stream, leading to the need for novel design techniques such as scenario-based design [35].

Users have high expectations about the perceived video and audio quality delivered by multimedia terminals like TVs, or DVD-players. The need for scalable applications is driven by the dynamism and concurrency in applications. Applications must be able to react to and cope with sudden changes in resource requirements and resource availability. Scalable applications can handle changes in need or availability of resources at run-time, allowing for example a graceful degradation or enhancement of perceived quality.

Future SoC designs are becoming so complex that they would need to monitor their performance at run-time; the complete performance validation at design time being out of the question. This would be needed to tune the execution of an application on the given hardware platform and hence the perceived Quality of Service (QoS). This implies the presence of a certain degree of monitoring in the loop, the on-line management of resources being (at least partly) monitoring assisted.

1.2 Monitoring Span

General Monitoring Span

Monitoring has been around us for a very long time. It is the process of collecting information about items. The item is a very general term and can be: a technical system e.g. a router or a set-top box, a physical object e.g. a star or a planet, a being or part of it e.g. the human body or its brain.

There are many orthogonal axes along which monitoring systems can be classified; we only mention four important ones:

- the domain in which the monitoring takes place,
- the form in which the monitoring is achieved,
- the way in which the data is collected from the involved monitors, and
- where the initiative for monitoring lays.

Monitoring has been successfully employed in many domains. In the health care domain patient monitoring systems have been employed for a long time to allow a quick assessment of the physical condition of the patient during e.g. surgery or intensive care. In the energy production domain monitoring has been employed to monitor nuclear power plants, while in the energy distribution domain monitoring has been employed to assess capacity of the transport network at any moment. In the IT domain monitoring has found a second home. One of the places where monitoring proved itself worthy is network monitoring. Network monitoring tools are employed to detect network failures (nodes can be down), network topology changes (users continuously joining), the flows of data in the network, network traffic. They are also employed to detect vulnerabilities in the network on intrusions. Monitoring output is used as input for network management tools, e.g. admission control (allowing new flows of data in the network), balancing traffic over network links, avoiding bad links or nodes, reduce latencies and the like. Set top boxes are remotely monitored to understand how the users are using them, as well as to extract viewing behavior. In the consumer electronics domain, complex ICs in the form of SoCs are monitored during silicon bring-up and during their life to understand the reasons of the field returns.

If we look at the form in which the monitoring is achieved than monitoring can be also classified as hardware, software or hybrid monitoring. In hardware monitoring the monitors are pieces of hardware attached to the items to be monitored. In software monitoring the monitors are simple software routines which are embedded in the software. Hybrid monitors will combine both software and hardware parts. E.g. in network monitoring, the employed monitors can be both hardware or software.

Monitoring systems may incorporate multiple monitors. These monitors can be stand alone, with each monitor operating in isolation and having its monitored

data employed locally; this local employment of monitored data may be done by the monitor itself or by a third party IP. The monitors can also be employed together, with data collected by multiple single monitors which is aggregated to give a global picture. This leads to the centralized, hierarchical or (near) distributed monitoring systems. In a centralized monitoring system all monitors involved are part of a single monitoring domain and all the data is sent to a central point. In a hierarchical monitoring system the overall monitoring domain is partitioned in multiple smaller monitoring domains. Each of these monitoring domains has a central entity which aggregates its data. Aggregated data from multiple of these central entities are again aggregated at a higher level and the process can continue until the monitoring root or the top aggregator is found. As in a hierarchical monitoring system in a distributed monitoring system the overall monitoring domain is split in multiple monitoring domains. The difference is that there is no central aggregator, but multiple aggregators which collect data from one or more monitoring domains.

Looking where the initiative for monitoring lays we can classify monitoring systems in active and passive monitoring systems; in the active monitoring systems the monitor has the initiative in gathering the data it needs while in the passive monitoring systems the monitor expects the data to be readily available to him. As an example for an active monitoring system we can mention latency measures by means of the round trip delay. One packet is sent by the latency monitor via a route and the latency is measured in this way. Opposite to this, a monitor which tracks a link's usage will passively track each packet which passes that link; in this way the initiative does not lay with it. Passive or active monitors can be both realized in hardware and software.

In this work we restrict ourselves to technical systems, in the consumer electronics domain, specifically on-chip monitoring in the NoC-based SoC domain, and we look at passive hardware monitoring in the form of a centralized or distributed monitoring service.

On-chip monitoring span

On-chip performance monitors were the first to be investigated and implemented. [61] proposes the use of a performance monitoring unit (PMU) for the System-On-Chip (SOC) platform for the AMBA AXI bus to give insight to the system designers and analyze the performance bottlenecks of the target system. This monitor is designed to gather the information for the bus performance related metrics such as bus-transaction related events like number of requests, size of the burst transfer request, bus latency for the specific master requests, and amount of memory traffic for specific durations. It can also measure the contention of the bus masters and slaves in the SoC. The authors postulate that the bus transactions are the most interesting to observe and are not observable from outer pins, which prompts the inclusion of on-chip monitors.

One of Intel's former flagship, the Pentium 4 processor also features embedded

performance monitors [84]. It supports 48 event detectors and 18 event counters, capable of counting 18 events concurrently. Because it supports multi-threaded executions it includes qualification of event detection by thread ID and qualification of event counting by thread mode. These monitors are usually embedded in the processor itself and are mainly used for dynamically tuning of applications. An event example is "branch retired" with a mask in the form of branch types: taken, not taken, predicted, and mispredicted; each of these can be counted by a counter.

Another class of on chip monitors are thermal monitors. They have been employed in [15] to support a run-time thermal management strategy in the form of a thermal-aware OS which can employ task migration as a way to control and reduce hot spots. The authors provide no specific details of these thermal monitors or their area cost.

Hardware assertion-checkers have been successfully employed in the verification of large ICs, e.g. in [9]. The authors believe that hardware designers will need to reconcile to the fact that hardware, exactly as software today, will not be shipped bug free. However, they consider that this situation can be acceptably handled with appropriate mechanisms for runtime validation that detect bugs and recover from them when needed. Their proposed mechanism is an on-chip hardware run-time assertion checker, that monitors the design continuously at runtime for purpose of error detection. An assertion is a statement that is part of the design specification in a design specification language that specifies the property which should hold; e.g. any request should be acknowledged in the next cycle. Any deviation from the specification, i.e. invalidation of a property, is considered to be an error. While the authors only consider properties from temporal logic, their work represents a step forward from the traditional design time assertion checking.

Core (IP) monitors have been also employed in complex SoCs for debug purposes [88, 89]. Each computational core in the design is instrumented with a monitor, which is placed between the core and their corresponding network interface. Each core can be debugged in isolation. While the authors do not focus on the area costs of their solution, their work advances the field with the insertion of multiple monitors, where multi-core cross-triggering and global synchronized time stamping is made possible.

Both on-chip performance as well as debug monitors are of interest in this work specifically in the NoC-based SoC domain, and we are not looking at e.g. thermal monitors.

1.3 Motivating Examples

SoC Monitoring (Design for Debug)

In today's chips debugging can be classified either as run-time control or real-time tracing [97]. The former can be considered a proactive approach (monitoring and control) while the latter a passive approach (monitoring only). The former is intrusive in the chip behavior while the latter is not. Both have to be provisioned at design time. They are not exclusive but rather complementary. In our view, real time tracing is a step further from run-time control, which relies on the run-time control as a fall back scenario.

Run-time control relies on setting breakpoints to stop the execution of the chip under debug. The breakpoints are the points in time where the system will stop execution. As there is no prior knowledge when their functionality will be required they are programmable, e.g. when the program counter reaches a certain value. When breakpoints are activated the system stops execution. There are several approaches to stopping the system: stopping the entire system, stopping several cores, using idle modes (called halting) and the like. Stopping large multi-core ICs has proved to be problematic due to multiple clock domains and multi-core aspects; this may result in system states not identical during multiple stops at the same breakpoint in the same conditions. When the system has stopped the system internal state can be examined, e.g via the present scan chains. If the stopping problems are bypassed, it is possible to continue the functional execution in stepping mode. There are various ways to single step ICs depending e.g. on the granularity at which this happens. The steps can be defined in terms of function calls, instructions, transactions, messages, data words, or clock cycles.

Real-time tracing (monitoring) relies on bringing internal signals out of the chip on a set of pins. A variety of solutions exists for making these pins available for monitoring: directly on the device pins, or using a separate debug interface, be it low or high speed, serial or parallel. Captured signals can be internally kept in a trace buffer or immediately made available to the output pins. The signals to be captured are a design time choice and can span over a fixed choice (hardwired solution) or a programmable solution, when the signals to be made available can be selected at runtime from a predefined design time set.

The tenet of debugging, with a direct impact on monitoring, is that it is not known in advance what kind of errors are expected. Several examples of real errors and how the engineers have fixed these errors in industry-grade SoCs by either using run-time control, real-time tracing or their combination, are presented in [97]; these examples reflect today's semiconductor industry practices. Note that in the five IC examples presented in this paper three feature real-time tracing. Two error cases, one employing run-time control and the other real-time tracing, literary taken from this paper are shown here for motivational purposes. Both cases would have not been solved without the presence of the monitoring hardware.

"The scan-based silicon debug feature helped the designers diagnose a video synchronization problem. This problem occurred only after approximately 50 to 100 input video frames 1 to 2 seconds of real-time video processing. For no obvious reason, the image would disappear from a TV monitor connected to the chips video output. Before tape-out, the SoCs entire timing-back-annotated netlist had been simulated, but only for the first five video frames, because of the very long simulation runtimes. None of these video frames showed any error, leading to the conclusion that the design was error free. Nevertheless, the silicon failed at bring-up. After repeated state dumps during the SoCs initialization sequence, and after examining the on-chip bus transactions in detail, the designers discovered that the internal ROM from which the SoC was booting contained a programming error. An incorrect value in one of the video output processors configuration registers caused the output processor to eventually lose synchronization with the input stream." (quoted from [97], debugging Philips's Co-processor Array IC, during silicon bring-up) Note that this example directly points to the need of monitoring (bus) transactions.

"Analyzing the GPIO state machine. Engineers also used the real-time trace infrastructure to monitor the state machine in the general-purpose I/O module, to learn why it was not correctly responding to external triggers. Observing that the state machines state did not change even in the clear presence of external triggers, engineers looked elsewhere for the problem. They more closely examined the external-trigger source selection, leading to the discovery of a programming error. By fixing this error, they showed a correctly functioning state machine, which transitioned correctly when external triggers were applied." (quoted from [97], debugging NXP's PNX8525 and Codec IC, during first silicon bring-up and early development of several embedded-software applications)

1.4 Problem Statement

We need NoC monitoring, and this is difficult/complex

One of the most vexing problems is the run-time SoC monitoring problem in the context of NoC-based SoCs. The general observability problem exists for current and future SoCs. Driven by debugging (as part of the validation track) and QoS/performance reasons, there is a growing need to observe (monitor) what is going on or happening in today's SoCs. This information can be utilized on-chip or off-chip to understand what the system is doing, to detect a functional error or a performance bottleneck, or simply to improve the on-chip resource management. The general observability problem is accentuated by the fact that the only access points to the SoC internals are a limited number of pins, e.g. the NEXUS debug port [92], and by the increase in the number of functional cores per pin ratio. Furthermore, current SoC designs incorporate multiple (heterogeneous) programmable cores and interconnect (NoC), adding to the overall complexity.

NoC monitoring has to be solved in synergy with the NoC design

The decision on the presence, form and integration of NoC monitoring has to be taken at NoC design time, and not after the NoC or NoC-based SoC has already been designed. As NoCs are the result of sophisticated design flows there is a further need to solve the above mentioned monitoring problem not in isolation but in synergy with NoC design.

All implications of NoC monitoring must be analyzed and quantified

All the monitoring infrastructure and design efforts for realizing and integrating it in the NoC designs come at a cost, and have a potential implication on the resulting SoC. Therefore, the implications and costs of monitoring for NoC-based SoCs have to be evaluated and known, as the presence of monitoring (e.g. for debugging) in future SoCs is amenable to the will of SoC architects. The tenet of NoC monitoring is to prove that it is feasible in the resource constrained SoC environment and that it is cost effective; e.g. area overhead should not be substantial.

1.5 Approach

Our approach to alleviate the run-time monitoring problem of NoC-based SoCs is the use of *Communication-Centric Monitoring* to counterbalance computation-centric monitoring of SoCs historically emphasized so far in the research community. We elaborate in this thesis on the benefits and use of monitoring the internal NoC communication.

One salient characteristic of monitoring the internal NoC communication is that it corresponds to monitoring all inter-IP interaction, thus showing a complete picture of what is happening in the chip. The observed behavior can be interpreted in the form of information supporting debugging, or in the form of performance information related to the performance of the interconnect or the SoC supporting Quality of Service techniques.

In the spirit of *Communication-Centric Monitoring*, we argue that monitoring has to eventually take the form of a dedicated monitoring service. Our approach is to focus onto a *generic monitoring service* that has to be part of the NoC itself, and is composed of multiple, spatially distributed monitors supporting NoC components (routers and network interfaces), but supporting also IP monitors if provided by a third party. The monitors support multiple levels of abstraction enabling debugging or performance analysis.

1.6 State of the Art and Related Work

Tackling a subject which combines the recent NoC hype with the already existing monitoring problem we can look at general monitoring, monitoring for busses, debug, transaction monitoring, NoC related work and their combinations. A part of this related work generally referring to the current trends and in particular to the validation track has been introduced in Section 1.1, this section only completes the picture with the missing parts.

There has been a lot of work towards run-time observability or monitoring and towards NoCs, but little on the combination of the two. As NoCs are treated in a separate chapter, Chapter 2, the NoC related work is not further detailed here, but only briefly mentioned. The test and verification implications of using NoCs have been inventoried in [98]. Currently, in the NoC research community, focus is on the design [55, 60, 40], analysis [37, 76] and use [42, 67] of NoCs.

A lot af work has also been done for monitoring busses; ARM's Coresight [3] technology combines ETMs [5] for ARM cores, with the AHB Trace Macrocell which gives visibility on AMBA AHB busses. First Silicon's on-chip instrumentation technology (OCI), provides on-chip logic analyzers [32] for AMBA AHB, OCP, and Sonics SiliconBackplane bus systems. These allow the user to run-time capture bus activity, and in a multi-core embedded debug system [56] they can be combined with in-system analyzers for cores, e.g. for MIPS cores.

Summarizing, although state-of-the-art monitoring bus solutions exist, they are not able to cope with a NoC-based SoC, and in NoCs the focus has not been on the monitoring. More related work is treated in each of the following contribution chapters.

1.7 Contribution

At the time this work started there was no NoC monitoring support. With the addition of our work we have managed to provide it in the form of a NoC monitoring service which has introduced the main NoC monitoring concepts. Thanks to our monitoring service, internal NoC information is made available at run-time. This provides the knobs and switches for assisting debugging at different levels of abstraction up to the transaction level, or to the understanding of the system behavior at run-time, allowing to take it into account for QoS management. Our monitoring service is proven in two distinct cases (debugging and QoS) and we have automated the instantiation of it with the help of an associated monitoring-aware NoC design flow.

Summarizing, our main contributions are:

1. *a generic NoC monitoring service,*
2. *proof of feasibility via two instances of the NoC monitoring service corresponding to two driver application areas (debugging and QoS management)*

3. *a monitoring-aware NoC design flow able to take into account the monitoring requirements at any step in the NoC design flow,*

4. *experimental evaluation and cost quantification of all of the above.*

Generic NoC monitoring service

The NoC monitoring service (NoCMS) allows the run-time observability of data in the NoC at different levels of abstraction in a non-intrusive way. The service is run-time accessible and reconfigurable.

The proposed NoCMS is generic and can be instantiated for the monitoring task at hand. The monitoring service consists of hardware probes which can be attached to NoC components (routers and network interfaces) or to IPs. The hardware probes feature a modular design composed of a Sniffer, an Event Generator (EG), and a Monitoring Network Interface (MNI). The capturing of data in the NoC-based SoC is non-intrusive. The probes feature a programming model allowing all or part of them to start in sync.

The NoCMS includes one or more Monitoring Service Access points (MSAs), which configure the probes at run-time and receive the monitored data. A centralized or a distributed version of the NoCMS can be instantiated. The NoCMS assumes an event-based model for the monitored data, allowing for on-chip data abstraction.

Due to the flexibility and richness of NoCs, we also exploit new possibilities of pipelining or combining monitors, or optimizing communication of the monitors with an MSA. These relate to the (Philips Research and NXP) patent applications [20, 39, 38, 21].

Two NoCMS instances.

To prove the concepts and evaluate the monitoring service, we propose two instances of the service corresponding to the two main drivers of monitoring: debugging and run-time performance (Quality of Service) instances.

The **debug instance** attacks the monitoring problem by focusing at the transaction-level. IPs in a SoC interact by means of transactions, e.g. a write transaction. An event generator (EG) capable to ultimately reconstruct (and abstract) transactions at run-time is proposed, bypassing the problems caused by the non-alignment of packets and messages in the NoC. This monitor is able to work at four levels of abstraction, programmable at run-time. The EG can itself be instantiated for supporting one or more of the abstraction levels.

In the **Quality of Service instance** the focus is on the performance monitoring. An EG capable of following the network utilization is proposed. It can be configured at run-time to track the number of flits, payload words or headers (for up to 10 router links). This EG can be employed in a performance NoCMS, via a third part use case, in order to support a new congestion control BE service.

Monitoring awareness in the NoC design flow.

NoCs are the result of sophisticated design flows. They usually contain several steps: topology generation, mapping of cores to NI ports, path selection and slot allocation. These steps can be done serially [40] as in the majority of NoC design flows or in an integrated manner like in UMARS [46]. Integration of monitors in the system has an influence on the design flow.

Multiple monitoring probes are part of the NoCMS. To integrate them in the design flow, the following steps are proposed: placement of the monitors, dimensioning of their communication requirements and activation. We investigate three possibilities of interconnecting the monitors. For each of the proposed solutions, the required design flow modifications are investigated. One of the three options, the one using the same NoC for both user data and monitoring data is further worked out in detail as it provides the most area-efficient solution.

When using the same network for application data and monitoring data, we have two interdependent problems: the one of functional dimensioning of the NoC and mapping of cores while accounting for their communication requirements, and the other of monitor placement and monitoring bandwidth specification. If these two problems are solved sequentially, the monitoring communication requirements can be pre-computed. If the communication requirements of the monitors do not fit directly on the generated application NoC, a new NoC must be generated, e.g., by increasing the topology and repeating the process. However, by increasing the topology, the number of NoC routers increases. In turn, the mapping, path selection and allocation of resources may change and the number of required monitoring probes may increase as well (e.g. if probing all routers is required) and their communication requirements may change. Therefore, in the mentioned cases, the monitoring problem must be solved within or at least tightly coupled with the NoC design process. We show that we can solve both problems in synergy, this being beneficial for the NoCMS cost.

Experimental evaluation and cost quantification.

We have evaluated hardware area cost and traffic for the mentioned cases, taking into account the area of the probes involved (e.g., a transaction monitor), all the aspects of the NoC monitoring service (e.g., an extra monitoring network interface for the monitoring service access point), the monitoring instance at hand (e.g., at least one transaction monitor must be present on the path of a data stream) and the NoC design flow (e.g., the original NoC topology). Placement strategies for monitors are investigated. We have investigated the total configuration time for the NoCMS in several scenarios.

The monitors alone come at a low area cost: $0.026mm^2$ for the transaction monitor [1] and $0.016mm^2$ for the performance monitor for an arity five router [2],

[1] This area number corresponds to a fully configurable monitor for debugging, able to decode transactions from the data stream, but without further abstraction capabilities.

[2] This area number corresponds to a fully configurable monitor for performance analysis, able

both in CMOS12. The results show that the total area cost of a complete NoCMS remains below 20-25% of the total NoC area.

1.8 Thesis Overview

Chapter 2 introduces the main concepts of NoCs in general, on which this work is based, makes an inventory of existing NoCs with their shared features and details the Æthereal NoC in particular, as this was used as a test-bench for this work.

Chapter 3 presents the generic NoC monitoring service. It starts with the concepts of the monitoring service, followed by the details of the generic probe architecture together with its associated event model and programming model. All these are completed by a monitoring traffic management approach and the (model of) monitoring service cost. This chapter is based on publications [17, 18].

Chapter 4 presents the instance of the NoCMS for transaction monitoring. It starts with motivating the need for transaction monitoring. The underlying architecture is detailed, presenting all intermediate layers of monitoring up to transaction monitoring, which is made possible for virtually all known packetization schemes. All this is exemplified for the Æthereal NoC. This chapter is based on publication [19].

Chapter 5 presents the instance of the NoC monitoring service for QoS. After the introduction of the performance monitor characteristics, a link is made with the arising scalable applications. The proposed service is evaluated for the controlled BE service case study. This chapter is based on the author's contributions from papers [94, 71].

Chapter 6 shows the impact of the NoC monitoring service on design flows for ASICs. Three alternative interconnect options for interconnecting the monitoring probes are proposed and evaluated. This is completed by the proposal of a monitoring aware NoC design flow for one alternative which is able to automatically account for monitoring at NoC design time. This chapter is based on publications [22, 23, 24].

Chapter 7 summarizes the main results of this work and the remaining open issues. It also features a short visionary look into the future of the presented ideas.

to count words, packet headers or payload words.

Chapter 2

Networks on Chip Preliminaries

After many years of designing SoCs around point to point interconnects and busses, a new interconnect has been proposed: networks on chip (NoCs). This chapter presents a bare minimum introduction to NoCs, such that the following contribution chapters come natural to the reader. The focus is on understanding the main concepts of NoCs and the motivation behind them. It touches the NoC building blocks, NoC topologies, as well as the NoC-based system architectures. A brief inventory of existing NoCs with their associated shared or distinctive features is made. The Æthereal NoC is used as an example throughout this work and it is therefore detailed together with its associated design flow. The entire chapter elaborates on and is reduced in the conclusions to the common features which are shared by the majority of NoCs, as a solid basis for a generic NoC monitoring solution.

2.1 Basic Concepts

Networks on Chip

Networks on chip have recently received considerable attention as an emerging future-proof interconnect. Many NoCs [10, 25, 41, 45, 60, 52, 13] have already been proposed in academia as well as in the industry.

Networks on chip have emerged as a modular, scalable (they scale better than busses), future-proof SoC interconnect and tend to become the preferred interconnect solution for large scale inherently multiprocessor SoCs. They enable IP reuse and structuring of the design process by decoupling computation from communication and by offering well defined interfaces.

In general, NoCs are composed of network interfaces (NI), which implement the NoC interface to IPs, and of routers which transport packets of data from NI to NI. Note that the literature sometimes refers to NoCs as the entire SoC

including the connected IPs (cores). To avoid the ambiguity, in this thesis, we refer to the interconnect only (NIs plus the routers) as the NoC.

Many references are available for NoC details, as mentioned throughout this work, but in general the reader may refer to [30] for the more general interconnection networks details and to [27] for the more NoC specific things.

Network Interfaces and Routers

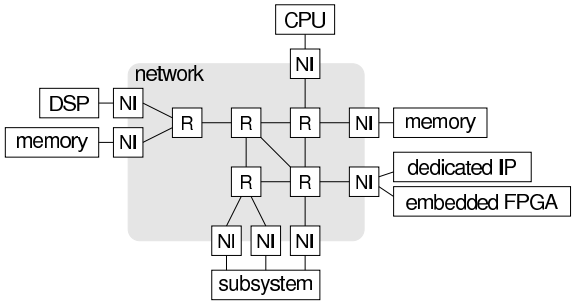


Figure 2.1: Example NoC

The two NoC components, routers (R) and network interfaces (NIs), are depicted in Figure 2.1, together with example IPs connected to NIs. These IPs can be genuine CPUs or DSPs, but also memories, dedicated (hardwired) IPs, embedded FPGAs or complete subsystems. The routers can be connected among themselves and to the NIs.

The routers flip data between NIs and are bound together and to network interfaces by links. The links are in general bidirectional and are implemented as a set of two unidirectional links. Note that in principle there can be multiple links between the same pair of routers but only one between a router and one NI.

The network interfaces implement the NoC interface to IPs. They enable end-to-end services [78] to the IP modules and are key in decoupling computation from communication [13, 102]. The NI allows the designer to simplify communication issues to local point-to-point transactions at IP module boundaries, using protocols natural to the IP [102]. In general they convert IP level communication (reads and writes) into packets which the NoC can handle. They are responsible for (de-)packetization, for implementing the connections and services, and for offering a standard interface (e.g., AXI or OCP) to the IP modules.

NoC Topologies

Many network topologies have already been investigated in the context of regular networks, e.g. [30]. The most simple classification comprises regular and irregular

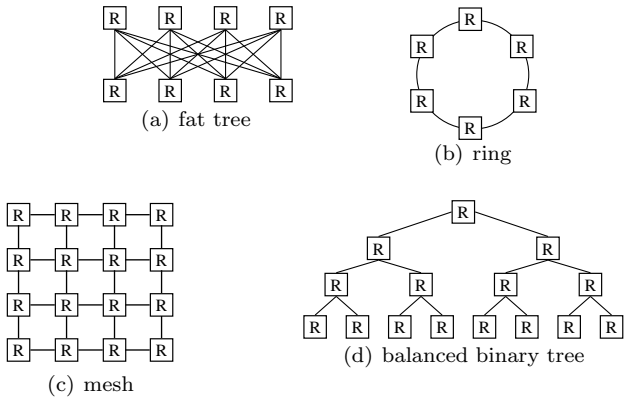


Figure 2.2: NoC regular topologies

topologies. The regular topologies group all topologies that show a regular pattern in the connection of network components, and the irregular groups the ones that show no regular pattern.

A few regular NoC topologies, like a ring, mesh, tree or fat tree are graphically presented in Figure 2.2. Note that topologies mainly refer to the relative positions of routers and not to the relation between routers and NIs, or the number of IPs connected to a single NI, a relation which is detailed later in this chapter. For the sake of comparison, Figure 2.1 with the example NoC features an irregular topology. Regular topologies exhibit characteristics desirable in the design of large scale ICs. E.g. the mesh topology leads to a high link utilization and is good for floorplanning while the tree-like topologies are very well suited for designs that exploit locality of traffic.

For NoCs in general, there are in principle no topology constraints except that the supported topologies are mainly planar topologies, which can easily be implemented in silicon. There are also no guarantees that all the proposed NoCs support random topologies or even some of the most common regular topologies. Some NoCs are not limited to a specific topology like the Æthereal NoC [40], but other NoCs may actually be limited to a specific topology, e.g. the SPIN NoC [45] assumes the fat tree topology.

IP-NI-R

One interesting dimension is the one which relates the IPs, NIs and routers. Regardless of the use of regular or irregular (custom) topologies in the NoC design, one relevant classification of NoCs from the monitoring point of view is the one taking into account the number of IPs they support per NI, and the number of supported NIs per router. Visually depicted in Figure 2.3, using as a basis the

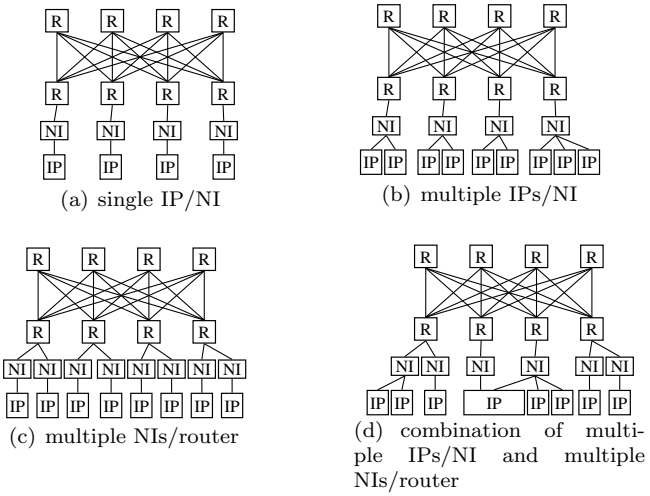


Figure 2.3: Various IP-NI-R relationships

previously presented fat tree topology from Figure 2.2(a), this classification is very important for monitoring decisions like where to place the monitors. Simple NoCs allow a single NI attached to a router with a corresponding single IP attached to the mentioned NI as shown in Figure 2.3(a). Closely related is the fact whether there are NIs attached to all routers or not. Due to the nature of the fat tree topology (indirect topology), in Figure 2.3(a), only half of the routers have NIs attached to them. One example where all routers may have NIs attached to them is the mesh topology, see e.g. Figure 2.4.

Multiple (but not necessarily the same number of) IPs may be connected to the same NI, as in Figure 2.3(b). From the monitoring point of view, this means that a monitor added to that particular network interface may be able to monitor the communication of any of the connected IPs. The clustering of multiple IPs per NI is done to aggregate the bandwidth required by these IPs to the NI-R link in order to obtain a high link utilization, meaning that the silicon area is efficiently used. It is also done in order to keep the paths between the IPs that communicate a lot together short in order to reduce the power consumption. The clustering of multiple NIs per router as in Figure 2.3(c) is actually done for the same reasons. Clustering more (but not necessarily the same number of) NIs per router means that a monitor attached to that specific router may potentially monitor the communication of all or any of IPs connected to its connected NIs. There exists also the case when a single IP connects to multiple NIs, e.g. a multiport memory, having each port connected to one NI; however; however from the monitoring point of view each port can be viewed as a different IP.

Table 2.1 presents the possible combinations of options. The most complex

Table 2.1: IP-NI-R

IPs per NI	NIs per R	topology
one	one	regular
one	multiple	regular
multiple	multiple	regular
one	one	irregular
one	multiple	irregular
multiple	multiple	irregular

cases are the NoCs which will potentially cluster multiple NIs per router with potentially multiple IPs per NI in a regular or irregular topology. As another interesting option, an IP with multiple ports, e.g., a multiport memory, may connect to more than one NI. Such a case with a regular topology is shown in Figure 2.3(d).

Routing and Switching

Communication between IPs takes place in the form of transaction components called messages, which in NoCs are split into packets. As part of the communication between a sender IP and a receiver IP and their corresponding source and destination NIs (the place where the IPs connect to NoC), routing is the way of selecting a desired path as a sequence of routers which will deliver the data. Routing is one of the main NoC characteristics and influences primarily the performance of the NoC and ultimately its final area and (dynamic) power required.

Both static and dynamic routing have been employed the NoCs. In static routing, the path from the source to the destination is pre-computed and the packet follows a fixed route. In dynamic routing, only the source and destination pair are known and the packets belonging to the same communication may follow different routes based on routing decisions taken at individual routers based on the current status of the network, an individual router or of a group of routers.

Switching refers to the way in which inputs and outputs of a router connect together, and the way the information belonging to a source destination pair actually travels through the NoC. We can distinguish two main classes of switching techniques: packet switching and circuit switching. In circuit switching, a circuit through potentially multiple routers in the NoC, corresponding to the desired routing path, is established prior to the communication start, by reserving physical links. The entire circuit is therefore released after the communication takes place. Virtual circuits make a step forward by multiplexing circuits on links.

In packet switching, the information to be communicated is partitioned into packets which contain routing information in the first bits, usually referred to as the packet header. These packets are routed from source to destination in a time-space scheme. Unlike circuit switching no path has to be reserved. Two

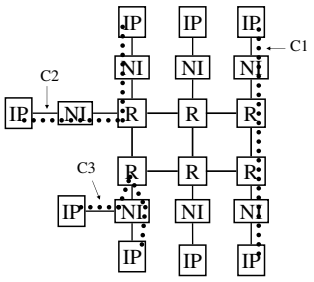


Figure 2.4: Example of NoC communication options

variations of packet switching employed in NoCs are wormhole and virtual cut-through switching, with the former being the switching of choice in the NoC world. Both are based on the principle of pipelining the communication through the NoC at flit level avoiding the expensive buffering of complete messages at routers, the main difference being the level of granularity at which they operate, the former at a finer level (flit) than the latter (packet). In wormhole switching when receiving a flit the router tries to send it to the next router; when this succeeds the next flits follow in a pipeline fashion, if ir does not succeed the message is blocked. In virtual cut-through switching if the flit cannot be sent to the next router, the rest of the packet is received and buffered locally; if the flit can be sent then the rest follow him. While in principle both try to avoid buffering entire messages, only the former does it always, while the latter may actually buffer them in the worst case scenario at high network loads. Combinations of multiple switching techniques within single NoCs exist and are beneficial for NoCs, e.g., the combination of circuit and packet switching in the Æthereal [41] and Mango [12] NoCs.

Communication options

Interesting enough, and related to the previously presented NoC topologies and the relation IP-NI-R, there are multiple options for the type of the path of a connection that may exist in a certain NoC. With our communication centric monitoring approach understanding these options is crucial. The type determines how the path relates to the number of NIs and routers it traverses, which is important from the monitoring point of view. Note that here we only refer to point-to-point connections (which support the communication between two IPs), and not to, e.g., multicast connections. The three main options are depicted in Figure 2.4.

The first option, connection C1 in the figure, is the most common option and is present and supported in basically all of the existing NoCs. It adheres to the restriction that a connection has to pass through two different NIs, and at least

two routers.

The second option, option C2 in the figure, enables a connection to pass through one router and two different NIs. This option only applies to NoCs which support multiple NIs per router.

Finally, the least restrictive option of all, the third option, connection C3 in the figure, restricts the length of the path to at least one router and two different NI ports, which can be on the same NIs. Obviously this option only applies to NoCs which support multiple IPs per NI. The support for any of these types of connection must be built in the corresponding NoC design flow, and has an impact on the monitoring support.

NoC services

One of the main strengths of NoCs is the decoupling of the computation from communication. This issue is key in the service based design of SoCs, as the communication and the computation parts can now be designed in isolation.

This decoupling is done by means of NoC services. NoC services abstract from the NoC internals, hence belonging to the transport layer. These services are offered to the NoC connected IPs. These IPs are refining their communication onto the offered services. In order to communicate with each other, these IPs do not know and do not make assumptions about the implementation of the NoC which stands between them, and solely rely on the offered transport layer services.

The offered services can be grouped as guaranteed or best effort services, and connection-oriented or connectionless services. Connection-oriented or connection-based services requires an end to end connection to be set up before any communication between two NoC connected IPs at the ends of this connection can take place; it guarantees that the transmitted data arrives in the proper order at the receiver. Connectionless services do not require the prior establishment of such a connection for communication; the sender starts to send data to the receiver; in general there is no guaranteed that data will arrive and that it will arrive without duplication, without delays, and in sequence.

In guaranteed services (GS), in its best known form of guaranteed throughput (GT), e.g. as employed in the Æthereal NoC, the fixed bandwidth allocated for the communication (of a higher level session) does not change dynamically during the time of the session, regardless the state of the system. The GS is maintained by means of appropriate NoC resource reservations for the entire duration of the session. Time division multiple access (TDMA) is generally employed in NoCs to provide time and bandwidth-related guarantees, with TDMA circuits corresponding to NoC connections.

In best effort (BE) services or non-guaranteed services, the bandwidth allocation for a specific communication may change dynamically subject to e.g. the bandwidth availability in the NoC. These services can be perturbed by congestion, existing or new guaranteed services used by the NoC connected IPs, as well as by other best-effort services.

In fact, the guaranteed services can be leveraged on and hence are often associated with the connection-oriented services. Connectionless services do not usually offer much guarantees and are rather associated with the best effort services.

Connection-based flow control, also known as end to end flow control, is required to prevent the overflow of the buffers at NIs. If flow control is not implemented, then the NIs can drop packets when buffers overflow. In NoCs, a credit-based flow control is usually employed, where credits are associated with empty buffer space at the slave (receiving) NIs. The credit for the slave NI is kept at the master (sending) NI, and is increased when data is consumed from the slave NI buffer and lowered when data is sent to the buffer.

NoC design flows

With their inherent complexity, NoCs require sophisticated tools to aid in design-time decisions [46, 67]. These tools are generally referred to as NoC design flows. The typical NoC design flow [40, 67, 14] is normally split into four steps as shown in Figure 2.5: topology selection, mapping, path selection and slot allocation. Each step adheres to the decisions taken in the previous steps.

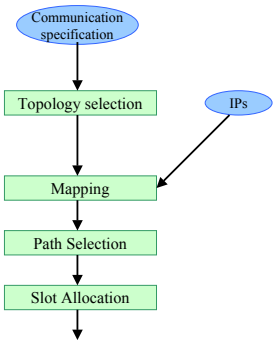


Figure 2.5: General NoC Design Flow

As prerequisites for NoC design, communication requirements must be derived, and the set of IPs to be connected to the NoC must be specified. In the topology selection step, the router network together with the bordering NIs are generated, based on the previously derived communication requirements. Note that even NoCs that have no architectural topology constraints can be restricted in the use (automated generation) of a specific topology by its associated design flow. Using this topology together with the IP specification, the binding of IP ports to NI ports is done in the mapping step. These two initial steps are done taking into account the supported IP-NI-R relations, as previously presented.

In the path selection step, paths are allocated for all the communication flows specified, taking into account which of the previously presented communication

options are supported. In the slot allocation step each of the flows gets its own TDMA time slots for the traversed NoC links. Some design flows may omit or combine various steps.

NoC examples

As previously mentioned, many NoCs have been already proposed, both in the academia and in the industry. Here, we briefly outline a few example of existing NoCs with their main features:

Æthereal NoC [41, 40]. The Æthereal NoC is a connection-oriented packet-switched network on chip, wit routers using input queuing and link level flow control. It offers a unique blend of guaranteed throughput (GT) and best-effort (BE) services. It allows multiple IPs per NI, and multiple NIs per router, exposing the most general solution from the investigated NoCs. In the remainder of this work we use the Æthereal NoC as an example for our work. Therefore, the relevant details for this work of this particular NoC are detailed in Section 2.3.

MANGO [12]. The message passing asynchronous network on chip providing guaranteed services over the OCP (Open Core Protocol) interface (MANGO) was developed in academia. It's main characteristic is that it is a clockless NoC. It offers NIs to convert the data from the connected OCP compliant IPs. It offers a combination of best effort and guaranteed services. It provides only a one to one correspondence between IPs and NIs, resulting in the fact that each communication path will pass two NIs and at least two routers.

SPIN [45]. The scalable, programmable, integrated network (SPIN) is one of the pioneers of the on-chip packet-switched network. This NoC is clearly associated with the fat tree topology. It uses wormhole routing. Each terminal fat tree router (e.g. the four routers with NIs attached in Figure 2.3(a)) accommodates four terminals in SPIN terminology. Each terminal consist of an NI in the form of VCI (Virtual Component Interface) wrapper with its corresponding IP. It can only accommodate one IP per NI.

xPIPES [50, 68]. This work refers to the combination of the SUNMAP/xPipes network on chip design methodology. Besides the genuine NoC work, this work focuses on robustness, reliability and error correction. It supports a range of topologies like mesh, torus, hypercube and butterfly. It combines wormhole routing with source routing and offers connectionless best effort services. It requires one NI per each IP involved, but it can connect multiple NIs per router. The NI supports OCP protocol.

Arteris [7]. This a one of the first commercially available NoC, and comes from a French company. Their focus is on producing commercial designs and they were between the first in the industry to believe that NoCs will soon replace busses [8]. The Arteris NoC is a packet switched interconnect. It supports a range of topologies including user specified topologies. It also supports the OCP and

AMBA IP interfaces in the form of NI wrappers. It offers transport level services in the form of best effort traffic. The network interfaces employed by the Arteris NoC are basic ones which only allow the connection of a single IP per NI.

Chameleon [83, 53]. This packet-switched network with virtual channels uses source routing. It offers both BE and GT services; the GT services are based on the virtual channel allocations while the BE is made available by sharing virtual channels. The destination processing element and the route to be followed are specified in the packet header. The Chameleon NoC supports only a regular mesh topology, combining each processing element with a single NI and a single router. Thus, it provides a one to one correspondence between IP and NI as well as between the NI and R. It uses a single PE as a configuration manager, to configure the other PEs, the network and communication channels by means of control messages.

2.2 General NoC-based Architectures

The routers and NIs, with their associated NoC topologies, routing and switching schemes and design flows are mere parts of the more general NoC system architectures. While, many classifications of these architectures have already been presented the one we found most inspiring and closest to our opinion was presented in [16]. We further summarize the main characteristics of the two main NoC based architectures.

Application Specific NoCs

The application specific NoCs assume that one or more applications are known upfront. This class of architectures encompass the application specific ICs (ASICs) and application-specific standard products (ASSPs).

ASICs are designed for a single application while ASSPs may support several application instances being targeted to an application domain. The fact that the applications are known at design time makes it easy to predict their communication requirements and therefore the corresponding network usage patterns before the NoC is actually designed. A minimal NoC supporting the application(s) is synthesized by the corresponding NoC design flow. Therefore, application specific NoC design is of utmost importance, and these designs are usually area and power efficient.

In general the class of application specific NoCs will cluster multiple IPs per NI, multiple NIs per router and may employ regular or custom NoC topologies, with all possible communication options as presented in section 2.1. This is because in this case all the information about the application is known.

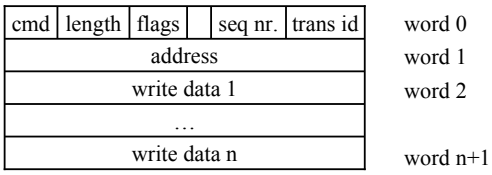


Figure 2.6: Æthereal write message format

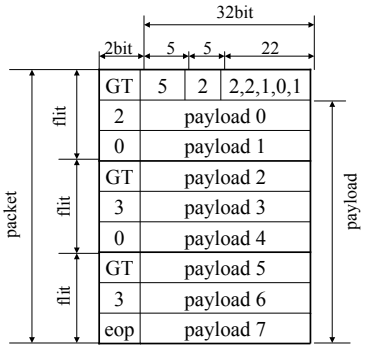


Figure 2.7: Æthereal packet example

General Purpose NoCs

Also called chip multi processor NoCs, they capture the general purpose aspects of NoCs. They are not built to support a specific application or application domain but are built to serve as a platform on which various applications, known after the platform was built, are mapped.

The application communication requirements are not known at design time. Therefore, a minimum cost NoC cannot be designed. As a result, the mapping of an application onto this platform and routing of communication connections are the most important steps, the corresponding traffic pattern being known only at run-time. The NoC design step merely consist of a topology choice in this case.

In general the class of general purpose NoCs employs a one to one mapping between IPs and network interfaces in a mesh topology of routers, supporting the first communication option as presented in Section 2.1. The resulting components, IP with the corresponding NI or even including their corresponding router are in general called tiles. Variations though exist.

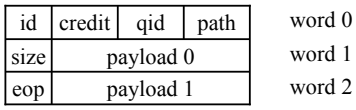


Figure 2.8: Æthereal flit format

2.3 TheÆthereal NoC

Implementation Details

The Æthereal NoC [41, 40] is a connection-oriented packet-switched network. The Æthereal routers use input queuing and link level flow control. The Æthereal network interfaces have a modular design, composed of an NI kernel and NI shells. The NI kernel provides the basic NI functionality, packetization of messages, arbitration between connections, and end-to-end flow control. The NI shells implement protocol adapters for protocols such as AXI or DTL and additional functionality, such as multicast and narrowcast connections, which are not always needed in a design. Multicast and narrowcast are connections between a single master and multiple slaves; while for the former all slaves receive a transmission from the master, for the latter only one slave will receive it. These shells can be plugged in or left out at design time in order to optimize area cost.

The Æthereal NoC supports the entire range of topologies from the regular ones to custom ones, not being bound to a specific one. Furthermore, it allows clustering of multiple IPs per NI and potentially multiple NIs per router with any topology. It also supports all three communication options as presented in Section 2.1.

Running at 500 MHz, the Æthereal NoC offers a raw link bandwidth of 2GB/s in a 0.13μm CMOS technology. Æthereal offers transport-layer communication services to IPs, in the form of connections, comprising best-effort (BE) and guaranteed-throughput (GT) services. Guarantees are obtained by means of TDMA slot reservations in NIs.

Æthereal NoC instances are reconfigurable at run-time. This is achieved by programming the NIs using standard memory-mapped I/O ports. The current setup uses centralized programming of the NoC and source routing, but distributed solutions are also possible.

The interconnected IPs interact with each other by means of transactions, which are read and write transactions from IPs. In principle one transaction may comprise one or optionally more messages. From the IP perspective these transactions (reads and writes) are performed on connections, consisting of one request and one response channel. The paths of request and response channels may be different. Messages are differentiated as request and response messages. E.g., a request message can be the write message depicted in Figure 2.6. A response message is for example data coming back as a result of a read operation, or an

acknowledgment as a result of a write operation. More details on transactions in general and the Æthereal NoC transactions in particular together with examples are presented in Section 4.3.

The NIs convert these messages into packets, by chopping them into pieces of a maximum length and adding a header to each of these pieces. Packets may be of different lengths. An example packet is shown in Figure 2.7. The packet header consists of:

(1) a path to the destination NI as a sequence of router output ports,

(2) a queue identifier (qid) at the destination NI, and

(3) piggybacked credits for end to end flow control.

Packets are further split into flits, the minimum transfer or flow-control unit between hops. The flit format is presented in Figure 2.8, with the mention that this particular flit contains a packet header as the first word. One flit corresponds to one TDMA slot. Note that the packet presented in Figure 2.7 corresponds to three consecutively allocated slots. One flit comprises three 32-bit words. For each of the three words there are two bits of sideband information. The first two sideband information bits, id in Figure 2.8, show whether the flit is BE or GT and whether it contains a packet header or not. The second two bits show the number of valid words in the flit. The last two bits indicate the end of packet.

Connections have properties such as data integrity, transaction ordering or flow control. Data integrity guarantees that the data is not altered in the NoC. Transaction ordering guarantees that the order of separate communications is preserved per connection. Connection flow control guarantees that the data that is sent will fit in the buffers at the receiving end, to prevent data loss and network congestion.

The Æthereal NoC uses wormhole routing which requires less memory because its storage unit is a flit rather than a packet. Packets are partitioned in flits. A flit is passed to the next router when that router indicates it accepts that flit. As soon as a flit of a packet is sent over an output port, that output port is reserved for flits of that packet only. Therefore, a packet can be spread over a number of routers.

Corresponding Design Flow(s)

The basic ideas of the Æthereal NoC design flow have been presented in [40]. It targets application specific NoC designs and fits the four step sequential NoC design flow profile previously presented relying on topology selection, mapping, path selection and slot allocation. As shown by [37], the Æthereal NoC design flow supports a wide range of topologies including meshes, trees and fat-trees. It also supports all possible communication options of Section 2.1. As output, it can generate synthesizable VHDL or a flit accurate SystemC simulator.

Further upgrades to this design flow in the form of a combination of its steps achieving better run-time and lower area cost of the final designs has resulted in UMARS [46]. UMARS is a QoS constrained NoC design algorithm. It unifies the three resource allocation phases: spatial mapping of cores, spatial routing of communication, and the restricted form of temporal mapping that assigns time-slots to these routes. UMARS considers the real-time communication requirements, and guarantees that application constraints on bandwidth and latency are met.

UMARS is a greedy algorithm, iterating over the monotonically decreasing set of unallocated channels until they are all accommodated in the NoC, or until allocation failed. The algorithm, as outlined in Algorithm 2.3.1, never backtracks to reevaluate an already allocated flow, enabling run-times in the order of milliseconds.

Algorithm 2.3.1 Outer loop of UMARS

1. While there are unallocated channels
   (a) Select the channel with highest bandwidth
   (b) Find a mapping and a path for the selected channel
   (c) Select slots on this path

Channels are allocated ordered on their bandwidth requirements. This is done as it:

(1) helps in reducing bandwidth fragmentation [59];

(2) is important from an energy consumption and resource conservation perspective since the benefits of a shorter path grow with communication demands [49];

(3) gives precedence to flows with a more limited set of possible paths [49]. This ordering assures us that no channel succeeding the one currently being allocated has higher bandwidth requirements.

2.4 Conclusions

NoCs are becoming the interconnect of choice in the embedded world due to their scalability and to the decoupling of the communication from the computation. With many NoCs already proposed and a few which have reached maturity, they support complex and varied topologies. Different relations between the number of IPs per NI and the numbers of NIs per router together with different communication options may be supported in the current NoCs.

The Æthereal NoC, which is used throughout this work as an example, is a very rich NoC in the sense that it supports a wide range of topologies and that it

allows all IP-NI-R relations and communication options. One of the key issues in the NIs, the packetization was detailed, together with the associated state of the art NoC design flow required as support for real designs.

Besides being a mere introduction to NoCs, this chapter shows that common features exist for the majority of NoCs. It is important to observe that despite implementation details or particular NoC cases most NoCs share the features on which our NoC monitoring service is built. We further summarize these common features:

(1) Routers are the basic data switch unit and are connected by links. Network interfaces connect and interface the IPs to the router network. Concentrating the entire SoC communication, the routers and NIs make ideal points for monitoring.

(2) IPs are connected to the NoC via NIs. The main process that happens at the insertion of IP data in the NoC, via the NI, is the packetization, and splitting of these packets in flits. This implies that extra information is added to the original IP data, e.g. headers or sideband information. For monitoring at the routers or NIs this is premium information as it allows the original IP communication to be identified and decoded.

(3) The packetized data is transported via a supported multi-hop communication path. This allows router or NI monitors to be inserted (at design time) or activated (at run-time) at different positions along this communication path.

(4) Advanced design-flows are indispensable tools to tackle the associated inherent complexity of the NoC designs. NoCs are not just taken from an IP library but are designed around routers and NIs from such libraries. Monitoring, in the form of the router and NI monitors must be added at design time, and the impact (if any) on the existing NoC must be investigated and be accounted for. This means that a synergy exist between the monitoring and NoC design flows; the NoC design flow must integrate and be aware of monitoring.

Chapter 3

Generic NoC Monitoring Service

Networks on chip are considered by many as the interconnect of choice for the SoC design. While the analysis and design of such NoCs have been the focus of the research community, one important issue, monitoring, has not yet received the deserved attention. This chapter presents the basic features of a generic NoC monitoring service. After a brief motivation showing the need for a monitoring service and the associated related work, Section 3.3 clarifies what a NoC monitoring service actually is, what are its associated main features and what it means to add a NoCMS to a NoC. The monitoring service consists of generic hardware monitors and monitoring service access points. The monitors can be instantiated for the monitoring task at hand. They are run-time configurable, and their corresponding programming model is presented. The monitoring service access points are IPs which can configure the monitoring service, and which can receive the monitored data. They are also the entry points for the clients requesting monitoring data. The monitoring service can be instantiated as a centralized or a distributed system.

We advocate a generic monitor architecture enabling reuse comprised of three major components: a sniffer, an event generator and a monitoring network interface. This is made clear in Section 3.5. In order to be able to use the employed monitors the monitoring service must be configured. The configuration obeys a monitoring service programming model as depicted in Section 3.6.

The monitored data is locally abstracted at monitors in the form of events. The event model used by the monitoring system is presented in Section 3.4. Monitoring data generated by the monitors which has been abstracted at monitors must be sent towards monitoring service access points. Also monitoring service access point configuration data is required for programming the monitors. This is done by means of a traffic management strategy, elaborated in Section 3.7, with the help of two data transport scenarios.

With all the NoC monitoring service features presented it is time to revisit all from a cost perspective, as described in Section 3.8 where an initial cost estimation

is made. With this we conclude that the costs of our proposed monitoring service are acceptable.

The first ideas supporting this chapter have been published as *"An Event-based Network-on-Chip Monitoring Service"*; Calin Ciordas, Twan Basten, Andrei Radulescu, Kees Goossens, and Jef van Meerbergen; In Proceedings of International High Level Design Validation and Test Workshop (HLDVT), November 2004. IEEE,2004 [17]

An extended version of this paper with a closer resemblance to this chapter has appeared as *"An Event-based Monitoring Service for Networks on Chip"*; Calin Ciordas, Twan Basten, Andrei Radulescu, Kees Goossens, and Jef van Meerbergen; In ACM Transactions on Design Automation of Electronic Systems, 10(4), Oct 2005 [18]

3.1 Motivation

The need for multiple monitors

As already mentioned in the introductory chapter, system level observability solutions must include on-chip instrumentation modules called monitors that support the entire system of interest, the computation and the communication part. Such monitors are common for computation, e.g. at the core level [5], and for bus-based communication [32].

As the SoC world is moving towards NoCs, inter-IP communication becomes more sophisticated relying on run-time programmable solutions and will be able to use multiple truly parallel communication paths, as opposed to centralized bus communication in current SoCs.

As a performance monitoring example, a single central bus performance monitor is enough for bus-based systems to track the performance of the bus interconnect (e.g. utilization). As opposed to this, multiple performance monitors (e.g. the performance monitor of Chapter 5) are required to track the performance (e.g. utilization) of the system interconnect in NoC-based SoCs.

Also a single transaction monitor suffices in the centralized bus-communication to gather the transaction history. Due to the inherent parallel behavior of communications, where multiple pipelined parallel communications may exist between IPs, in the NoC-based SoCs more than one such transaction monitor (e.g. the transaction monitor of Chapter 4) has to be employed to recreate a transaction history of the system.

It is therefore clear that for a successful monitoring task we will have to deal with a collection of multiple monitors.

Interconnecting multiple monitors

While supporting multiple monitors in future NoC-based SoCs is a must, the problem of their interconnection arises. The employed monitors need to send their data towards the monitoring requestor, and have to be able to receive requests for monitoring, in the form of configuration data, requiring their interconnection together with the monitoring requestors. Interconnecting the chip-wide spatially distributed monitors poses a number of significant challenges. These challenges shall reflect in the features offered by any proposed NoCMS.

Many monitors participate together or in isolation in one or more monitoring tasks. Some of these monitors might be third party monitors. Any such monitoring interconnect must be scalable, non-intrusive, run-time usable and configurable, and of minimum area cost. Furthermore, we want generality, i.e. we want the same monitoring service concept, including the interconnecting of the monitors, to successfully apply to more NoCs.

As NoCs are a scalable interconnect they appear a natural fit for the task. Options like sharing or not sharing a single interconnect for functional and monitoring traffic are key to monitoring system design. By using a scalable interconnect like the NoC for the transport of monitored data from multiple monitors, as opposed to a single big multiplexer tree, the physical wiring problem (bottleneck) has been traded for potential congestion in the NoC.

Three scalable alternatives for interconnecting the spatially distributed monitors, all employing NoCs, are presented in Chapter 6, because of the inherent binding to the NoC design flow that is presented in the mentioned chapter. The rest of the chapters, including the current one, assumes that a single NoC is used for the monitoring as well as for the functional traffic, because it is the most area efficient solution. The motivation of this choice is further detailed in Chapter 6.

3.2 Related Work

While the previous chapter has presented related work on NoCs in detail and the state of the art in bus monitoring, including multi-core support has been presented in Section 1.6, this section presents the existing NoC monitoring related work.

Multiple monitors have been employed in one combination of NoCs and monitoring [69]. Here, the use of end-to-end performance monitors is proposed in order to run-time assist the operating system controlling the NoC. The monitored data uses a separate NoC, called the control NoC instead of the application NoC. It fails however to show what are the associated costs or implications of using monitoring, for example whether it is area efficient or not.

Embedded monitors in an FPGA environment are used to track end to end run-time behavior (queue utilization) as feedback for the design exploration phase in [63]. The employed hardware monitors have dedicated wires for transport of their results multiplexed in front of an output port, showing an approach that is

not scalable.

[1] proposes a dynamic routing scheme for reducing jitter in the latency of BE traffic, only in the combination with GT traffic, which can benefit from monitoring. It is assumed that the monitoring system is already in place, solved by a third party. The work also fails to show the associated costs.

The presented related work shows that while the need for multiple monitors and their interconnection is real, as multiple monitors have already been suggested or employed in different works, no work actually presents a coherent picture of what is actually needed in NoCs for NoC monitoring, or what are the options for interconnecting multiple monitors, their associated costs or a generic monitoring strategy which scales well with the size of the NoC. Our work tries to close this gap with the proposal of a generic NoC monitoring service which can be instantiated for the monitoring task at hand.

3.3 Introducing the NoC Monitoring Service

3.3.1 Monitoring Service Concepts

Monitoring service or monitoring system ?

In our view a NoC monitoring service provides its clients with information gathered from the NoC. This information is about the NoC or the surrounding NoC-connected IPs, and it is provided in a raw or a more abstracted form. For example, a client in the form of an internal protocol checking IP is provided with monitored transaction information gathered and abstracted by transaction monitors spying on a NoC link. The information comprising only the read and write messages with the commands and addresses (but without the data), between two IPs over a NoC connection. As a second example, another client taking the form of a run-time application quality manager is provided with NoC utilization information comprising link utilization for a selected number of links, gathered by performance monitors directly from the NoC.

The NoCMS is a service which is offered by the NoC in addition to the communication services offered to the IPs. Our proposed NoCMS seamlessly integrates the concepts of a NoC monitoring service on top of a NoC monitoring system. Confusingly enough, even in the (networking related) literature the form taken by monitoring is sometimes referred to as a service (a monitoring service) and some other times as a system (a monitoring system), without a clear difference being made between them. To keep the confusion minimal the existing subtle difference between them is pointed out here; the monitoring service is something which is offered to someone else, e.g. to a client, while the monitoring system is something which makes the monitoring itself or the delivery of the monitoring service to its clients possible, e.g. a monitor or a dedicated monitoring interconnect.

Throughout this work both the monitoring service and the monitoring system are jointly referred as the NoCMS (or network on chip monitoring service), the

terms "is offered" and "consists of" making the distinction clear between the monitoring service part and the monitoring system part.

NoCMS architecture

As previously mentioned, the NoCMS (the monitoring service part of it) is offered by the NoC itself, in addition to the communication services [79] offered to IPs. The NoCMS (the monitoring system part of it) consists of configurable monitors (P) attached to NoC components (routers and NIs) and monitoring service access points (MSAs), see Figures 3.1 and 3.2 for an illustrated version of it. The monitoring service access point (MSA) is an IP which controls the configuration of the monitors at run-time and receives the monitored data from the monitors it controls. E.g. the MSA can stream this data outside the chip through a debug port. The system wide distributed monitors and the MSAs are interconnected by a scalable interconnect.

The monitor modular design comprises three parts: the sniffer (S), the event generator (EG) and the monitoring network interface (MNI). The MNI can be a separate NI or can be merged with an existing NI. The generic and modular NoC monitor architecture (S + EG + MNI), allows e.g. to seamlessly change the EG, e.g. the performance monitor of Chapter 5 with the transaction monitor of Chapter 4, during the design process without changing the rest.

Note that we have chosen to add monitors to routers and NIs, because this gives access to the internals of these components. As an alternative one could consider attaching monitors to links but limiting in this way the observability to the information passing the links. In fact, the transaction monitor of Chapter 4 and the performance monitor of Chapter 5 use only the information passing the links.

Note also that the NoCMS provides support for the chip-wide monitoring system (on the computational part, e.g. processors) by offering the option of integrating third party probes (monitoring IPs) for the IPs connected to the NoC, like ARMs ETM probe [5] for the ARM processors. These connect to the monitoring NoC, see Chapter 6 for details, as any other IPs in the system. Their communication requirements are taken into account the same way as for any other monitors (e.g. performance or transaction monitors). In this work we further refer to the monitors as attached only to routers and NIs. The extension of it to include IP monitors is left as an exercise for the reader.

The NoCMS is configured by means of monitor programming via the NoC using memory-mapped I/O write transactions. Monitors are configured using their associated programming model. The monitor gathered data may be sent as is (raw) or locally abstracted at the monitors in the form of events; e.g. the reader may want to check Chapter 4 for levels of data abstraction from physical raw to logical transaction-event based, including the associated challenges. The data for monitor configuration and the data gathered by monitors is sent using a predefined monitoring traffic management strategy. In the following paragraphs

we briefly explain the main concepts, as they are detailed in the following sections.

Event model

All the monitored information is modeled at the monitors in the form of events. This supports event-based monitoring and on-chip abstraction of data. An event model specifies the event format, e.g. timestamped events or not. Currently, we focus only on timestamped events. An event taxonomy helps to distinguish different classes of events and to present their meaning. An event model with an associated event taxonomy and one instantiation of it is described in Section 3.4.

Monitors

The monitors are responsible for collecting the required information from NoC components. The monitors, Ps in Figures 3.1 and 3.2, capture data from the NoC components. This monitored information is locally converted in the form of timestamped events. The monitors are run-time usable; this means that they collect and locally process or abstract and send data while the system is running.

Multiple classes of events can be generated by each monitor, based on a predefined instance of an event model. Monitors are not necessarily attached to all NoC components. E.g. the top-right router in Figure 3.1 has no monitor attached. The placement of monitors is a design-time choice and is related to the cost versus observability trade off, see for more details Chapter 6. The employed monitors may be of different types, e.g. the transaction monitor of Chapter 4 or the performance monitor of Chapter 5. The generic monitor architecture is detailed in Section 3.5.

Programming model

The programming model describes the way in which the monitoring service is being set up or torn down. It consists of a sequence of steps for configuring the monitors and the means of implementing those steps. Monitors are programmed via the NoC, e.g. using memory-mapped I/O [78, 75]. Single monitors are run-time configurable, implying that the entire monitoring service can be configured at run-time. This configuration can be done by any master IP connected to the NoC from the corresponding NI. This master IP is called the monitoring service access point (MSA), e.g. MSA1 and MSA2 in Figure 3.2. The programming model is detailed in Section 3.6. The time required to configure the entire NoCMS is called monitoring service configuration time, see Section 4.10 for details.

Monitoring Service Access points

MSAs are NoC connected IPs which receive the monitored data from the monitors, make sense of this data locally, and/or forward it somewhere. They can also for example be memories used to store the monitoring data or simply a debug port

which streams the monitored data off-chip. As already mentioned, in principle any master IP connected to the NoC can act as an MSA; this IP only needs the knowledge (what and where to send) to configure the monitors. Note that al least one MSA is always required in a NoCMS.

Traffic management

Traffic management regulates the traffic from the MSA to the monitors, required to configure the monitors, and the traffic from the monitors to the MSA, required to get the monitoring information out of the NoC. Already available NoC communication services or dedicated solutions, e.g. a separate bus, can be used for the traffic management for monitoring. Several alternatives for interconnecting the monitors are detailed and evaluated in Chapter 6.

Non-intrusiveness

Non-intrusiveness is a key aspect in debugging, one of the main run-time monitoring drivers, and also in performance monitoring where we do not want the monitoring process and the monitoring data to perturb the functionality or the performance of the monitored system. Non-intrusiveness must be ensured at all levels.

We employ passive hardware monitoring in our proposed NoCMS. In the large context of networks, passive monitoring means that the monitoring device, e.g. a monitor, passively tracks a monitored device, e.g. a physical link by e.g. collecting utilization statistics. In the same context active monitoring means that the monitoring device is pro-actively trying to assess actual properties of the network by e.g. sending packets for a round trip in order to detect network latency.

For our proposed NoCMS, the employed passive monitoring ensures non-intrusiveness at the level of data gathering (SPY), see Section 3.5 for details. For a non-intrusive monitoring service, not only the monitoring itself but also the transport of the monitored data has to be done in a non-intrusive way. When a separate NoC is used for monitoring, non-intrusiveness is guaranteed, see Chapter 6 for this case. However, when the same NoC is used for transporting the monitoring and the functional data, non-intrusiveness is not guaranteed by default and extra steps during design time need to be done to ensure it, as presented in [20].

3.3.2 Distributed vs. centralized NoC monitoring service

We propose a NoC monitoring service that can be configured as a distributed or a centralized service, during run-time, at arbitrary moments in time. The same service can be later re-configured in a different form.

In a centralized monitoring service, as shown in Figure 3.1, the monitoring information from the selected monitors is collected in a central point, in this case a

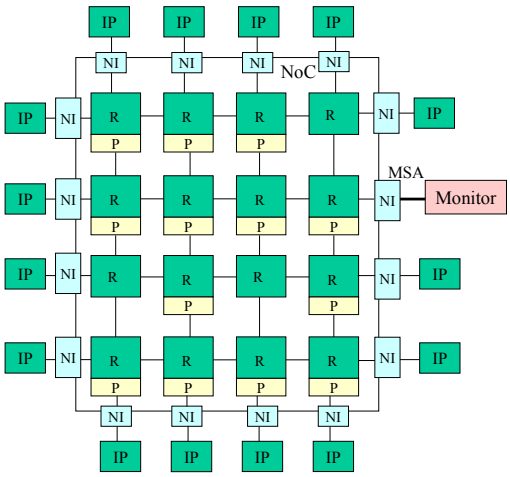


Figure 3.1: Centralized Monitoring Service

collector monitor, through a single MSA. For small NoCs, a centralized monitoring service is possible and convenient. However, the convergence of monitoring data to a central point may become a bottleneck in large NoCs.

In a distributed monitoring service, the monitoring information is collected for different subsets of NoC components at different points through multiple MSAs. In this way, bottlenecks are removed and we achieve scalability. If all of the present IPs have MSA capabilities as previously outlined, each of them can set up its own monitoring service, eventually sharing probes.

Figure 3.2 shows a distributed monitoring service composed of two subsets of components. One connects directly to a dedicated monitoring IP through MSA1. The second connects, indirectly through a router which is not part of the subset, to an off-chip interface through MSA2. The subsets can be programmed at run-time offering increased flexibility. Hence, a monitor probe can be part of one subset at one time and of a different subset at other times; see the monitors attached to the routers in the middle of the figure. Monitoring information can be either used on-chip, e.g. by the dedicated IP in Figure 3.2, or it can be sent off-chip either directly through an off-chip link or via a memory.

3.3.3 Adding a NoCMS

In our view adding a NoCMS to an existing NoC means:

(1) adding the monitors; this is done taking into account the monitoring task that is driving the monitoring activity, e.g. transaction-based debugging or performance analysis. Note that as previously explained the number of monitors may depend on the NoC topology, mapping of cores to NIs,

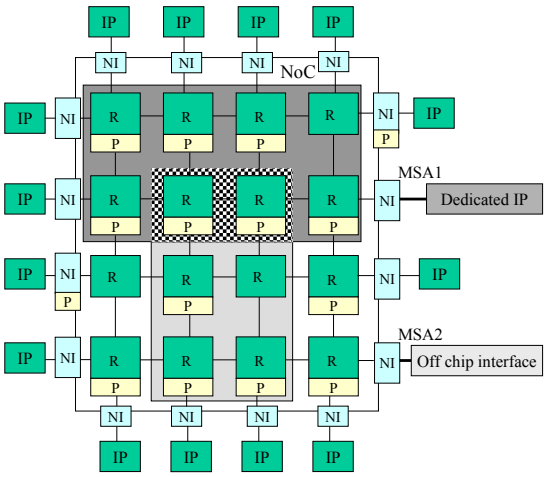


Figure 3.2: Distributed Monitoring Service

number of NIs connected to routers, etc.;

(2) adding one or more MSAs; a single MSA is required assuming a centralized monitoring system and more MSAs may be employed assuming a distributed monitoring service;

(3) connecting the monitors to the MSA(s) for the dual purpose of monitoring data transport and run-time monitor (re)configuration; this includes the selection of a suitable monitoring interconnect in case the functional NoC is not shared for monitoring. In this case, as well as in the case when the functional NoC is shared for monitoring, using the communication services of the NoC (or of the other interconnect) is the following step.

3.4 Event Model

3.4.1 Events

An event [57] is a happening of interest, which occurs instantaneously at a certain time. In our view, an event is a tuple:

Event=(identifier, timestamp, producer, attributes)

The mandatory event **identifier** identifies events belonging to a certain class of events and is unique for each class. The event identifier can be explicit or implicit. In the frame of a NoCMS it makes sense to have it explicit to distinguish between event classes when the same monitor is able to generate two or more event classes at the same time, as opposed to a single event class being enabled at a certain time. In the case of transaction monitors of Chapter 4 and performance monitors

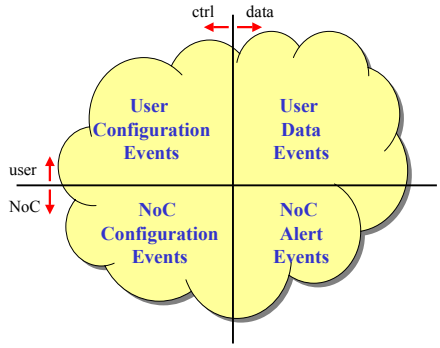


Figure 3.3: NoC Event Space

of Chapter 5 this is implicit, as these monitors can only generate a single event class during a certain time period after being programmed, until next time it is being programmed.

The **timestamp** defines the time at which the producer, in our case the monitor via its event generator, generates the event. The **producer** is the entity that generates the event. **Attributes** are the useful payload of events. Each attribute is present in the form:

Attribute=(attribute identifier, value)

An attribute consists of its **attribute identifier** and its **value**. The attributes and the number of attributes may depend on the event type.

Note that in some particular NoC context the identifier, timestamp or producer are not always needed to be explicit at an event creation but can be inferred when the events are collected.

3.4.2 NoC Event Taxonomy

In the following, we present a taxonomy of NoC events. The term user is used to identify an IP. In general, we can group NoC events in five main classes: user configuration events, user data events, NoC configuration events, NoC alert events, and monitoring service internal events.

The criteria used for this taxonomy are graphically presented in Figure 3.3. Within the NoC, we can have either user traffic or NoC internal traffic. Furthermore, there can be control traffic or regular data traffic. These two dimensions define four of the five mentioned classes. All these classes represent data of the monitored system. The fifth class of events is the class of monitoring service internal events. Whether data is generated by the monitoring service or by the monitored system can be seen as a third dimension. The resulting taxonomy covers all relevant groups of events and is general enough to be valid for different types of NoCs, although event types may need to be redefined for each specific

NoC. The reason to choose these classes is that each of them may be useful in different debugging tasks. Also, other dimensions can be added to this taxonomy in order to refine it. One such dimension could be the grouping of events according to the component where they are generated, e.g. in the probes attached to routers or NIs.

In the remaining chapters of this thesis, only the user data events and NoC alert events were further investigated and resulted in the proposal of a transaction monitor and a performance monitor, although with other event capabilities than the ones described here.

User Configuration Events

A NoC is used by IPs to communicate to each other. User configuration events expose this communication, presenting a system-level view of it, hiding NoC specific details. User configuration events can show for example that processor X is communicating with processor Y via a connection. These events can be very detailed, for example exposing the properties of the connection, or very abstract showing just the timing of the communication and the communicating parties. This class of events may be useful to check the system level interaction of components.

User Data Events

This class of events is the class that allows the sniffing or spying of functional data from the NoC. The sniffing itself can be from the NoC elements or from the links. Sniffing may be required for example to check whether the transmitted data, such as a memory address, is exactly the intended data. It can allow for example sniffing of flits, network packets or complete messages (as done in Chapter 4) depending on the NoC and the purpose sniffing is used for. This class of events may be useful to check data details of the specific interaction of components.

NoC Configuration Events

To achieve interprocessor communication, the NoC must be programmed or configured, statically or dynamically, in a centralized or distributed way. NoC configuration events expose this configuration of the network, enabling the system debugger to trace configurations, allowing it for example to observe the setup of a specific connection. Examples of such events are the fact that a new entry in a routing table has been completed or that the routing table is full. Attributes of these events can be for example the party that wrote the routing table entry. These events are particularly useful for NoC debugging and optimizations.

NoC Alert Events

After programming the NoC, network problems can arise. Problems like buffer overflow, congestion, starvation, livelock or deadlock can appear. In real-time

systems, missing a hard real-time deadline is a serious error. Therefore, it is imperative to monitor the network behavior and be alerted if signals of overload or misbehavior appear. An example of an event fitting into this class is the continuous lack of progress in a non-empty router queue indicating that the NoC might be congested or even deadlocked. An attribute of such an event can be the number of cycles the specific queue was idle. Performance analysis events, like average queue fillings or other metrics, can be included here also. Performance debugging may typically need this class of events, as illustrated in Chapter 5.

Monitoring Service Internal Events

This class of events contains all the events used by the monitoring service for its own purposes, such as synchronizing or ordering of events, or to signal extraordinary behavior of the monitoring service, e.g. monitoring data loss.

3.4.3 Æthereal Events

In principle, many instantiations of the above taxonomy for any given NoC are possible, depending on the level of abstraction of the defined events and the monitoring purpose. This section presents examples of potential Æthereal NoC events in each of the corresponding event classes. As producer and timestamp are always present, we only give the identifier and the attributes for each event.

User Configuration Events

Interprocessor communication via the Æthereal NoC is performed by means of connections. As an example the *Connection Opened* event could shows when a certain connection has been opened. The attributes are the connection identifier, the type of the connection, e.g. narrowcast, the ports between which the connection exists, the path of the connection and whether it is a GT or a BE connection. A *Connection Closed* event shows when a connection is torn down. Its attribute is the connection identifier.

User Data Events

Sniff events for the Æthereal NoC refer to sniffing flits, either BE or GT. Flits are sniffed from the router queues. Sniffing multiple flits can emulate sniffing a complete packet or even complete messages (see Chapter 4). *BE Sniff* and *GT Sniff* events are example events that can be generated when a BE flit and GT flit are sniffed, respectively. Their attributes are the identifier of the queue from which the flit was sniffed and the BE or GT flit itself.

NoC Configuration Events

The *Reserve Slot* event is an example of NoC configuration event that shows when a certain slot in the slot table of a router or network interface has been reserved. The attributes are the slot number and its value. The *Free Slot* event shows that a slot table in the router has been freed. Its attribute is the slot number. A *System packet arrived* event shows when a programming packet addressed to a router has arrived there. Its attribute is the entire packet. The purpose of this attribute is to trace a system packet and to see what actions will occur because of it. E.g. a setup system packet can go through multiple routers and program all of them.

NoC Alert Events

A *Queue filling* event could be specified taking into account the number of queues a router has. In case of a four port router, we have four attributes, namely the queue fillings in absolute numbers for each of the four queues. A *Queue full for X cycles* event could have as attributes the queue identifier and a value for X. The queue identifier pinpoints the specific queue while the X attribute is a number specifying the number of clock cycles the queue stayed full. A *Queue resuming sending* event, with attribute queue identifier, shows that the queue has resumed sending packets after it has been idle for some time. An *End-to-end credit 0* event is an example flow control event showing when the remaining buffering credit for a certain connection is zero, leading to a blocked IP. Its attribute is the connection identifier.

Monitoring Service Internal Events

Each monitor in a NoC can generate events. The total order of events for one event generator is given by the timestamp. For area efficiency reasons, a timestamp is necessarily limited to a specific maximal value. After reaching this value, the timestamp counter wraps itself. A *Synchronization* event could be used to indicate when the event counter wraps. The event has no attributes and is only required to allow the proper synchronization for one probe. Currently, Æthereal works with a totally synchronous NoC. The event definition for the Æthereal setup allows to reliably reconstruct the overall partial order of monitoring events. The methods described in this work will also work with asynchronous NoCs but the timestamping policy will be influenced.

3.5 Monitors

3.5.1 Generic Architecture

The generic NoC monitor architecture consists of three components, see Figure 3.4: a sniffer (S), an event generator (EG) and a monitoring network interface

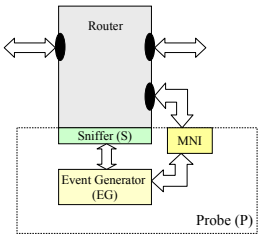


Figure 3.4: NoC Probe Architecture

(MNI). The monitor has as input a number of signals obtained by the sniffer from the router. Based on these signals, the monitor generates events through the EG, which must be sent to the MSA. This means that the events must be packetized and sent through the network. This is what the MNI is doing. The means of implementing the sniffer, EG and MNI blocks and their architectural details, e.g. timestamping policy, are NoC dependent.

3.5.2 Æthereal Probe Architecture

This section presents the architecture of the monitors for the Æthereal NoC, based on a prototype implementation for evaluation purposes. Monitors are implemented in hardware and are a design-time choice. The system designer has to decide what level of monitoring is desirable and affordable. The proposed monitoring architecture is suitable for multiple levels of abstraction.

An Æthereal monitor is attached to a router or to an NI. For simplicity, we restrict ourselves to routers in the following. For NIs, there are no conceptual differences.

The Æthereal monitor can be seen in Figure 3.5. It features a modular design, and can be used without changing the design of the router or NI. In the following, we briefly present each of its components. The basic scenario illustrated in the example figure is very simple: programming packets are coming through the NoC on any of the router ports, e.g. I1 in Figure 3.5. They are transferred by the router from I1 to output O5. The MNI depacketizes the programming packets and configures itself via CNIP and the EG via the Configuration Port. The EG generates events and transfers them to the MNI, via the Event Port, where they are packetized. Packets are sent to the input I5 of the router to be sent to the MSA, via e.g. O2 in Figure 3.5.

Sniffer

The task of the sniffer is to get info from the router and offer it as input data to the EG. Sniffing the signals is not intrusive. The input data are signals obtained by means of SPY-like [100] mechanisms. The SPY approach allows a limited set

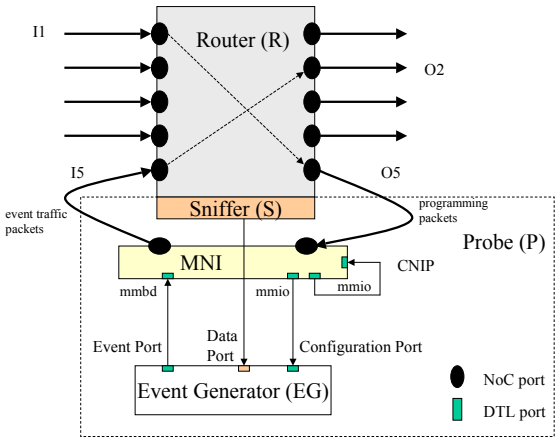


Figure 3.5: Æthereal Monitor Architecture

of signals to be monitored in real time, while the chip is running an application, using a multiplexer tree of dedicated wires attached to points of interest in the circuit/SoC. Sniffers can be attached either to routers or to the links between them. We attach the sniffer directly to routers because then we can also have access to the router internals. The sniffer delivers the signals to the EG data port.

Event Generator

The task of the EG is to generate timestamped events based on the input received from the sniffer. The EG can generate instances of multiple event types. Our test implementation implements all the example events of Section 3.4.2. Event types supported are a design-time choice, their selection is a run-time choice. The EG passes the generated events to the MNI.

The general format of the Æthereal NoC event in our evaluation implementation is:

identifier	timestamp	producer	attributes
8 bits	16 bits	8 bits	$\geq$ 0 words

The Æthereal event **identifier** is an 8 bit code in the proposed format. Æthereal events are not all of the same size. Event attributes can be enabled or disabled allowing any combination of existing attributes for one event. The number and size of attributes is determined by the identifier.

The format uses a 16 bit **timestamp**. This timestamp is obtained by taking advantage of the 8 bit counter in the router used for slot table iterations. As an 8 bit counter is considered too small for the timestamp, for synchronization reasons, it is extended with another 8 bit counter in the event generator itself. In this way, we can iterate 256 revolutions of 256 slots, without wrapping the

timestamp counter. Æthereal NoC has showcased four router designs [41] with the first three supporting a combination of GT and BE traffic and the last one supporting only GT traffic. The first from the three GT/BE router designs, for which this monitor was initially built, features an internal slot table unit with the mentioned 8 bit counter; this counter is not present in the other router designs, which means that it has to be provisioned as part of the monitor.

The **producer** specifies the router that has the monitor generating the event attached to it. An 8 bit code is used, allowing for 256 producers, which fits today's requirements.

The identifier, timestamp and producer together consist of one (32bit) word matching the Æthereal link width. The **attributes** can have any maximum size in principle. In a test implementation of all the example events discussed in Section 3.4.2, our longest event with all attributes enabled is 5 words.

EGs contain an on/off switch, masks for selection of events, their corresponding attributes and activation time, the timestamping unit and a queue for events. In the evaluation implementation, we use a 10 words queue to accommodate two maximum size events.

The proposed Æthereal EG has three ports, see Figure 3.5: one data port, one configuration port and one event port. The data port gets the input from the sniffer. The programming port, a memory-mapped I/O slave, is connected to the MNI output port. The event port, a master memory mapped block data is connected to the MNI input port. The generated events are posted in the internal queue and from there they are passed to the network interface.

Monitoring NI

The MNI is a standard NI, see Chapter 2 for NI details; we call it the MNI in order to help distinguish it from the other NIs present in the NoC. The events generated in the EG are transferred to the MNI. The MNI packetizes events and sends them via the NoC to MSAs like any other data. The MNI can be configured by any master attached to the NoC through its configuration port CNIP [78] in order to setup the connection for the monitoring packets.

The MNI has two Network Interface Ports (NIPs) for communication with the EG: one master memory-mapped I/O port and one slave memory-mapped block data port. The master port connects to its slave pair in the EG and the slave port connects to its master pair in the EG. This MNI has also one bidirectional port to communicate with the router, and is independent of the events fed to it. Packets are queued internally in the MNI queue and then sent to the router.

The design of the MNI makes it possible to treat the EG like any other IP connected to the NoC, which has advantages in the design of the monitoring service and in the co-design of the system and its debug support.

3.6 Monitor Programming Model

3.6.1 Generic Model

The previous section explains the architectural features of the monitors. This section presents the associated programming model.

In general, it is important to decide when a monitor can be configured. Assuming monitors are physically present in the NoC, they can be configured at three possible moments: NoC initialization time, NoC reconfiguration time, or run-time.

Our goal is to make the service available at arbitrary moments at run-time. The monitors are programmed using the NoC itself. It has been shown [78] how to configure the NoC at run-time, using the NoC itself. Similar techniques can also be used for programming the monitors, requiring no additional communication infrastructure.

The programming of the monitors must include the selection of desired events to be generated, including their desired attributes, the selection of monitors to generate the events, a means to enable or disable monitors, a way of timing and the setup of traffic monitoring connections.

3.6.2 Æthereal Model

For Æthereal, we are able to configure the monitors at any of the above mentioned moments using memory-mapped I/O read/writes as in AXI [6], OCP [70] or DTL [75]. The EG is a slave with a memory-mapped configuration space slave interface. This means that the registers in the EG appear in the general memory map allowing them to be read or written by any master. In this way we implement the programming of the EGs. This is in line with the de facto programming model for the NIs. An EG can therefore be configured by means of simple memory-mapped I/O write operations. Multiple monitors can be programmed independently in parallel. If more monitors must be configured, then each can be selected at run-time and each monitor must be configured separately.

Programming follows two conceptual programming steps for each monitor:

1. Monitoring connection setup. Events are generated in the EG and then packetized in the MNI. Packets containing events must reach the MSA where the monitoring service has been requested. This is done by setting standard Æthereal connections from the MNI to the MSA. Note that both the MNI and the MSA are standard NIs supporting such connections.

2. Monitor setup.

   (a) Event selection. After a connection has been configured, the desired events to be generated must be enabled. By default, all events are disabled. Events are activated by writing the event masks in the EG. Multiple events can be active simultaneously.

(b) Select attributes. Each of the attributes of an event can be selected. Therefore, for each of the events selected in the previous step, the selection of desired attributes is mandatory. The default is that all attributes are disabled. This means that by default all events will be limited to identifier, producer and timestamp. Attributes are enabled by writing attribute masks in the EG.

(c) Select time. When programming the monitor, the programmer can optionally set the time that the event generation should start. This is done by setting the time mask in the EG. This mask will be compared with the 8-bit value of the counter in the EG, and at the first match the event generation will start. Selecting the start time of the event generation guarantees in a synchronous NoC the simultaneous start of the event generation by multiple monitors.

(d) Enable/disable monitor. Even after the time selection, the monitors have to be enabled. Monitors are by default disabled. The user can enable or disable any monitor in the NoC. Only when the monitor is enabled, the event generation starts. Without the time mask the event generation starts immediately. If multiple monitors are enabled, it is not possible to guarantee that they will be simultaneously enabled, because of network latency. If the same time mask is set for all EGs, all EGs will simultaneously start at the first encounter of the time set. Enabling or disabling monitors is done by writing the enable/disable mask.

The time required to set up the monitoring system is composed of the time required for setting the connections between the EGs and the MSA, and the time required to configure the monitors. As an example, the setup time for one connection with a path of length four is 90ns [78]. The time required for configuring the monitors depends on the time required for a write operation to a monitor register and the number of registers to be written. E.g. doing a write transaction for monitor configuration, with a payload of two words, to a monitor via a GT connection of length three takes 54ns. The write transactions for programming the monitors can be pipelined. We need further experimentation with realistic applications of the monitoring service to assess the impact of the setup time on the performance of the monitoring service.

A concrete example of configuring a transaction monitoring system spanning an entire 2x3 Æthereal mesh NoC with six transaction monitors and having a single MSA is presented in Section 4.10, together with a few configuration policies: using BE or GT, using different message orderings, each using a protocol with or without acknowledgement. All of the configuration policies exemplified have resulted in acceptable configuration times, e.g. 1212ns when using GT and a 64-bit DTL-MMBD write operation.

3.7 Traffic Management

3.7.1 Generic Strategy

The monitoring traffic is composed of the monitor configuration traffic and the event traffic.

Monitor configuration traffic

The monitor configuration traffic is all traffic required to setup and configure the monitoring service. It includes the traffic required to configure the monitors, which flows from the MSA NI to the MNI, and the traffic for setting up the connections for the transport of data from the monitors' MNIs to the NI port of the MSA that requested the monitoring service. The monitor configuration traffic depends on the number of probes being setup.

Event traffic

The event traffic is all the traffic produced as a result of event generation in monitors. This traffic flows from the monitors' MNIs towards the MSA NI. The event traffic depends on the number of monitors set up as well as on the time a monitor is enabled.

The monitoring traffic can use existing NoC communication services or a dedicated interconnect, e.g. a debug bus. In case existing NoC services are used, additional traffic is introduced in the NoC but no extra interconnect is needed. However, in certain cases, the existing NoC may have to be re-dimensioned to accommodate the additional monitor configuration and event traffic; see Chapter 6 for a detailed analysis and examples. In case of a dedicated interconnect for monitoring only, no additional monitoring traffic is introduced on the existing NoC but more effort is required to design or use another scalable interconnect.

3.7.2 Æthereal Strategy

The Æthereal monitoring traffic uses the NoC itself and it is based on the existing Æthereal communication services. In this way, no separate interconnect, for control as well as for use, is required for the monitors. There are several choices:

Using GT services

All the monitoring traffic, the event traffic and the monitor configuration traffic, uses GT connections. Two connections are set up between the MSA NI and the MNI of the specific monitor; one for the event traffic and one for monitoring configuration traffic. Each monitor in the system uses its own pair of GT connections. In this way, even if the network is congested, monitoring traffic can still

reach the MSA NI at a guaranteed data rate, offering a real-time behavior of the monitoring service. BE user traffic can use the reserved slots when no monitoring traffic is present. It is the safest option from the debugging point of view, but it may interfere with other BE traffic. BE traffic can be employed to setup of new GT connections or as traffic not related to monitoring but to the monitored system (ordinary BE traffic). In our test implementation, we use GT services as the implementation choice.

Using BE services

All monitoring traffic uses BE connections. In case of congestion, it may not be possible for monitoring traffic to reach the MSA NI at a predefined data rate. The use of BE services is the least intrusive for existing traffic because it does not interfere with GT traffic, but it may interfere with other BE traffic. Note that GT monitoring connections as discussed in the previous option, interfere with user GT connections in the sense that they limit the slot table allocation for the latter. Also note that the use of BE connections for monitor configuration traffic does not guarantee an upper bound for configuration time. One way to alleviate this problem is the use of acknowledgements during monitoring service configuration. In this way an acknowledgement message is sent back to the MSA when each monitor is configured; when the last acknowledgement is received at the MSA the configuration is considered completed.

Using GT and BE services

When configuring the monitoring service, if multiple monitors are used, it is possible to use either GT or BE for each monitor, in order to balance the overhead of the monitoring service. For the distributed monitoring service shown in Figure 3.2, for example, the traffic from all the monitors in the subset of the dedicated IP can use GT services and the traffic from all the monitors in the subset of the off-chip interface can use BE services.

Note that a combination of GT and BE per monitor is also possible, as these are connection properties, assuming that the single monitor uses more than one connection for the transport of its monitored data; this work assumes that one monitor uses a single monitoring connection for the event traffic.

3.7.3 Data Transport Scenarios

Whether the monitoring traffic is in the form of monitor configuration traffic or in the form of event traffic as previously presented, two data transport scenarios for this data are possible: the memory mapped and the streaming data scenarios. Both scenarios are visually depicted in Figure 3.6 and further explained. Note that these two data transport scenarios also apply to any other IPs connected to the network exchanging data between them, and not only to the monitors.

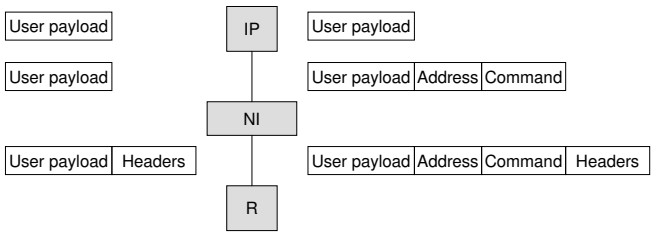


Figure 3.6: Streaming Data (left) versus Memory Mapped (right) scenarios

If the sniffed data is memory-mapped, e.g. the MSA is a memory, the monitor uses write transactions to send the data to the MSA, and a command and address have to be added to the monitor data. Therefore, a write transaction from a monitor to an MSA will always contain a command, an address, and the useful payload. In the remainder of this work, we refer to this as the *memory-mapped* scenario.

If the sniffed data is not memory-mapped, e.g. the MSA is another IP which takes care of streaming the data off chip, no commands and addresses are added to this data. Therefore, a peer-to-peer streaming data transaction from the monitor to MSA will contain just the useful payload. In the remainder of this work, we refer to this as the *streaming data* scenario.

Note that in both scenarios headers are added in the NI to the sniffed data as the effect of packetization.

3.8 Monitoring Service Cost

3.8.1 Cost Inventory

Our proposed NoC monitoring service provides run-time observability of the NoC behavior. However, this capability does not come for free for the NoC or SoC designer. This section presents an inventory of costs associated with it: area, traffic, and energy.

Area

Area target. We target an area budget for the NoC monitoring service of 15-20% of the NoC area. NoC area does not include the area of the IPs or other parts of the chip, it refers to the NoC interconnect area only. Looking at the area cost of commercially available run-time monitoring solutions for computation cores like ARM's ETM has led us to this area budget.

The area cost of several ARM cores [2] and ETMs [5] are presented in table 3.1. Together with the ARM cores their cell technology is presented, and their corresponding ETM version with its own area cost in the same technology

Table 3.1: ARM Cores and corresponding ETMs

core	*technology* (μm)	*core area* (mm^2)	*ETM*	*ETM area* (mm^2)	*% of core*	*% of total*
ARM7TDMI	0.18	0.53	ETM7	0.34	64.15	39.08
ARM7TDMI-S	0.18	0.62	ETM7	0.34	54.84	35.4
ARM720T	0.18	4.7	ETM7	0.34	8.08	6.75
ARM926EJ-S	0.13	2.39	ETM9CS	0.38	15.9	13.72
ARM968E-S	0.13	0.4	ETM9CS	0.38	95	48.72
ARM966E-S	0.13	1	ETM9CS	0.38	38	27.53
ARM1022E	0.13	6.9	ETM10	1.12	16.23	13.97

as the ARM core. The last two columns of the table show the area percentage the monitoring, in the form of the ETM core, takes relative to the corresponding ARM core and relative to the subsystem they form, the ARM core together with the ETM core. Relative to the core the monitoring spans from the very optimist 8% to 95% with an average of around 42%. Relative to the total it spans from around 7% to almost 49% with an average of around 26%; if we average only for the ARM cores realized in the 0.13μm) CMOS cell technology and we leave out the core with the highest percentage of monitoring relative to it, ARM968E-S, we obtain an average of around 18%. Note that these numbers do not include the cost of an Embedded Trace Buffer (ETB) [4] that may be required, e.g. when no high speed port is available. For communication observability, e.g., our monitoring system, we believe a similar area budget is reasonable; therefore, we target an area budget for the NoC monitoring service of 15-20% of the NoC area.

Area inventory. Looking at Figure 3.5, we get a basic idea of the components that must be physically added to the routers for the NoC monitoring system. All these components have an impact on the area:

1. Figure 3.5 suggest that an extra bidirectional router port is required to connect the monitor. This means that all probed routers will have a higher arity. For example, moving from arity 4 to arity 5 for an Æthereal router means increasing router area from 0.11mm^2 to 0.13mm^2, in a 0.13 micron technology. As we see later, in Section 3.8.2, it turns out that we can often optimize the monitoring system in such a way that an extra port is not needed.

2. A network interface with two network interface ports (NIPs)(mmio and mmbd in Figure 3.5) and one configuration port (CNIP) needs to be added per probed router. The cost of such an Æthereal NI is 0.07mm^2 in a 0.13 micron technology. For our monitoring solution this can be optimized; see Section 3.8.2 for more details.

3. A sniffer is required in order to get information from the routers in a non-intrusive way. Details of the SPY mechanisms and their area costs are presented in [100]. The costs are determined by the number of signals the designer wants to be monitored. Thus, it makes for example a difference whether a designer wants to monitor a single router queue at a time or all queues simultaneously.

4. An Event Generator creates events based on the signals monitored by the sniffer. The cost of the EG depends on the number and type of events desired. Therefore, EGs can vary from very simple ones, coming at basically no area cost, to very complex ones with significant area cost. For example, an estimation of area cost for one EG in the form of a watchpointing unit for four Æthereal router links, together with its corresponding Sniffer, is $0.028mm^2$, in a 0.13 micron technology. This estimate is based on the cost of a 128-bit Sniffer, corresponding to sniffing the four 32-bit router links, a watchpointing unit with four 32-bit comparators, and one 128-bit control register, meaning that all the Æthereal router links can be watchpointed simultaneously.

Traffic

The NoC monitoring system presented uses the NoC for transporting the events it generates. Therefore, the NoC monitoring traffic coexists in the NoC with the user traffic. In this way, it uses NoC resources. The NoC monitoring traffic should be as low as possible compared to the user traffic.

The previously mentioned area costs, are the directly visible costs of the NoC monitoring system. It may be possible to also have a hidden cost: the NoC designer may have to overdesign the NoC in order to be able to accommodate the monitoring traffic besides the user traffic.

Energy

The energy consumed by the NoC monitoring system consist of a variable part related to the monitoring traffic and a constant part required by the probes to function.

In the remainder of this section and throughout this work, we further quantify only the area and traffic costs. The energy aspect is left for future work.

3.8.2 Area

MPEG Examples

In order to study the area cost, we start with two NoC instances for an MPEG codec with 24 IPs including 3 memories, 21 GT connections with bandwidth

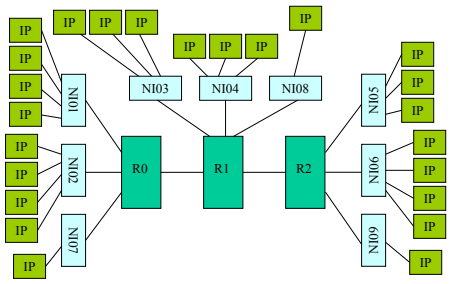


Figure 3.7: MPEG NoC1

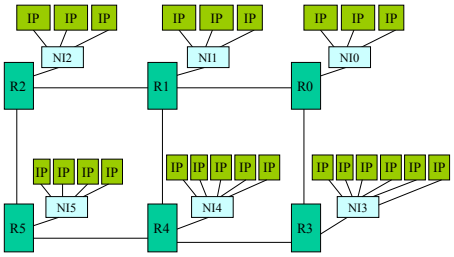


Figure 3.8: MPEG NoC2

varying from 54 to 120 MB/s. The two examples were designed, evaluated and presented in [40].

The first example, NoC1 in Figure 3.7 is a 3x1 mesh. Each router has three NIs attached to it, each NI connecting multiple IPs to the NoC. The original area of this NoC, without monitoring, is 1.86mm^2. The area does not include the IPs connected to the NoC.

The second example, NoC2 in Figure 3.8 is a 2x3 mesh. Each router has one NI connected to it and one or more IPs are connected to each NI. The original area of this NoC, without monitoring, is 2.35mm^2, again not including the IPs.

The NoCs used in the two examples have different topology, sizes, and number of NIs attached to the routers. Both NoC instances are automatically generated using the Æthereal design flow [40]. Results are SystemC and synthesisable RTL VHDL, directly usable in a back-end design flow. NoC1 of Figure 3.7 was subsequently manually optimized for area.

Adding the probes. Naive version

We have added the monitors to the already presented examples. We have implemented a centralized monitoring service for both NoC examples. Figure 3.9 shows the architectural implications for NoC2. A distributed monitoring service, see Figure 3.10, has been developed for NoC2. In the centralized monitoring ser-

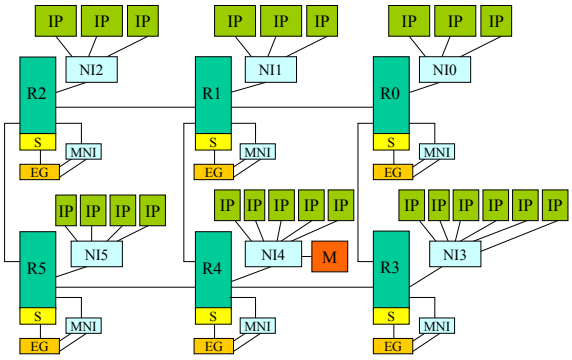


Figure 3.9: NoC2 extended with monitors, centralized service

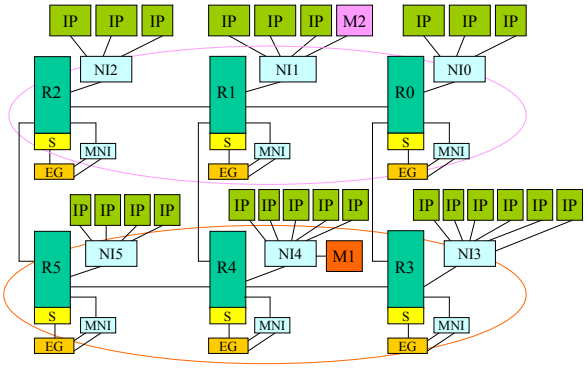


Figure 3.10: NoC2 extended with monitors, distributed service

vice of Figure 3.9, all monitoring traffic goes to the IP called M. In the distributed monitoring service, the traffic from monitors attached to routers R0, R1 and R2 goes to IP M2, and the traffic from the monitors connected to routers R3, R4, and R5 goes to IP M1.

The area of NoC1 extended with monitors is 2.26mm^2. The area of NoC2 with probes is approximatively 3.09mm^2. The centralized version is slightly smaller than the distributed one but that is not visible in the area numbers with two decimals accuracy. The monitoring service needs a NIP for each MSA. The area results do not include the area for the sniffers and the area for the EGs, but only the area of the extra cost parts in the routers and MNIs. As already mentioned, the size of the sniffer and the EG heavily depends on the events that should be supported.

Taking into account the area estimate of 0.028mm^2 for the EG in the form of a complex watchpointing unit with its associated sniffer, as described in Section 3.8.1, the area of NoC1 extended with monitors becomes 2.34mm^2 and the

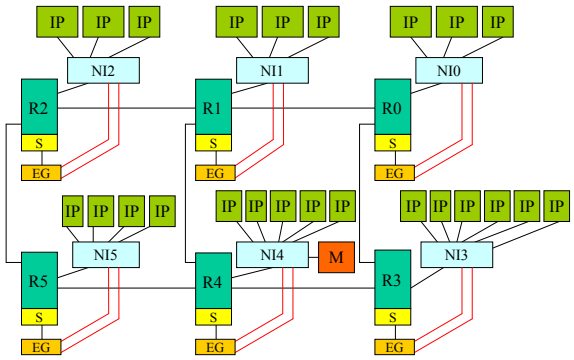


Figure 3.11: NoC2 with optimized probes, centralized service

area of NoC2 with monitors becomes 3.26mm^2.

Improved version

Observing the probed MPEG NoCs from Figures 3.9 and 3.10 we see that on the probed routers we already have at least one NI besides the MNI of the monitor. As we have previously mentioned, the MNI is a common NI parameterized for monitoring purposes. It is therefore possible and straightforward to merge the MNI with one of the NIs already connected to the routers, reducing the area cost, see Figure 3.11. The resulting NIs are larger in area than the NIs not merged with their MNI counterpart but we can reuse the NI present together with its configuration port and one of the router ports which already connects the NI. One router port is therefore also spared when compared to the naive solution, further reducing the area cost. Please note that this optimization is only possible because we have reused the NI design for the MNI. The resulting area of the optimized NoC2 as presented in Figure 3.11 is approximatively 2.75mm^2. This area result does again not include the area of the sniffers and EGs. Taking into account the area estimate for the watchpointing EG with its associated sniffer, like in the previous paragraph, the area of NoC1 extended with monitors becomes 2.17mm^2 and the area of NoC2 with monitors becomes 2.92mm^2

Area conclusions

To conclude the area section, we summarize the results:

area (mm^2)	NoC1(8 slots)	NoC1(16 slots)	NoC2
initial	1.86	1.94	2.35
naive	2.26 (+21%)	2.26 (+16%)	3.09 (+31%)
naive+EG/S	2.34 (+26%)	2.34 (+21%)	3.26 (+39%)
improved	2.09 (+12%)	2.09 (+8%)	2.75 (+17%)
improved+EG/S	2.17 (+17%)	2.17 (+12%)	2.92 (+24%)

It turns out that the original hand optimized NoC1 cannot accommodate the monitoring traffic with the built-in 8-slot routing tables. Therefore, the number of slots had to be increased from 8 to 16, leading to an increase in the NoC1 area from $1.86mm^2$ in the original version to $1.94mm^2$. If we consider the original NoC1 instance with 8 slots as the starting point of comparison, the result is an increase in area with 12% (not counting the EG/Sniffer components); otherwise the result is an increase in area with 8%. Taking into account also the EG/Sniffer area estimates, area increases with 17% and 12% respectively.

NoC2 can accommodate the monitoring traffic without modifications. Compared to the original version, we see an increase in area of 17%. Taking into account also the EG/S area estimates, area increases with 24%. Please note that the 24% increase for the NoC2 case is only $0.57mm^2$.

These examples show that:

1. The area cost of the NoC monitoring service based on the NoC itself, not including the area of the sniffer and the EG, are contained within an average of 14.5% (average of 12% for NoC1 and 17% for NoC2). The sniffer and EG sizes depend on the events desired to be monitored and therefore can substantially vary.

2. The total area costs of the NoC monitoring service including the watchpointing EG and sniffer estimates are sustainable, and in the order of 17-24% compared to the whole NoC area. The monitoring system for NoC1 (17%), fits into the proposed area budget. The monitoring system for NoC2 (24%), exceeds the proposed area budget. On average (20.5%), the area cost is very close to the proposed area budget of 15-20%. Together with further optimizations, it should be possible to bring the total area cost for NoC2 monitoring system as well as for other examples within the proposed NoC monitoring service total area budget of 15-20%.

The presented area costs are first results and can be improved. No optimizations, except the most straightforward ones were made. However, such optimizations are possible; for example, not probing each router but using a smart placement of the monitors can drastically reduce the overall area cost of the monitoring system, making the NoC monitoring service area cost even more acceptable. It is beyond the purpose of this chapter to present these optimizations; they will be presented in Chapter 6. Also a further specialization towards specific tasks may reduce the costs, as illustrated in Chapters 4 and 5.

3.8.3 Event Traffic

Media processing SoCs for Set-top Box applications or digital TV consist of audio encoding or decoding, e.g. AC3, and video processing functions, e.g. H263 or MPEG-2 [42]. We briefly look into the monitoring event overhead for such a media processing SoC using a NoC.

From [42], we learn that the typical task graph of such a SoC has approximately 200 connections all over the NoC. When the application task graph changes, a partial or complete reconfiguration of the NoC is needed. A complete reconfiguration means that all the connections are torn down and a new set of connections is set up. Reconfiguration is required at a maximum rate of once per second. A partial reconfiguration means that only part of connections, e.g. half, are torn down and a similar number of connections are set up.

In the case we monitor reconfiguration, we focus only on two events, namely *OpenConnection* and *CloseConnection* events. The average cost of these events is two (32bit) words per event. Each monitor monitors the *OpenConnection* and *CloseConnection* events, with all attributes enabled. One flit comprises three words, one being the header and two being the useful payload. For a complete reconfiguration, the total useful event payload is 3.2KB, leading to an event traffic of 4.8KB i.e:

200(connections) x 2(events) x 1(flit) x 3(words) x 4(bytes) = 4800 bytes

In case we monitor the functional data over one connection, e.g. a reserved 30MB/s GT connection, used for writes to a memory, with an actual usage of 28.6MB/s, we focus on the *GTSniff* event. In this case, as we monitor only one connection, the source, the identifier and the timestamp of the events are not necessary. An event will be the flit itself. We are interested in the functional data passing the connection. The 30MB/s GT user connection uses 50.25MB/s traffic in the NoC, considering all the overhead caused by the transaction protocol used (commands and addresses are added to messages) and the packetization. In this way, we sniff 50.25MB/s of data which is the payload for the debug connection. Assuming the monitoring connection writes this data into memory, through the MSA, in the memory mapped scenario, commands and addresses are added bringing the total required before packetization to 62.3MB/s, and to 83MB/s afterwards it. These numbers can be further improved, e.g. by (1) using the streaming data scenario for sending the sniffed data to the MSA or (2) removing the packetization overhead from the sniffed data, the sniffed packet headers.

These examples show that the traffic related to the monitoring service is sustainable by mature NoCs if events are carefully selected and enabled at the right time. For example, the raw bandwidth of the Æthereal NoC is 2GB/s per link and the monitoring traffic from our reconfiguration monitoring is 4.8KB/s, approximately six orders of magnitude less, while the monitoring traffic from our connection sniff example is 62.3MB/s, two orders of magnitude less. More enabled events would lead of course to more traffic.

3.9 Monitoring Bandwidth Optimization

From the previous sections of this chapter it is clear that sometimes the required monitoring bandwidth can be very low, even below what a single slot reservation can offer to an IP. This section presents a method, together with an example, which enables a more efficient monitoring bandwidth utilization. Two monitoring connections can be merged if their bandwidths or the combination of their bandwidths does not exceed the available bandwidth of a network link. In other words, if at least two NoC connections share at least one network link and if their respective bandwidths are less than the basic bandwidth of the link, these connections can be merged in at least one shared network link. This method uses sub-slot accuracy.

Figure 3.12 shows a block diagram of part of a system-on-chip where three low bandwidth monitors connected to MNI1, MNI2 and MNI3 are each utilizing a low bandwidth connection. Here, merely the monitoring network interfaces MNI1- MNI4 and the routers R1-R5 are shown. A first connection C1 extends from the monitoring network interface MNI1 via the router R1, R4 and R5 to the monitoring network interface MNI4. The second connection C2 extends from the monitoring network interface MNI2 to the monitoring network interface MNI4 via the router R2, R4 and R5. The third connection C3 extends from the monitoring network interface MNI3 to the monitoring network interface MNI4 via the router R3, R4 and R5. Accordingly, the link L7 and L8 between the router R4 and R5 and between the router R5 and the monitoring network interface MNI4 is used by the three connections. As an illustrative example, each of the connections C1 - C3 occupies 1/3 of the available bandwidth.

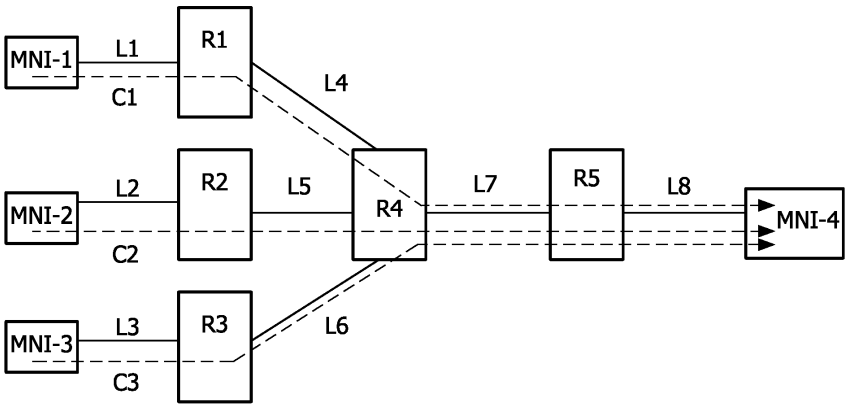


Figure 3.12: Original NoC featuring three monitoring connections

Figure 3.13 shows the corresponding slot table reservation for the part of the system depicted in Figure 3.12. In particular, the slot table reservation is shown

for different points along the time axis t. The usage or the reservation of each link is shown for the time slots S1 - S4. Those slots reserved for the first connection are indicated by C1. Those slots required by the second connection are indicated by C2 and those slots required for the third connection are indicated by C3. Those slots which are reserved but not actually induced are indicated by R in the figure.

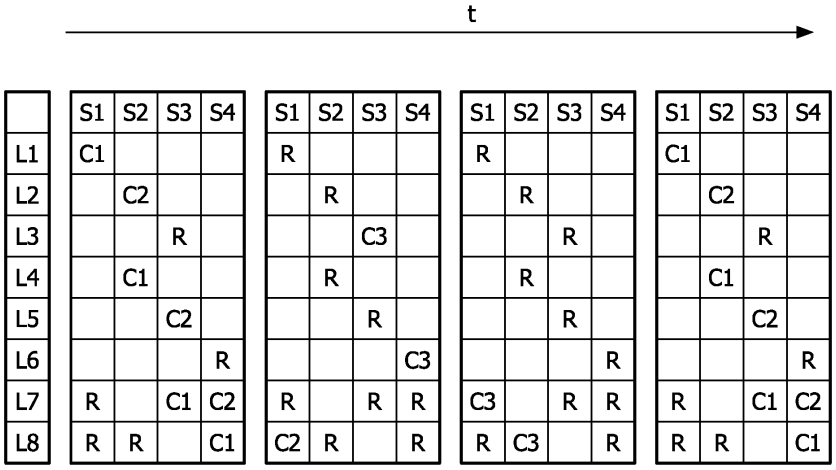


Figure 3.13: Original slot table revolutions

Figure 3.14 shows the block diagram of the system from Figure 3.12, where the three connections C1, C2 and C4 from the latter have been merged into connection C from the former. The connections C1-C3 are merged into a single connection C in the router R4.

Figure 3.15 shows a representation of a slot table reservation for the case when the three connections C1 - C3 are merged into a single connection C. This can be achieved by sharing the links L7 and L8 among these connections.

Besides the slot table, each of the monitoring network interfaces MNIs may maintain a minislot MS1 - MS3 of size 3. As the original connections only require 1/3 of the bandwidth available, only one packet is generated at 3 revolutions of the slot table. The minislot MS1 - MS3 contain the information for the monitoring network interface MNI in which slot table revolution it can place the data on the network N.

If the above-mentioned minislots MSl - MS3 are to be used effectively, the scheduling of the data transfer needs to be adapted. Guaranteed throughput flits may only stay for one flit clock in a router. Accordingly, as the links L7 and L8 are shared among the connections, i.e. the links are shared within the same slot, the time slot reservation in any previous links must be rearranged. This can clearly be seen if the slot time reservation table according to Figure 3.15 is compared to the table according to Figure 3.13. In the same way the destination MNI keeps

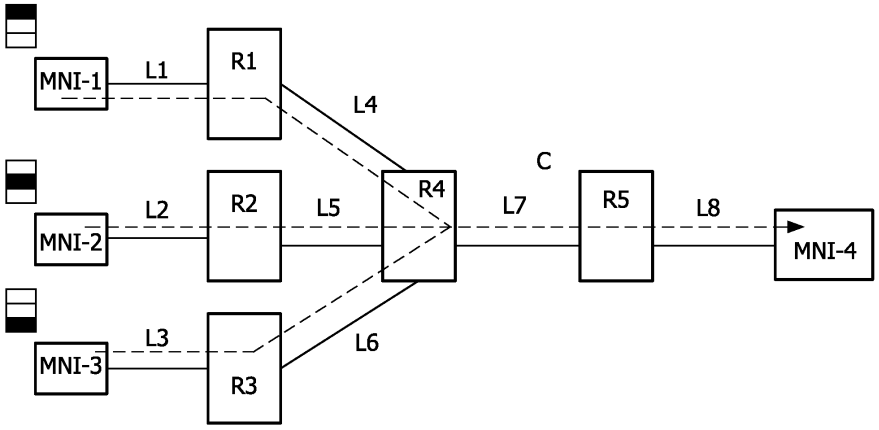


Figure 3.14: Original NoC featuring three monitoring connections with the associated minislots

a minislot in which it knows from which connection it receives data at each slot table revolution.

We conclude that minislots, or sub-slots are an efficient way to manage low bandwidth monitoring connection.

3.10 Conclusion

In this chapter, we have presented the concepts of a NoC monitoring service, the first one described in the scientific literature, and first described in [17, 18]. This monitoring service offers communication observability at run-time. It can be used for example for on-chip or off-chip application and system-level debugging, but also for run-time performance analysis. The monitoring service can be configured and used at arbitrary moments during run-time, offering increased flexibility.

The monitoring service is integrated in the NoC and uses the NoC communication services for configuration as well as for the event traffic. It can be instantiated automatically together with the NoC, saving design time. The monitoring service consists of monitors attached to NoC components, allowing easy scalability of the service, and monitoring service access points, the points where the monitoring service can be setup and monitored data can be accessed. The generic architectural concepts of the monitor feature a programmable modular design composed of sniffer, monitoring network interface and event generator, providing flexibility to target the service to the monitoring task at hand.

Proof of concept is achieved via implementation for the Æthereal NoC. The monitors model the monitored information in the form of timestamped events. We have presented our event model, a generic event taxonomy for NoCs and one

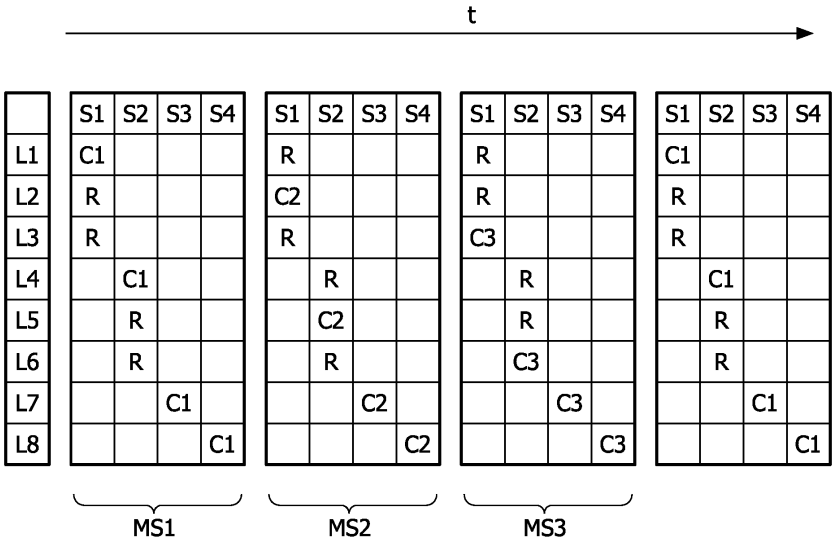


Figure 3.15: Original slot table revolutions with minislots

of the possible instantiations for the Æthereal NoC. Run-time programmability of the probes is achieved via memory-mapped configuration ports in the monitor. Traffic management is achieved by reusing the guaranteed communication services of the NoC. This provides potentially non-intrusive real-time monitoring.

The cost of the monitoring traffic is low, being several orders of magnitude lower than the bandwidth available in the NoC, for two realistic examples in a typical media processing SoC. The area cost for the NoC monitoring service targets a 15-20% of the total NoC area. Initial experiments on two different MPEG NoCs show that this target is realistic.

The generic concepts presented allow to retarget the NoC monitoring service to other NoCs. Almost all NoCs have network interfaces and routers as basic building blocks. They all provide some sort of communication services, and they support a certain design flow to instantiate these blocks and services. Furthermore, they all provide means of configuration or reconfiguration. In order to instantiate the proposed NoC monitoring service for another NoC, these concepts can be reused leading to a reasonable implementation effort.

In the next chapters we focus on cost analysis and optimization of the NoC monitoring service for the particular cases of transaction monitoring and performance analysis, as well as on the impact of such a monitoring service on the general NoC design flow. Since not all monitoring tasks require the same monitoring functionality, we will look into functionality vs. cost trade-offs.

Chapter 4

Transaction-based Monitoring

Being one of the main contributions of this thesis, this chapter presents the concepts of monitoring NoC transactions on chip at run-time. Transactions are the means of communication between IPs, and are comprised of messages. We want to be able to present a transaction-level view of the communication between the SoC components, in line with the communication-centric monitoring of NoC-based SoCs we advocate. With the generic NoC monitoring service presented in Chapter 3 we have shown that monitoring networks on chip at run-time is possible and can be feasible in general. This chapter focuses the discussion on the adequate abstraction levels for monitoring. It shows how the generic NoC monitoring service can be instantiated at design-time for run-time transaction monitoring, as well as run-time reconfiguration options.

The chapter starts with good motivation for transaction monitoring and an overview of the potential benefits of it in Section 4.1. Related work is discussed in Section 4.2. An introduction to transactions in general and to NoC transactions in particular is given in Section 4.3.

Section 4.4 details our proposed solution for monitoring the NoC transaction-based communication, concretely exemplified for the Æthereal NoC. The details regarding the Æthereal NoC and in particular the employed packetization scheme have already been presented in Section 2.3. We propose a specialized hardware monitor to replace the generic monitor of the NoC monitoring service. This monitor supports on-chip transaction reconstruction and several intermediate levels of abstractions needed for it, raising the abstraction level gradually from physical 'raw' to logical connection-based, transaction-based and abstract transaction event-based. We call the resulting monitoring system a NoC analyzer and the supported abstraction levels, analyzer modes. The NoC analyzer supports the two basic data transport scenarios of Section 3.7.3.

The four specific NoC analyzer modes are separately treated and the underlying monitor architecture is further detailed. Section 4.5 describes the raw mode of the NoC analyzer, the lowest granularity at which monitoring takes place. The

logical connection-based mode is detailed in Section 4.6 showing how a connection can be isolated from the intercepted raw data stream. Section 4.7 shows how we can reconstruct the transaction view (read and write actions from IPs) from the raw, low-level monitored data flowing through a single connection. Transactions are reconstructed in the form of composing messages. The raw data can be monitored at any router, a module which has no understanding of any notion of transaction. We show that transactions can be reconstructed regardless the way in which packetization has been done in network interfaces, covering all existing (and all possible) NoC packetization schemes. The obtained transaction information can be further abstracted in the form of events as shown in Section 4.8.

An analysis of the generated traffic for each of the analyzer modes is presented in the context of four realistic Æthereal NoC designs based on an MPEG codec, underlining advantages and potential problems for each of them. We further show that such monitoring is feasible area wise, as the transaction monitor supporting both GT and BE traffic classes is 0.026mm^2 in a 0.13μm CMOS technology, small even when compared with a corresponding 0.13mm^2 combined GT/BE NoC router realized in the same technology. The area and traffic implications of transaction monitoring including all the associated monitoring abstractions are presented in Section 4.9 together with an example of monitoring NoC configuration master activity in the connection-based mode. Section 4.10 investigated run-time reconfiguration strategies and reconfiguration times. The main conclusions of this work are presented in Section 4.12.

The core of this chapter including the main results and examples has been published as *"Transaction Monitoring in Networks on Chip: The On-Chip Run-Time Perspective"*; Calin Ciordas, Kees Goossens, Twan Basten, Andrei Radulescu, and Andre Boon; In Proceedings of the IEEE Symposium on Industrial Embedded Systems (IES 2006), October 2006. [19]

4.1 Motivation

Chapter 1 gave a general motivation on the necessity of monitoring and presented the related work. Knowing what transactions are and with the transaction monitoring at the heart of this chapter, we give now an overview of the problems that need or can benefit from transaction monitoring in special.

Complex systems require debugging, hence monitoring at different levels of abstraction. The use of monitoring is amenable to at least two key aspects: the desired level of monitoring and the possibility to tune, adjust and focus the monitoring system at the desired aspects at runtime. The former is merely a question of which task is driving the monitoring, in which case the desired level of monitoring can be provisioned for at design time. E.g., in transaction-based debugging, monitoring at transaction level is a prerequisite. The latter is more a question of what we can do when the desired level of monitoring is not known in

advance but must be adapted at run-time. In this case, several levels of abstraction must be provisioned for at design time, with the option to select them at run-time. Transaction level monitoring makes a good candidate in this case as one of the high levels of abstraction to be provided, realizing a real productivity gain by analyzing the data at such a high level of abstraction.

For example in *silicon debug and diagnosis*, which is supposed to solve all problems which may arise after a first silicon exists, the desired level of monitoring is the lowest possible, i.e., bit-level. That is the reason why the monitoring infrastructure in that case (in fact reused from the test infrastructure - therefore sharing the costs), the scan chains [96], is targeted at this. Of course even this would benefit from a higher abstraction level [99] at which debugging, or at least the triggering of the breakpoints, can be done, in order to reduce the debugging time.

If we extend the reasoning to complex multiprocessors with programmable cores, the *system level debug* has to take into account the hardware/software aspects, like how software interacts with the hardware, or the hardware modules with each other, making no assumptions whether the errors are in the software or in the hardware part. Clearly referred to as transaction-based debugging [43] the desired level of monitoring in this case is the transaction level, as IPs interact by means of transactions.

Obviously the silicon debug and diagnosis and the system level debug are interleaved activities, no clear separation existing. Therefore a high abstraction level for monitoring is desired but still with the fall back scenario of monitoring the lowest level possible, driving the debug quest towards multiple abstraction levels which are usable, tunable and selectable at run-time.

In the closely related area of *run-time assertion checking* [73], sometimes employed in complex systems, at run-time certain properties of the system are checked. One example is communication protocol checking. The desired level of monitoring is related to the level at which the protocol takes place, for example a request-response protocol for bus transfers. One desired level of monitoring is at the transaction level, being another application area for the generic transaction monitoring.

4.2 Related Work

There has been quite some work in transaction monitoring for busses. Monitoring is used for RTL level simulation environments but also for real SoC designs to trace the bus interconnect in its own environment. For NoCs, as far as we are aware, there is no related work (not counting our own publications) on the transaction monitoring topic.

DesignWare Verification IP for AMBA busses [87] from Synopsys, provides an easy way to verify SoC designs that make use of AMBA busses, compliant with AMBA 2.0 specification. It includes AHB and APB monitors for the AHB

and APB busses. The main features include: full transaction tracking and logging, protocol checking giving warnings and errors when protocol violations are encountered, and constrained random test generation used for automatic transaction generation to stress the maximum number of protocol combinations. The AHB monitor supports up to 16 masters and 16 slaves, being able to monitor all AHB transaction types. The APB monitor supports up to 16 slave devices. This is provided for RTL level simulations. However, there is no major obstacle in porting it on-chip.

Transaction-based debugging of PCI Express Embedded SoC platforms [11] in RTL simulations motivates that the grouping together of communication event sequences into abstract transactions provide a new way of describing interaction of modules within a SoC. Note that here abstraction refers to the way of ignoring pin/wire level activity. The authors argue that it is more convenient to analyze at the high abstraction level of a transaction. However, they also acknowledge that the detailed signal level activity is also required and monitoring has to be used do provide both the transaction level and signal level view when debugging.

Illustrative examples for porting bus monitors on-chip are industry's Silicon-Backplane Navigator [33] and OCP Navigator [34] from First Silicon Solutions. Both systems include an synthesizable on-chip instrumentation module which supports transaction monitoring. This module is synthesized in the SoC designs, FPGAs or ASICs. The OCP Navigator in particular offers the possibility of triggering on IO operations, on memory operations or on address values or ranges.

In busses, the level at which the monitoring is being conducted is usually the level where read and write actions are visible. Therefore, it is not needed to reconstruct reads and writes from an associated data stream, like it is the case for a NoC if the monitoring is done at routers. If we compare this with the NoCs, it is as if monitoring is performed between the NIs and the IPs connected to them, and not at the routers.

Furthermore, single monitors employed in busses can track the entire transaction history of bus activity. It is a mere task of logging these transactions in a centralized manner, using a circular buffer (a buffer that only keep the last X bus actions, the oldest being replaced with the new ones). In NoCs, multiple communication paths exist, requiring the use of multiple monitors, to be able to gain access to all potential transactions between IPs.

One rare combination of debugging, transactions and NoCs, clearly referred to as transaction-based debugging in the frame of system level debug is presented in [43, 99]. In this work, transactions are simply a higher level of abstraction at which triggering of breakpoints can be done, enhancing debugging. The work also shows the associated area cost to be around 5%. Based on the traditional stepping of the SoC, this approach is complementary to ours and relies on the presence of transaction monitors, and quickly discards sets of IPs which are not part of the problem.

4.3 NoC Transactions

4.3.1 Transactions

Before detailing transaction monitoring we have to clarify what transactions really are. Typical embedded systems contain a multitude of IP blocks like processors, DSPs, along with peripherals, memories and interface modules. They interact by means of transactions. Transaction examples are presented in Figures 4.1, 4.2, and 4.3. Normally, transactions take place between two or more IPs, involving both masters (M) and slaves (S). Transactions involving only slaves are normally excluded.

Figure 4.1 shows a read transaction between a single master and single slave. This is a typical way of communication between for example a programmable core as master and a memory as slave. The master issues a READ command towards the slave and as a response, the slave delivers the requested DATA. At this smaller level of granularity, for example READ and DATA, the atomic units are called messages, each transaction consisting of one or more messages. Note that this picture is conceptual, meaning only that the command READ is issued before the data; in real designs, for example busses have separate wires for commands and data. For the case of Figure 4.1, the number of messages involved in the presented transaction is two, and the communication is bidirectional from the master to the slave and from the slave to the master.

An even simpler transaction is depicted in Figure 4.2, where the communication is unidirectional, the master sending to the slave the command WRITE, followed by DATA. The slave is not expected to react.

The last example, shown in Figure 4.3, is an acknowledged or tagged write. The master sends the WRITE command to the slave, shortly followed by the DATA. After reception, the slave reacts by sending an ACK back, called a tag. In general, the ACK contains a means of identifying the last properly received message, for example a sequence number. In this case, communication is again bidirectional.

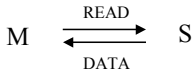


Figure 4.1: Read Transaction

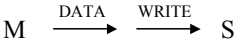


Figure 4.2: Write Transaction

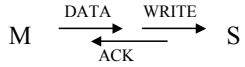


Figure 4.3: Acknowledged (Tagged) Write Transaction

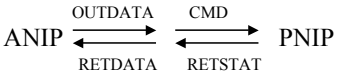


Figure 4.4: Æthereal Transaction Composition

4.3.2 Æthereal NoC Transactions

We have to consider now that a network on chip exists in between the master and slave IPs from the before mentioned figures, while the IPs are connected to the NI ports. In the Æthereal NoC terminology, a transaction is always between an active network interface port (ANIP) and one or more passive network interface ports (PNIP), as the transactions take place over connections and connections can involve two (in the case of a simple connection) or more network interface ports (in the case of a multicast or narrowcast connection); Æthereal NoC details have been presented in Section 2.3. The ANIP resembles the genuine master (M) and the PNIP the genuine slave (S) from Figures 4.1, 4.2 and 4.3. As the transactions take place over connections, transactions are pipelined between a single master-slave pair. Transactions are composed of a request and possibly a response.

In order to match the transaction-based communication model of IPs, the Æthereal NoC proposes the use of four message types as presented in Figure 4.4: CMD (a command message), OUTDATA (a data message), RETDATA (a data message) and RETSTAT (a status message). It implements a two phase request-response protocol. As bidirectional communication between IPs is a must, the messages are divided between outgoing (CMD and OUTDATA) and response (RETDATA and RETSTAT) messages. Note that the presence of all message types in a transaction is not required. The messages are further referred in this work as request and response messages. To match this setup with, for example, the generic simple write transaction of Figure 4.2, we must match the WRITE and DATA messages with the CMD and OUTDATA messages from Figure 4.4 respectively.

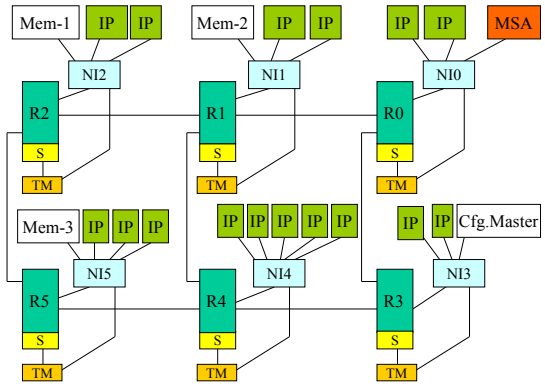


Figure 4.5: 2x3 MPEG Codec with NOCMS

4.4 NoC Analyzer

4.4.1 General Setup

As an example for our transaction monitoring work we have used the Æthereal NoC detailed in Section 2.3. The concepts presented in this chapter, however, are more general and can be reused for other NoCs, as they rely on the shared features of the NoCs as described in Chapter 2.

For readability, we summarize the relevant details of the Æthereal NoC. Transactions (reads and writes) are performed on connections. One transaction comprises one or more messages. Messages are differentiated as request and response messages. The NIs convert these messages into packets, by chopping them into pieces of a maximum length and adding a header to each of these pieces. Packets may be of different lengths. Packets are further split into flits, the minimum transfer unit between hops. The concrete message, packet and flit formats as well as examples of those are detailed in Section 2.3.

For the basic access to NoC flits, we have used the NoC Monitoring Service (NoCMS) detailed in Chapter 3. Our specialized transaction monitor replaces the generic event generator. Figure 4.5 presents an MPEG codec with a 2x3 NoC with its NoC Analyzer. All routers of the NoC example from Figure 4.5 have a transaction monitor (TM). The general process works as follows; the sniffer obtains the raw flits (bit-level) from the NoC components and passes this information to the transaction monitor. The transaction monitor performs local processing, specific to each of the analyzer modes. The monitors involved might operate in different modes. Single monitors might operate in different modes at different points in time while multiple monitors might operate in the same or in different modes at the same point in time. Monitors forward the results to the MNI. The MNI packetizes the result as payload and sends them over the network to the

Monitoring Service Access point (MSA), over a previously established monitoring connection, just like any other data in the NoC. The monitoring connections can be BE or GT or a combination of them, as presented in Section 3.7.2.

Throughout this chapter we make the assumption that we use the same NoC for the resulting monitoring traffic as well as for the user traffic because this solution allows a logical, dynamic partitioning of NoC resources; resources can be used for monitoring when needed, and freed when not. Note that this fact does not influence the techniques used to achieve multiple abstraction levels, the decoding of higher level protocols (transactions) from the raw data stream, or the transaction monitor design and cost. Alternative architectural options like a separate interconnect only for monitoring or just extra resources reserved only for monitoring may be used as well, as later described in Section 6.3 of this thesis.

For the experimental part we have implemented and used both *memory-mapped* and *streaming-data* scenarios as described in Section 3.7.3.

4.4.2 Multi-level abstraction capability

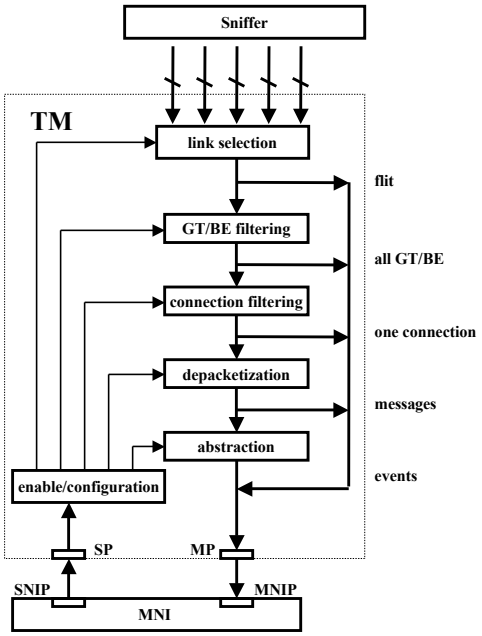


Figure 4.6: Transaction Monitor Architecture

The NoC analyzer has to be able to check the functional details of user traffic from the observed router link data, at different levels of abstraction. For this, the transaction monitor architecture is defined as a set of five successive pipelined

filters.

A schematic of the transaction monitor's internal architecture is depicted in Figure 4.6. All the router links are sniffed by the sniffer which provides this info to the link selection block. In the link selection block, one link is selected for further analysis. Then, there are *four transaction monitor processing blocks*. One or more of these blocks can be enabled and configured through the *enable/configuration block*. Two ports connect the transaction monitor to the MNI, one slave port (SP in Figure 4.6), for programming the transaction monitor, and one master port (MP in Figure 4.6), for sending the transaction monitor data to the MSA.

Transaction monitors are programmed using memory-mapped I/O, by means of write transactions. This is similar with the way NIs are programmed by means of write transactions, as explained in Chapter 3. Each of the processing blocks corresponds to one of the analyzer modes that we have identified. Each analyzer mode is detailed in the following sections.

4.5 Raw Mode

In the raw mode, the analyzer provides full observability on all bits passing a certain physical link. The desired link can be selected from all router links at runtime through the enable/configuration block. The link selection block provides at its output all flits passing one link. These flits can be part of different connections as TDMA is used for every link. The flits can be directly forwarded to the MNI or passed as input to the GT/BE filtering block. Looking only at the flit structure and not beyond, we can do only limited filtering. Local filtering is possible based on the traffic category. For the Æthereal NoC, we are able to filter GT or BE traffic from the raw flits. This is made possible by the 2-bit sideband information of each flit which specifies whether the flit is GT or BE. See Figures 2.7 and 2.8 from Chapter 2 for a single flit structure, and a packet comprising a sequence of flits. The resulting flits are forwarded to the MNI. The MNI packetizes the flits as payload of a write transaction (memory-mapped scenario) or a data stream (streaming-data scenario).

A potential problem may arise: assuming for a certain link that utilization is very high, and that we do raw sniffing, the sniffed data has to be sent over a connection to the MSA. Due to the packetization overhead (the packet headers added to the useful sniff payload), the total can be more than the physical link bandwidth, making the transport of the sniffed data impossible. Filtering whether the traffic is GT or BE can alleviate this problem in certain cases, but not always solve it. As an area expensive alternative, the use of a separate interconnect properly dimensioned may be employed.

Note that the transaction monitor in the current setup can select for monitoring a single physical link per monitor, from all the links that are sniffed from the router.

The raw mode is useful in case all details of the flits are important to be exam-

ined. Bit-level details can be inspected. In practice, it works for low bandwidth connections, or for short snapshots of high bandwidth connections.

4.6 Connection-based Mode

As sending the raw sniffing results to the MSA cannot always work in practice, a more advanced mode is needed. In the connection-based mode, the analyzer provides full observability on all bits of a selected connection, raising the abstraction level from physical raw to logical connection. The transaction monitor must allow further filtering of the sniffed data, besides the traffic class (e.g. BE/GT) in order to reduce the traffic from monitoring probe to MSA. All the NoC user traffic goes over connections, and connections share NoC links based on a TDMA scheme. Filtering of the sniffed data is possible if a certain connection can be identified from the other connections sharing the same link.

The connection filtering block of the transaction monitor uses as input the output of the GT/BE filtering block, and therefore all the link traffic of the selected traffic class. Connection identification can be done for the Æthereal NoC by means of the queue identifier and path; this pair uniquely identifies a connection. Both have to be programmed in the transaction monitor using the transaction monitor's slave port SP. Both the queue identifier and path can be found in the header of packets, see Figure 2.7. A packet header can be identified by the 2-bit sideband information of each flit. Furthermore, a packet header is always the first word in the flit. Once a header is intercepted the queue identifier of the destination queue in the NI, and the path to that NI, can be extracted from the header. The transaction monitor must have the desired connection set and compare the value stored locally with run-time values from the headers. Once a match is found, we have identified a packet belonging to the desired connection. The resulting packets are forwarded to the MNI.

Note that the proposed transaction monitor enabled in the connection-based mode is in the current setup able to monitor only a single connection at a given time; the connection must pass the link previously selected for monitoring. However, the extension to support the monitoring of multiple connections at a given time is straightforward.

The connection-based mode is useful in case all details of a certain connection have to be examined, e.g., packet headers or connection utilization. The previously mentioned problem of exceeding the physical link bandwidth is still possible although this would require extraordinary circumstances, e.g., all slots being reserved for a single connection. This is unlikely in a realistic scenario, and this mode is feasible in practice.

4.7 Transaction-based Mode

In the transaction-based mode, the analyzer raises the abstraction to transaction level and provides bit-level full observability on transactions over a certain connection. This implies full observability of all transaction components, which are messages. A full transaction may involve a single message (e.g. a single request) or multiple messages. The case of transactions using multiple messages on multiple connections, involves the combination of data from multiple transaction monitors or from one transaction monitor at different points in time, which can for example be done at the MSA.

Being able to identify all the flits of a certain single connection, the next step is the capability to identify the messages belonging to this connection. This allows to see, from within the NoC, when a write or a read message has been issued and from where or to which of the IPs or memories, providing a transaction level view.

The main problem is how to identify the messages. Within a data stream belonging to a single connection, it is difficult to detect the start of a message, because at NIs, and for the Æthereal NoC in particular at NI kernels, messages are considered payload and are packed in packets without any alignment. The routers, where the monitoring is done, have no notion of messages. A packet may contain a single message, part of a message, or parts of multiple messages. The last option also accounts for the particular case where multiple complete messages are packed within the same packet. It is therefore complex to see where a message starts just by looking at a packet.

In the general case, message identification requires depacketization, a procedure usually done at the NI at the receiving side, at every slave NI port (SNIP). Hardware modules for depacketization are available for the Æthereal NoC. These modules assume that the first packet over a connection carries the first message header, immediately after the packet header. From there they only count the number of words in the received message, knowing that after completion of the message there is a new message header.

Having detected the start of the message, the rest becomes simple. In the majority of the existing NoCs, the size of the message is coded in the first word of the message, the exact place depending on the exact protocol message format. So, if the message start is detected, the rest of the message can be obtained by counting the words in the following sniffed flits belonging to the same connection, till the message size is reached. Counting of the message words does not take into account the headers of the packets involved, which are discarded during depacketization. However, the main difficulty is detecting the start of the message.

Related to the packetization schemes, there are three possible situations, covering all existing NoCs, see Figure 4.7, with regard to alignment of packet header and message header, each of the situations having pros and cons regarding the NoC design:

(A) *The NoC does not distinguish between messages and packets, message/packet*

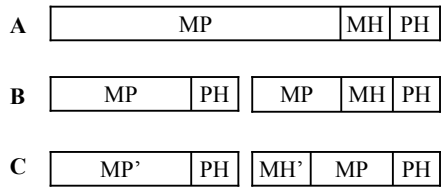


Figure 4.7: Message-packet alignment

correspondence is one to one, see Figure 4.7A. As previously explained, a packet header (PH) can be easily identified. In this case, we have also identified the message header (MH), which follows the packet header, and the message decoding can start. From the NoC point of view, this is a simple packetization scheme, but it may require long packets. Therefore, it may not fit well with the TDMA scheme, and may have partially empty packets.

(B) *The message header is aligned with the packet header but the NoC splits messages in multiple packets, see Figure 4.7B.* Even if a packet header can be easily identified, as message/packet correspondence is no longer one to one, it is important to know with which packet header the message header is aligned. Some packet headers are followed by message headers, others are followed by parts of the message payload (MP). This situation fits well with the TDMA scheme, but may have partially empty packets.

(C) *The NoC does not align messages and packets, see Figure 4.7C.* This is the most general and difficult case and is used by the Æthereal NoC. A message header can be anywhere in the payload of a packet. This is the most efficient packing of messages in packets and it is good for TDMA.

There are at least two solutions to solve the message identification problem described under items B and C:

(1) *One solution is to explicitly specify the message boundaries.* This means that each packet specifies whether it contains the start of the message and where in the packet is the message header. The presence of a message header in the packet and its offset can be coded either in the packet header itself or in the sideband information in the form of a control bit. This can be enforced by adapting the master NI port (MNIP) to include this information in the header or sideband information, which is not difficult because the NI has knowledge of the message start. Using this solution may require design modifications of the NI.

(2) *A second solution is to make sure that we can monitor the first packet going over the connection, and all the following flits belonging to the same connection.* In this case, we can continue identifying messages. This is because the

first packet will contain a packet header immediately followed by a message header and from there we can keep accounting for the following messages like the SNIP is already doing for the user traffic. This can be enforced e.g. by setting and enabling the transaction monitors before the connection is used. The advantage of this solution is that it requires no modifications of the NoC components.

For our experiments with the Æthereal NoC, we have enforced the second solution (2). We have made this choice because the Æthereal packet header does not contain information about message headers. The potential drawback of this solution, the fact that the transaction monitors must be enabled before the actual monitored connection is set up, is considered acceptable. Our solution requires a strict correlation between the (re)configuration moments of the NoC and the configuration of the NoCMS transaction monitors. It is possible to apply this solution for the Æthereal NoC, because our transaction monitors are run-time configurable by means of MMIO write operations, and because precedence of the transaction monitor configuration in front of the user connection configuration is enforced. The transaction monitors are configured before the actual connections are configured, being able to sniff all the data from a connection starting with the first flit of the first packet. In case of a NoC reconfiguration, it is again possible to reconfigure all the transaction monitors at the beginning of the reconfiguration when the rest of the connections are not yet configured.

The traffic introduced by the analyzer transaction-based mode is lower than in the connection-based mode as packet headers are removed in transaction reconstruction, when converting from packets to messages. This is done in the depacketization block of the transaction monitor, in fact a reused SNIP from the Æthereal NoC.

By identifying messages, local filtering of messages per connection is possible; all these options can be added to the depacketization block. For example, filtering of only write or read messages, or filtering of certain address range writes can be done, handy for debug purposes.

Note that in the current setup a single transaction monitor can track messages over a single connection, as a consequence of the previous mode which was able to gather the data stream of a single connection only.

The transaction-based mode is useful when all details of transactions or transaction components are required to be inspected. This is especially useful when inspecting IP to IP communication. However, details regarding packetization, like the content of packet headers, are no longer visible.

4.8 Transaction Event-based Mode

In the transaction event-based mode, the analyzer provides full observability on relevant transaction features or components and abstracts other irrelevant transaction features. Being able to identify messages over a certain connection is indeed

very useful. However, not all of this information is always needed for getting the picture of what is really going on in the NoC. Therefore, the abstraction level can be raised; whenever a transaction component is sniffed, a transaction event can be generated. E.g., a transaction event can state what was the command, address and the number of words in a message. A write message at address #0000 with 10 words of payload has a total of 12 words for the entire message. All this can be abstracted in an event of two words containing only the relevant features (command, address, nr. words), getting rid of the irrelevant (for this example) 10 words of payload. As another example, a transaction event could be generated only if certain value is written to a certain address.

Local filtering of relevant features of transactions is done in the abstraction block of the transaction monitor. The traffic introduced is lower than in the transaction-based mode as we get rid of packet headers and irrelevant transaction features by means of event abstractions.

Table 4.1: Comparison of analyzer modes

Mode	***TM capability***	***Filtering***	***Potential pb.***
raw	id. traffic	GT/BE	link bw.
connection	+id. connection	connections	link bw.
transaction	+id. msg.	messages	msg. start
tr. event	+event generation	msg. features	msg. start

Table 4.1 summarizes the capabilities of the analyzer, showing a comparison between all analyzer modes focusing on the capabilities built into the transaction monitor, the potential filtering and the potential problems in each of the modes.

4.9 Analysis

4.9.1 Implementation

Our final point is to prove that on chip run-time monitoring of NoC transactions is feasible in resource-constrained NoC designs. Therefore, we have investigated the area and traffic implications of our NoC analyzer and analyzer modes. For the experimental validation of our NoC analyzer, we have built a flit accurate SystemC model, and a cycle accurate synthesizable VHDL model of the transaction monitor. We have used these models in conjunction with the Æthereal NoC and design flow.

The placement of the transaction monitors at routers is a design time choice. Currently, the Æthereal NoC design flow of Section 2.3 has been extended to support monitoring in general, as further detailed in Chapter 6.

For our experiments all routers are instrumented using the monitoring-aware NoC design flow with transaction monitors, thus resulting in a fully probed NoC. For our traffic experiments with the SystemC models, we have used transaction

monitors supporting all four analyzer modes. For our area experiments with the VHDL models we have used transaction monitors supporting the first three analyzer modes (full transaction reconstruction without transaction abstraction). We use preestablished GT connections for each of the employed transaction monitors to transport the sniffed data from the transaction monitors to the MSA and for their run-time configuration. The application and monitoring traffic share the same NoC. For NoC design-time aspects related to monitoring, such as how to provision the monitoring requirements for NoC dimensioning or how to automate the insertion of monitors by means of a monitoring-aware NoC design flow refer to Sections 6.3 and 6.5 respectively.

To quantify the complete effects of monitoring, we had a look at four different SoC designs, using NoC mesh topologies, supporting combinations of several MPEG instances (from one to four) with a single audio instance consisting of sample rate conversion, MP3, audio-postprocessing and radio as presented in [62], using the memory mapped and streaming data transport scenarios of Section 3.7.3.

4.9.2 Area Analysis

The area overview of three probes, corresponding to the first three analyzer modes (raw, connection-based and transaction-based), realized in a 0.13μm CMOS technology is presented in Table 4.2. Since transaction monitors support both GT and BE traffic classes, a comparison is made with the area of an Æthereal arity 6, GT/BE router, presented in [41], which is 0.13mm^2 in the same 0.13μm CMOS technology.

Table 4.2: Area Impact

Probes	***area***(mm^2)	***comp. to router***
raw	0.020	15%
connection-based	0.024	18%
transaction-based	0.026	20%

The results show that offering raw data monitoring capability would require 15% more area compared to a single router, while connection-based capabilities would require around 18%. Full fledged on-chip transaction reconstruction is feasible at the cost of 20% of the router area. The probe area presented includes the configuration unit allowing to (re)configure the transaction monitors at run-time. Configuration includes the start and end time of transaction monitor activity, and the selection of the desired mode with the required characteristics. It also includes three words of internal storage.

The first three analyzer modes realize the transaction reconstruction, by decoding the transaction components, while the fourth one is doing the transaction abstraction. Therefore, the area cost of the probe supporting the first three modes is fixed, while the area cost of a probe supporting all four analyzer modes may

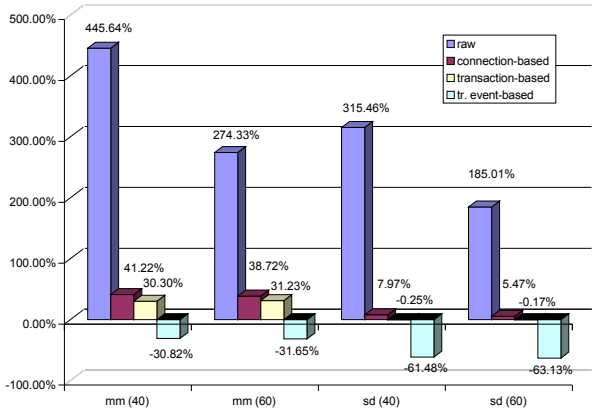


Figure 4.8: Monitoring traffic comparison

vary, depending on the transaction abstraction capability implemented. The area cost of transaction abstraction is left for future work.

Looking at the NoC level, on average, for our designs, the overall monitoring system adds 5% to the original NoC area (routers and NIs). This accounts for the transaction monitors area, and the increase in the number of NI ports with the corresponding buffers. These system-level area results are explained in more detail in Chapter 6.

In Chapter 3, we have targeted an area of the NoCMS of roughly 15-20% compared to the initial NoC area. The initial estimations for the watchpointing case of the same chapter have shown an average of around 20%. The obtained results of this chapter for the concrete case of transaction monitoring, with the restrictions presented for each of the analyzer modes, are much better than the mentioned estimations, showing that an efficient NoCMS may be constructed. The remaining area up to the acceptable total of around 15% may be used to ease these restrictions, e.g. by having the capability to select two connections that can be monitored simultaneously.

4.9.3 Traffic analysis

A traffic comparison for all analyzer modes is presented in Table 4.3. The monitoring targets are two user connections of 40 MB/s and 60 MB/s respectively. The monitoring was done by activating a single transaction monitor for each of the target connections, and configuring it in turns for each mode. Note that the traffic figures of Table 4.3 correspond to a single transaction monitor and are independent of the number of transaction monitors present in the NoC.

The IPs connected to the network operate in the memory mapped scenario

Table 4.3: Monitoring traffic details

		MB/s	MB/s
user data			
user payload	P	40	60
NoC payload	P+C+A	59.84	89.68
NoC traffic	P+C+A+H	65.20	95.04
raw			
debug payload	P'= $\sum$(P+C+A+H)	255.28	
(mm) NoC payload	P'+C'+A'	340.37	
(mm) NoC traffic	P'+C'+A'+H'	355.76	
(sd) NoC payload	P'	255.28	
(sd) NoC traffic	P'+H'	270.88	
connection-based			
debug payload	P'=P+C+A+H	65.20	95.04
(mm) NoC payload	P'+C'+A'	86.93	126.72
(mm) NoC traffic	P'+C'+A'+H'	92.08	131.84
(sd) NoC payload	P'	65.20	95.04
(sd) NoC traffic	P'+H'	70.40	100.24
transaction-based			
debug payload	P'=P+C+A	59.84	89.68
(mm) NoC payload	P'+C'+A'	79.79	119.57
(mm) NoC traffic	P'+C'+A'+H'	84.96	124.72
(sd) NoC payload	P'	59.84	89.68
(sd) NoC traffic	P'+H'	65.04	94.88
tr. event-based			
debug payload	P'=E	19.95	29.89
(mm) NoC payload	P'+C'+A'	39.89	59.79
(mm) NoC traffic	P'+C'+A'+H'	45.04	64.96
(sd) NoC payload	P'	19.95	29.89
(sd) NoC traffic	P'+H'	25.12	35.04

while the transaction monitors can operate in either memory mapped or streaming data scenario. The user payload, e.g. 40 MB/s, denoted with P in Table 4.3, represents the application data. To this user data, commands (C) and addresses (A) still have to be added before the NIs; the value is shown as P+C+A in Table 4.3. This represents the actual payload for the NoC. Taking into account the slot allocation, due to packetization, headers are added in the NIs to the actual payload, P+C+A+H in the table. P+C+A+H is the traffic going through the network.

When monitoring in the *raw mode* the sum of P+C+A+H for all connections passing the monitored link becomes the actual payload P' for the debug connection. In our experiment, the two monitored connections share the same monitored link, together with another 60MB/s GT user connection, which explains the large payload in Table 4.3. Note that the raw traffic is only directly related to the

monitored link utilization and not to the size of a particular connection.

When monitoring in the *connection-based mode* P+C+A+H becomes the actual payload P' for the debug connection. When monitoring in the *transaction-based mode* P+C+A becomes the actual payload P' for the debug connection, because the headers are removed in this analyzer mode, assuming no further message filtering which is the worst case for this mode. When monitoring in the *transaction event-based mode* the actual payload E is computed in this example by abstracting the 6-word messages (P=4 words, C=1 word and A=1 word) used on this connection in 2-word events. In general, it may vary depending on the abstraction capability of the event-model.

When using the *memory-mapped* scenario, the (mm) tag in Table 4.3, for the transport of the monitored data, new addresses and commands are added to the previously explained payload P', denoted P'+C'+A' in Table 4.3. New packetization is done, and new headers (H') are added to this, getting the final debug connection traffic to P'+C'+A'+H'. When using the *streaming-data* scenario, (sd) in Table 4.3, for the transport of the monitored data, new addresses and commands do not need to be added to P'. Headers are added though, getting the final debug connection traffic P'+H'.

Figure 4.8 presents the NoC analyzer traffic for all analyzer modes in percentages, compared to the initial NoC traffic of 65.20 MB/s and 95.04 MB/s for the 40MB/s and 60 MB/s connections respectively. This is done for both the *memory-mapped* (mm) and *streaming-data* (sd) scenarios. In the raw analyzer mode, the numbers show that there is a tremendous traffic on a link, several times bigger than the considered user connections. Note that the raw mode shows the overall link traffic comprising three connections in total. In the connection-based mode the numbers show that we introduce in the NoC new traffic, bigger than the monitored traffic. In the (mm) scenario this is 41% and 39% more, while in the (sd) scenario it is only 8% and 5% more. In the transaction-based mode, we introduce in the NoC new traffic around 30% higher than the monitored traffic in the (mm) scenario, and comparable but slightly lower than the monitored traffic in the (sd) scenario. In the transaction event-based mode, we introduce in the NoC new traffic, lower than the monitored traffic, which is always the case for sufficiently abstract events. This represents a real gain over the monitored connections. The gain amounts to around 30% and 60% in the (mm), respectively (sd) scenario, in the concrete example.

As expected, traffic wise it is a good idea to use transaction abstractions when doing online transaction monitoring, in case other details are not of interest. Combining them with a *'streaming-data'* scenario is beneficial. Monitoring at the lowest level of detail would produce the most load for the NoC.

4.9.4 Debugging the NoC Configuration Master

As previously mentioned, the Æthereal NoC can be configured at run-time. In the current Æthereal setup, a centralized programming module, e.g., an ARM

processor, is doing all the configuration work. This centralized module is called the *Configuration Master*, see the Cfg. Master IP in Figure 4.5. As support for NoC debug, it is important to see what the Configuration Master is doing at run-time, as all the inter-IP communication is going over connections. The observed behavior can then be compared to the expected behavior in order to catch possible errors.

The Configuration Master uses BE packets to configure the NIs. Our NoC analyzer can monitor configuration in the connection-based mode. For this, we take advantage of the Æthereal configuration details. All NIs have a port, with qid 0, called a configuration port. Through this port the NIs are configured at run-time. The observation of the Configuration Master requires a single probe. Therefore, at run-time, a single transaction monitor was activated in one test NoC, depicted in Figure 4.5, namely the transaction monitor attached to router R3. This transaction monitor monitors the link between NI3, where the Configuration Master is connected, and router R3. The transaction monitor was enabled in the connection-based mode, and was configured only with the queue identifier 0 and not also with the path. In this way, all the outgoing traffic from NI3 towards any destination with qid=0 is filtered, and then sent to the MSA. This amounted to 205 flits containing 529 (4-byte) words of configuration data for the entire reconfiguration of all NIs. In this way, all the Configuration Master behavior is observed. A single preestablished GT connection is used to transport the monitored data to the MSA. As another experiment, the transaction monitor was enabled in the transaction-based mode and the path was set. Transactions are only monitored over this path, corresponding to NI4 being configured, and this amounts to 24 write transactions.

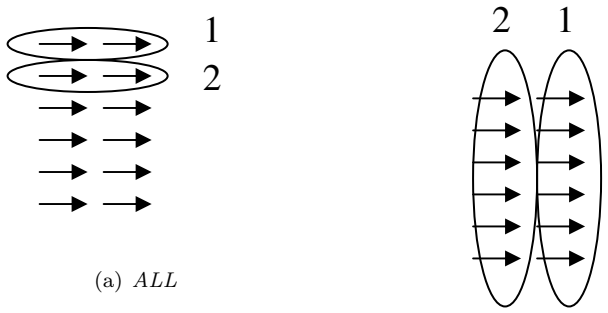
Due to the centralized programming of the NoC using a single Configuration Master in the current setup, one transaction monitor is currently sufficient to monitor NoC configuration. However, in the near future, distributed programming of the NoC may be another option. In order to keep up with this option multiple Configuration Master monitors will have to be employed (or activated), one for each Configuration Master.

4.10 Run-time reconfiguration

Up to 64 bits of configuration data are required by a single transaction monitor during the configuration. This can be done using either one 64-bit DTL-MMBD write operation or two 32-bit DTL-MMIO write operations. For the DTL details see [75]. In the first case the configuration data can be packed into a 32-bit (one word) command (C), 32-bit (one word) address (A), 64-bit (two words) payload (P) write message which would enter the MSA connected NI. In the second case the same data is packed into two (C,A,P) write messages. Note that in general the amount of configuration data required per monitor may differ with the monitor type, e.g. it may not be the same for a transaction monitor and a performance

monitor, potentially resulting in more/less required write messages per monitor. However, the same run-time configuration policies and techniques may apply.

It has been previously mentioned that transaction monitors can be (re-)configured at run-time by means of write transactions. As a separate experiment we have looked at complete monitoring service configuration and evaluated the configuration options and the resulting configuration times. For this we have used the example MPEG design case using a 2x3 mesh topology, which was fully probed, resulting in six transaction monitors. We have used a centralized monitoring service with one MSA. We have used a slot table size of 128. Each transaction monitor uses a dedicated connection to the MSA. We have investigated both using the existing GT and BE communication services for monitoring system configuration. When using GT connections we have reserved a single slot for each monitoring connection.



(a) *ALL*

(b) *FIRST*

Figure 4.9: Multiple configuration messages

We have tried two monitoring system-wide policies for configuration. One policy is based on simple write messages, which are not acknowledged by the transaction monitors. The total configuration time in this case is the time elapsed from the sending of the first message from the MSA to the first transaction monitor to be configured until the last received message at any of the transaction monitors. Note that the last message sent from the MSA may not be the last received message at the transaction monitors.

A second monitoring system configuration policy is based on acknowledgements. In this case, a 32-bit acknowledge is sent back from each of the transaction monitors upon reception of a configuration message and completion of the local transaction monitor configuration. The advantage of the second method is that the MSA knows when the monitoring system is configured. In this case the configuration time is the time elapsed between the time when the first message is sent from the MSA and the last acknowledgement is received at the MSA. Note

Table 4.4: ALL-FIRST

mpeg	*ALL GT(ns)*	*ALL BE(ns)*	*FIRST GT(ns)*	*FIRST BE(ns)*
WR64	1212	78	1212	78
2xWR32	3378	174	1980	174
WR_ACK64	1832	152	1832	152
2xWR_ACK32	4136	224	2600	224

that in general the acknowledgements are not received at the MSA in the sending order of the configuration messages from the MSA.

In the case of multiple configuration messages required for the same transaction monitor (e.g. two write messages) we have used two options. One option is to send all the messages for the same transaction monitor first then followed by all the messages for the second transaction monitor and so on. This is graphically depicted in Figure 4.9(a) and further referred as the ALL case. A second option is to send the first message to the first transaction monitor followed by the first message to the second transaction monitor, and so on, and only send the second message to all the transaction monitors when all the first messages for all transaction monitors have been sent, and so on. This is illustrated in Figure 4.9(b) and further referred to as the FIRST case.

Table 4.4 show the configuration time experimental results. In the first column we show the use of the write messages, where WR64 and WR32 corresponds to a write with 64 bits or 32 bits of payload; 2xWR32 shows that two write messages are used for the configuration, while the presence of ACK shows the presence of a 32-bit acknowledgement in the configuration process for a single transaction monitor. The table shows that using BE for configuration is several times faster than using GT as the configuration data does not have to wait for the reserved slot. This is expected because as soon as there is an empty slot or reserved but not used slot the BE configuration would sneak on the link. When using multiple configuration messages over GT monitoring connections for the same probe, it is more efficient to do it the FIRST way than to do it the ALL way. This is because in the ALL way the second configuration message for the first probe cannot be sent to the corresponding NI queue until there is space in the queue, thus delaying the first configuration message for the second probe. Table 4.4 finally shows the expected result that the use of acknowledgements increases the configuration time. Note that when using GT connections for configuration, the results in Table 4.4 do not account for the time required to set up these connections.

The results show that the run-time configuration is feasible for realistic cases, and the configuration time required for it is acceptable.

4.11 Transaction Monitoring Optimizations

Employing Multiple Monitors

Being able, in the previously described transaction monitor, to isolate and distinguish flits, messages, transactions and events, provides new options for efficient monitoring of data traffic inside the NoC.

Multiple monitors can be provided along a connection's path, wherein each monitor can sniff one or more connections. Particularly, in the cases where it is difficult and sometimes impossible to setup high bandwidth monitoring connections in a network-on-chip NoC resource constrained environment, it is advantageous to schedule more debug connections with lower-bandwidth; e.g. this can happen when monitoring in the raw mode. The sniffing of a high-bandwidth connection can be performed by providing multiple monitors on paths of connections or communications, wherein each of the monitors merely partly sniff the respective connection. Data is then sent over multiple monitoring connections to a central MSA where the complete connection trace is restored from the partial traces from the monitors.

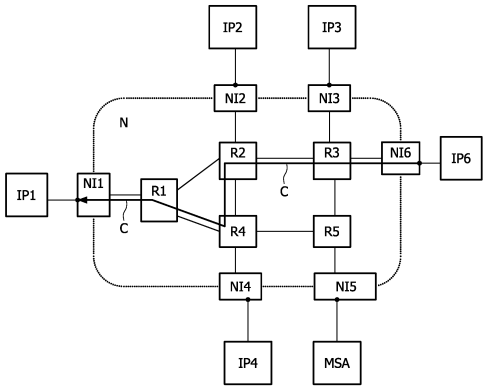


Figure 4.10: NoC Monitoring Service

Figure 4.10 shows an example NoC, with several IPs connected to it and the example connection C, established between NI6 and NI1, serving for the communication of IP6 with IP1. For example purposes we assume this user connection to have a bandwidth of 100MB/s. A monitoring service access unit is provided as a central access point for monitoring the data.

Figure 4.11 shows the same SoC as in Figure 4.10. The only difference is the provision of a monitor Pl which is attached to the router R2. The monitor is able to sniff data like flits, messages, transactions or other granularity depending how much intelligence is built in the monitor, as previously explained in this chapter. Here, as an example the monitor can sniff flits. The output of the monitor Pl is

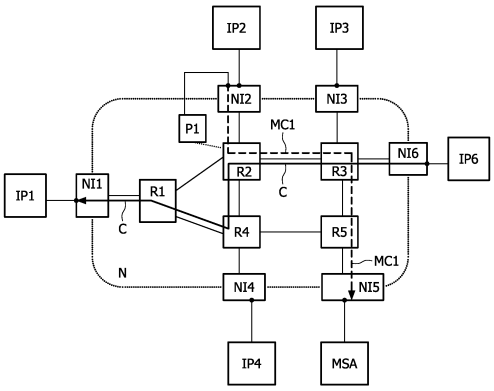


Figure 4.11: NoC Monitoring Service

coupled to the network interface NI2. The monitor Pl is able to sniff all flits of the connection C when the flits pass the router R2. The sniffed flits are passed to the network interface NI2 and is forwarded to the monitoring service access unit MSA. A monitoring connection MC1 from the second network interface NI2 to monitoring service access unit MSA is required in order to transport the data of the monitor, i.e. the monitoring data. The bandwidth BD of the monitoring connection MC1 is bandwidth B + a packetization overhead added by the network interface NI2, i.e. the bandwidth BD is at least of bandwidth B, e.g. BD=120MB/s.

Although a router link, for example for Æthereal routers, may offer a raw bandwidth of 2GB/link/s, part of this bandwidth may be already used by the existing mapping of user connections on the network on chip NOC. If the links (NI2-R2) and (R3-R5) can only allow a lower bandwidth connection than the bandwidth BD, e.g. 70MB/s, as the rest is in use due to the existing mapping of user connections on the network on chip NoC, the sniffing cannot be performed, as the bandwidth BD cannot be offered by the network on chip NoC on any single route from the network interface NI2 to network interface NI5 (NI5 connects the MSA where the sniffed data must go).

Figure 4.12 shows the same SoC as in Figure 4.10, which now comprises multiple monitors, e.g. all routers can be instrumented with monitors, ensuring a full coverage of all possible connections or communication paths to be set up.

Accordingly, N monitors may be present on the path of connection C. Here, a first monitor Pl is attached to the router R2 and a second monitor P2 is attached to the router R4. Therefore, N monitoring connections are required, namely a first monitoring connection MC1 and second monitoring connection MC2 with the bandwidth BMC1 and BMC2. The first monitoring connection MC1 is provided for the data of the first monitor and the second monitoring connection MC2 is

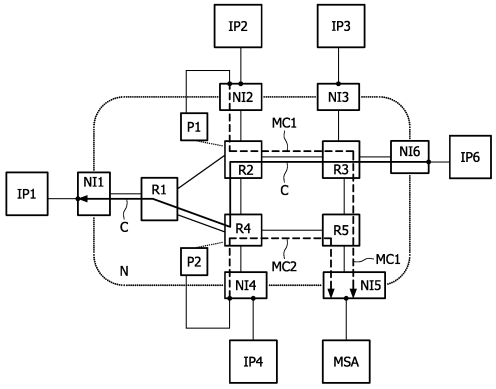


Figure 4.12: NoC Monitoring Service

provided for the data of the second monitor. The data from both monitors are sent to the MSA. Thus, the bandwidth previously required for sniffing can now be shared by these N or 2 connections, i.e. BD = BMC1 + BMC2, BMC1=60MB/s, BMC2=60MB/s. These two monitoring connections can be setup, if the resources are available.

As an example, each monitor can sniff one flit per slot. Therefore, the monitors comprise a counter, with respect to the number of flits which have passed the link. Each of the monitors can alternatively sniff a number of flits either in a balanced or an imbalanced way.

According to the balanced way the monitors Pl and P2 sniffs the same number of flits (evenly splitting the bandwidth required), e.g. monitor Pl sniffs odd flits and monitor P2 sniffs the even flits), as depicted in Table 4.5. According to an imbalanced way the monitors Pl and P2 sniffs different numbers of flits e.g. monitor P2 sniffs 1 flit after monitor Pl sniffs 2 flits, as depicted in Table 4.6.

Table 4.5: Monitoring Flits, Balanced Scheme

Flit counter	*P1*	*P2*
s1	Flit1	-
s2	-	Flit2
s3	Flit3	-
s4	-	Flit4
s5	Flit5	-
s6	-	Flit6

The monitoring service access unit MSA reorders the flits by reading the flits according to the sniff schedule from the network interface NI it connects to, e.g. NI5 in our figure. The network interface NI5 requires two buffers, one for each

Table 4.6: Monitoring Flits, Unbalanced Scheme

Flit counter	*P1*	*P2*
s1	Flit1	-
s2	Flit2	-
s3	-	Flit3
s4	Flit4	-
s5	Flit5	-
s6	-	Flit6

connection (MC1 and MC2). The monitoring service access unit MSA reads in the first example 1 flit from each of the debug connections alternatively, and in the second example 2 respectively 1 flit. Hence, the entire connection activity can be reconstructed.

If more intelligence is added to the monitors, e.g. like in the case of our transaction monitors, the sniffing can be done per message, which is in turn reconstructed from flits. The counter will just count the number of messages passing the router. The monitoring per message can be done in a balanced or unbalanced way, similar to the flits scheme.

Generalizing, N probes may be placed onto the same connection. In this case, the bandwidth BD can se shared over N debug or monitor connections (utilizing all probes). Additionally a single monitor may have multiple connections with the monitoring service access unit MSA with different paths, in order to avoid a constrained link on the path from monitor to the monitoring service access unit MSA.

In principle, this method can be used in any interconnect, e.g. networks on chip, where resource reservations can be made for traffic, for related monitoring activities. The above described solution guarantees a distributed sniffing of a connection and the distribution of bandwidth required for sniffing that connection into multiple lower bandwidth connections, depending on the number of available monitors. Furthermore, the reconstruction of the original (sniffed) connection information at the monitoring service access point MSA now received from multiple (distributed) monitors. This is in particular advantageous as a working solution in the case of bandwidth constrains on certain links is provided, due to physical limitations of links or mapping of connections.

4.12 Conclusions

Networks-on-chip are a scalable interconnect solution to multiprocessor systems on chip and a suitable place to monitor the internals of a SoC at multiple levels of abstraction. NoCs transport data in packets which are fragments of transactions, such as read and write actions of IPs. For debug purposes in general, reconstructing transactions at run-time is essential. Run-time analysis of the NoC behavior

at transaction level makes the complete MPSoC easier to understand.

We have presented a NoC analyzer able to perform run-time NoC transaction monitoring. The proposed NoC analyzer alleviates the run-time observability problem by providing hardware transaction monitors able to work on four different levels of abstraction. They correspond to four analyzer modes, ultimately being able to on-chip reconstruct transactions from low-level monitored router data and abstract them to events. All of the analyzer modes can be enabled and configured at run-time. They match difficult debug situations, and are a valuable asset when debugging multiprocessor NoC-based SoCs.

In NoC monitoring, it is important to go beyond the raw low-level data (bits), to understand what data means (transactions). Due to nonalignment of packets and messages, it is generally difficult to (re)construct a transaction-level view from the data stream of a connection. We have conceptually shown how this problem can be solved for all existing NI packetization schemes. Thus our concepts can be reused for any existing NoC.

A transaction monitor for the most difficult packetization scheme was implemented at the cost of one fifth of the router area. A transaction monitor has an area cost of 0.026mm^2 in a 0.13μm CMOS technology, and for several MPEG/audio SoC case studies, the entire monitoring system adds an average of 5% to the NoC area, much better than the initial target of 15-20% and the initial estimations of Chapter 3. The latter is due to the specialization of the transaction monitoring system, whereas the estimates of Chapter 3 were done for a very generic system.

A traffic analysis of analyzer modes has been presented. The traffic introduced, compared to the traffic of the monitored connection, varies from a penalty of 41% in the connection-based mode memory-mapped scenario, to a gain of 63% in the transaction event-based mode streaming-data scenario. We have shown the versatility of our NoC analyzer by monitoring the NoC configuration master. Finally we have investigated various scenarios for run-time reconfiguration of the NoC monitoring system and shown that it is feasible for realistic monitoring cases.

This work on transaction monitoring merely provides the knobs and controls for establishing and tuning a structured transaction-based debugging method. The development of such a method is left for future work.

Chapter 5

Monitoring-assisted QoS

We have proposed a generic NoC monitoring service as a basis of this thesis. This chapter is one more link in the chain of proofs spread throughout this work regarding the generic character of this service. After the specific instance targeted for transaction monitoring derived in Chapter 4, this chapter presents a second NoCMS instance targeted to run-time performance monitoring, showing the versatility of our proposed NoCMS. The performance monitoring NoCMS is presented together with two brief case studies showing its applicability.

The chapter starts with introducing the motivation for NoC run-time monitoring in the context of NoC services and application level QoS. Related work is discussed in Section 5.2. A brief introduction to performance measures is presented in Section 5.3. The general setup of the proposed performance monitoring NoCMS is detailed in Section 5.4. It relies on a specialized, run-time configurable performance monitor connected to NoC routers able to track the number of units, e.g. flits passing the monitored links. The internal details of the performance monitor, including its block architecture, its monitoring capabilities and configuration aspects including the synchronous start of multiple monitors are presented. Section 5.5 presents an evaluation of the main performance monitoring bandwidth requirements and area costs, including a comparison with the transaction monitoring of Chapter 4. Two small cases studies showing the added value of the performance monitoring NoCMS are briefly explained in Section 5.6. The chapter ends with the main conclusions.

The author's contribution in the form of a NoCMS instantiated for performance monitoring has been presented in part as a secondary topic in: *"Congestion-controlled best-effort communication for networks-on-chip"*; Jan Willem van den Brand, Calin Ciordas, Kees Goossens, and Twan Basten; In Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE 2007), April 2007. [94]

The use of the performance NoCMS as input for an application QoS manager was detailed in: *"Mixed adaptation and fixed-reservation QoS for improving Pic-*

ture Quality and Resource Usage of Multimedia (NoC) Chips"; Milan Pastrnak, Peter H.N. de With, Calin Ciordas, Jef van Meerbergen, Kees Goossens; In Proceedings of International Symposium on Consumer Electronics (ISCE 2006), Jun 2006. [71]

5.1 Motivation

A general motivation on the necessity of monitoring has been detailed in Chapter 1. Chapter 4 refines the general monitoring motivation to the particular case of transaction monitoring, as part of the functional, system-level debugging, or even run-time assertion checking. This section offers an overview of the main performance monitoring beneficiaries.

One of the first performance monitoring beneficiaries is the run-time performance analysis. Used as a very general term, performance analysis is the process of interpreting and understanding the gathered performance monitoring data.

Run-time performance analysis and the associated performance monitoring are needed to understand what the system is currently doing, what it has done in the past and eventually to predict its future behavior. Part of performance analysis, generally referred to as performance debugging, is concerned with the question whether the existing NoC is currently achieving the performance requirements of the running applications, directly in terms of the application or indirectly in terms of NoC resources.

Gathering of such performance statistics over a period of time can show whether the response of the NoC to certain corner cases of the application can affect performance behavior; such performance anomalies can normally be detected in real-time systems. In general this is referred to as anomaly detection.

In a case not related with performance debugging or with anomaly detection, performance monitoring reflects whether the NoC can support or accommodate new applications and/or services and to what degree. This is based on the trends identified by monitoring and predictions of impact of the future application requirements impact on the performance. In more general terms this is related with quality of service (QoS).

5.2 Related Work

When investigating performance analysis for NoCs, it is a good starting point to look at internet-like networks. At lot of work has been done in this area for such networks, ranging from the identification of appropriate metrics, monitoring architectures, monitoring agents, and standardized monitoring services to performance visualization or network management. Some of the performance metrics are: round trip delay, packet loss, reachability, availability, and router and link utilization. Most of such monitoring data is collected by monitoring agents residing on the monitored entities, e.g. a host. One good place to start is the standard

simple network management protocol (SNMP) or the remote network monitoring specification (RMON) described in virtually all network related books.

In the related field of network management, network monitoring is widely used. The employed mechanisms for network monitoring can be classified into two categories [51]: end-to-end network monitoring and distribution monitoring. This is done according to the level of information that can be obtained from the monitoring. In an end-to-end monitoring approach, only the end-to-end properties between the sender and receiver of a flow are monitored. However, in the distribution monitoring approach, the quality of service distribution experienced by the flow in different network segments is monitored. This is needed to clearly locate the network segments (e.g. links) causing possible QoS degradation. In network monitoring works [51, 91], the focus is on the mere task of locating relevant network monitors that can meter the present flows in multiple points in their paths.

We have to point out here that monitors can be added in a real network at any time if desired, while this is not the case for a system on a chip. Our proposed NoCMS instance for performance monitoring fits in the distribution monitoring class, for networks on chip, although there is no conceptual limitation to extend this NoCMS with NI monitors for end-to-end monitoring.

A set of framework and performance metric definitions targeted at NoCs in particular is presented in [36]. The intention of the authors is to present a generic and common set of performance oriented NoC metrics as a first step towards network-on-chip benchmarking. The end purpose is to be able to fairly compare very different NoCs based on this framework with the associated metrics. The performance metrics defined are mainly suited for end-to-end NoC properties. This work is focused towards the design phase of NoCs, mainly for simulation environments where performance tests can immediately reveal performance problems, or the impact of the design decisions on the NoC performance.

In a rare combination of NoCs and run-time monitoring [69], the use of end-to-end monitors is proposed in order to assist the operating system controlling the NoC. This work fits in the end-to-end monitoring class as employed in the regular (internet-like) network management. The work focuses on the use of such performance monitors to optimize communication resource usage. The monitored data uses a separate NoC, called the control NoC and does not reuse the NoC resources used by the application. It fails however to show what are the associated costs or implications of using monitoring, for example whether it is area efficient or not. Chapter 6 shows that the use of a separate NoC for monitoring, in general, regardless its use for performance or for transaction monitoring, is expensive in terms of area.

[1] proposes a dynamic routing scheme for reducing jitter in the latency of BE traffic, only in the combination with GT traffic, which can benefit from monitoring. In a manner similar to [69] the work requires NI statistics for packet injection control. In fact, the work assumes a monitoring system to be in place, solved by a third party. Our NoCMS can fulfill this role. The work fails to show

the associated costs, or to detail a monitoring scheme.

With the exception of the work targeted at performance measures and benchmarking [36] the rest of the related work is focused on statistics gathered at NIs, and not on the monitoring itself. All present approaches do not identify or tackle NoC monitoring problems, worry about a generic or scalable monitoring solution or even quantify area implications or other costs, e.g., required bandwidth or additional area. The focus is on the statistics gathered at NIs, or end-to-end monitoring, as opposed to monitoring at routers, or distribution monitoring. Our proposed NoCMS of Chapter 3 instantiated for run-time performance analysis reigns into these issues, showing a generic monitoring solution, supporting distribution monitoring and quantifying performance monitoring costs.

5.3 NoC Performance

As this chapter deals mainly with run-time performance monitoring it is important to first understand the main performance measures associated with the NoCs, where the measurements for obtaining or computing such measures can be done. This section details these NoC performance measures as well as their associated measurements.

As mentioned in the previous section, two works [36, 95] have already presented complementary parts of a performance framework targeted at NoC performance analysis. The used terminology distinguishes the notions of performance metrics, measures and measurements. Some of the most common performance metrics for NoCs, and also for networks in general, are throughput, utilization, latency or jitter. In general throughput is defined as the amount of units going though a resource during a defined period of time. Utilization is the percentage showing the actual usage of a resource from the total possible usage. Latency is the time taken by a unit to travel through one or more resources. Jitter is simply the deviation of a measure.

The performance measures are obtained by applying a performance metric to a performance target (or resources in the previous definitions). These performance targets can be physical NoC entities, e.g., routers, NIs, the entire NoC, buffers or ports, or logical NoC entities, e.g. connections, channels or messages. Some examples of performance measures are connection throughput, link utilization, router latency, network interface latency, or latency jitter.

Note that some measures can be computed based on other measures; e.g., an overall NoC utilization is computed based on router utilization which in turn can be computed based on link utilization. In general, they are composable; e.g., the end-to-end latency is composed of the NI latencies, router latencies, link latencies.

The measurements are the physical actions required to effectively gather data from the on-chip network, in order to be able to compute the performance measures. An example is the process of measuring the time difference between the time a message is offered to the sending NI and the time the same message is fully

delivered to the destination NI over a connection, in order to compute connection latency.

Performance measurements can be performed at different points in a NoC, and are in this work directly associated with the monitoring. The simplest differentiation between various measurements is whether these measurements are done at NIs or at routers. Our proposed NoCMS allows to perform the performance measurements at both the NI or at the routers. Similar to transaction monitoring of Chapter 4, we further assume in the rest of this work that performance measurements are always performed at routers; in the NI case there is no conceptual difference.

5.4 NoCMS for Run-time Performance Analysis

5.4.1 General Setup

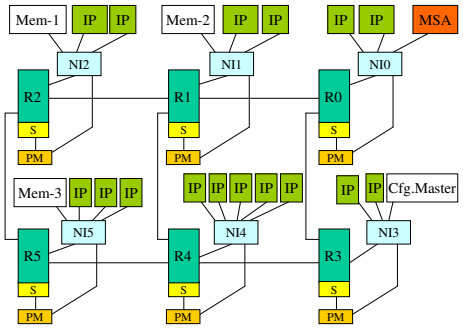


Figure 5.1: 2x3 MPEG Codec with NoCMS for run-time performance monitoring

As an example for our run-time performance monitoring work we have used the Æthereal NoC detailed in Section 2.3. The concepts presented in this chapter, however, similar to the ones concerning transaction monitoring of Chapter 4, are more general and can be reused for other NoCs, as they rely on the shared features of the NoCs as described in Chapter 2.

For the basic access to the flits passing to NoC routers we have used the generic NoCMS of Chapter 3 instantiated for performance monitoring. Our specialized performance monitor, PM in Figure 5.1, replaces the generic event generator. Figure 5.1 shows an example, in the form of a 2x3 MPEG Codec augmented with a NoCMS for run-time performance monitoring. The same NoC example has been used in Chapter 4 as an example for transaction monitoring. Note the similarities between figures 5.1 and 4.5, with the only difference being that the former uses a performance monitor (PM) and the latter a transaction monitor (TM), in the place of the generic event generator, showing the ease of instantiating a NoCMS

targeted at a specific monitoring task.

Similar to the transaction monitoring, the run-time performance monitoring process works as follows. The sniffer, S in Figure 5.1, obtains the raw flits (bit-level) from the NoC components and passes this information to the performance monitor. The performance monitor performs local processing on the sniffed data, in general simple filtering and counting. Multiple performance monitors are usually employed in a performance monitoring NoCMS. Individual performance monitors might measure different metrics at different points in time while multiple performance monitors might operate in the same or in different ways at the same point in time, e.g., counting different units, flits, headers or payloads. Our proposed performance monitor together with its capabilities is detailed in Section 5.4.2.

The performance monitors forward the results to the MNI. The MNI packetizes the result as payload and sends them over the network to the Monitoring Service Access point (MSA), over a previously established monitoring connection, just like any other data in the NoC. The monitoring connections can be BE or GT or a combination of them, as presented in Section 3.7.2.

Throughout this chapter we make the assumption that we use the same NoC for the resulting performance monitoring traffic as well as for the user traffic. This choice was made, as already mentioned in Chapter 4, because this solution allows a logical, dynamic partitioning of NoC resources; resources can be used for monitoring when needed, and freed when not. The alternative architectural options like a separate interconnect only for performance monitoring or just extra resources reserved only for the monitoring purpose may be used as well, as later described in Section 6.3 of this thesis.

5.4.2 Performance Monitor

Monitoring Measures

A very simple NoC wide map of link utilization is a very powerful tool in understanding the status of the NoC at certain moments in time, or understanding the QoS distribution. As in NoCs links are shared by multiple connections, link utilization is also a very good indicator for bottleneck identification. This information can be coupled as input to NoC control mechanisms, like the ones described in Section 5.6. Therefore, we focus our run-time performance monitoring of link utilization measures.

General NoC information has been presented in Chapter 2. The flit formats, and the information related to packetization in the particular case of the Æthereal NoC have been presented in Section 2.3. Summarizing, the IPs connected to the NoC communicate by means of transactions which in turn are composed of messages. Messages are packetized in packets and packets into flits. Flits are the minimum transfer or flow-control unit between hops: routers and NIs.

The means of identifying flits and headers, as well as the run-time header analysis by a monitor, the transaction monitor, has already been presented in the

previous chapter. The flit format was presented in Figure 2.8, in the general case when the flit contains a packet header as the first word.

Monitoring Capabilities

This work builds upon the work of Andre Boon as described in *"The Hardware Design of Monitoring Probes for the Æthereal NoC"*, Internship Report, Eindhoven University of Technology, Department of Electrical Engineering, 2006. In particular the author fully acknowledges the contribution of Andre Boon for the VHDL implementation of the performance monitors.

We propose a simple yet effective performance monitor able to count the number of units passing the set of links connected to the router it monitors, over a predefined period of time. Link utilization is computed in a generic way, per unit, by accumulating units that cross a link during a period of time, divided with the maximum number of such units that can potentially cross the link during the before mentioned period of time. In our view, these units can be

(1) flits,

(2) words, or

(3) payload words

Our proposed performance monitor in fact only counts the number of units passing the links, and does not effectively compute the utilization in terms of these units. The responsibility for this is taken by the MSA. From the predefined period of time, the MSA can infer the maximum number of units, e.g. as one flit corresponds to one TDMA slot.

$$link\ utilization = \frac{number\ of\ units}{maximum\ number\ of\ units} \tag{5.1}$$

By aggregating link utilization we can get router utilization, which in turn, by aggregation can reveal the overall NoC utilization. Nevertheless, the most important is the link utilization as this shows the distribution of utilization throughout the NoC enabling network control entities, e.g. QoS or network managers, to react.

To achieve this capability, the schematic of the performance monitor's internal architecture is depicted in Figure 5.2. The architecture is defined as a set of N *unit counters*, with N being the number of monitored router links, together with an *enable/configuration block*. In the mentioned figure, four unit counters are depicted, but N is a design-time constant, that means that we can generate a performance monitor with any N. All the router links are sniffed by the sniffer which provides this info to the corresponding unit counter. As opposed to the transaction monitor where a single link is selected for further processing, the performance monitor passes all flits to the corresponding unit counters.

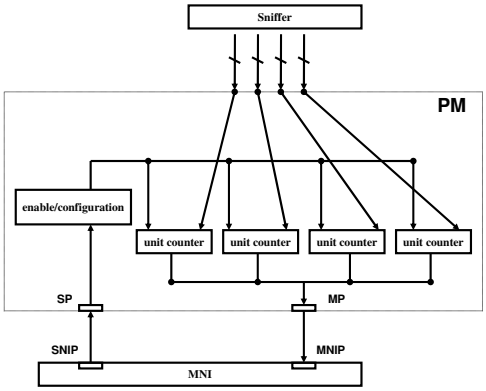


Figure 5.2: Performance Monitor

The information to be counted (the units) can be enabled and configured at run-time through the *enable/configuration block*. Two ports connect the performance monitor to the MNI, one slave port (SP in Figure 5.2), for programming the performance monitor, and one master port (MP in Figure 5.2), for sending the performance monitor data to the MSA.

(A) *Counting Flits.* When the units to be counted are set to flits then all the flits passing a link are counted, regardless of the fact whether the flits contain a header or only payload. Counting the flits gives an overall view of the NoC traffic in terms of flits. This measure is important as input for network management, or resource management at network level.

(B) *Counting Words.* If we count words both the header and the payload words are counted. While a flit may contain the packet header and useful payload, particular cases of flits may also contain only the header or only payload. First, if it exists, the header is counted. A header can be detected by looking at the second sideband bit of the first word of the flit. Looking at the both sideband bits of the second word Counting words gives an overall view in of the NoC traffic in terms of words; opposite to counting flits this measure reflects the under utilized flits. This measure is important as input for fine grain network management, or resource management at network level.

(C) *Counting Payloads.* When counting payloads, first, if it exists, the header is discarded (not counted). After this, the two sideband bits of the second word are checked. This reflects the actual payload of the flit and the result is added by the counter. Counting payloads gives an overall view of the NoC traffic in terms of the real load, the payloads, which does not account for the packetization. This measure is more suited as input for application management, or resource management at application level.

It makes no particular sense to make a particular performance monitor mode to count only headers, except perhaps to check the impact of the packetization on the NoC. This is an issue that has to be investigated at NoC design time, and has no extra value if it is done at run-time.

Monitor Configuration

Performance monitors are programmed using memory-mapped I/O, by means of write transactions. This is similar to the way NIs are programmed by means of write transactions, as explained in Chapter 3, and similar to the way transaction monitors are programmed, as explained in Chapter 4.

The content of performance monitor configuration data is kept in the configuration register. The memory map of the configuration register in our prototype implementation contains 20 bits: 10 bits for the counting interval timer, 8 bits for the start timer, and 2 bits for the unit configuration. Each of the unit counters outputs a 12 bit value.

Unit configuration. The unit configuration is valid for the configured performance monitor, for all the monitored links, and not for selected monitored links. This means that for a single performance monitor a single measure for all the monitored links can be selected. With two bits for unit configuration, the four possible settings for the transaction monitors are:

(00) not counting; this is the default setting for all performance monitors, all the other modes being only possible through configuration. This configuration is used also to stop the performance monitor. When the performance monitor receives this option it will immediately stop, regardless the values for the other configuration fields.

(01) counting flits, the performance monitor will count the flits

(10) counting words, the performance monitor will count the words

(11) counting payloads, the performance monitor will count the payloads

Start timer. For performance monitors, it is particularly important to be able to start their activity, the monitoring, at the exact same time, to have a synchronized start. Otherwise this data might be of less or even no value. Here is where the start timer gets into the picture, as a means of synchronizing the performance monitors. The eight bits of the start timer specify the time at which all probes in the system should start counting. If the unit configuration bits are set to anything different than (00), the performance monitor will start counting at the time indicated by the start timer. The unit of the start timer, these 8 bits, is 256 flit clocks.

We use a 16 bit time counter that is incremented each flit clock period, capable of counting 256 time periods of 256 flit clocks. The use of a 16 bit counter also

makes sure this counter does not wrap around too quickly for all the performance monitors to synchronize. The 8 bits provided in the configuration register are matched against its most significant 8 bits. The usage of the most significant 8 bits reduces the resolution of the time, but this granularity of 256 flit clock periods is considered sufficient for the synchronization of all performance monitors. In the particular Æthereal example, with a flit clock period of 6ns, this means that the performance monitors can synchronize at a granularity of 256*6ns = 1536ns.

Counting interval timer. The ten bits for the counting interval timer specify the time for which the performance monitor counts. After each of these periods, the probe puts the accumulated amount of flits, words or payloads to its output, in a 12 bit form. This time is given in the amount of flit clock periods; therefore, in our hardware implementation, at most 1024 flit clock periods can be counted before an output is generated. A larger counting interval timer means monitoring larger periods of time, and generates less often the output resulting in less traffic. A small counting interval timer generates results more often, and increases the traffic load.

The probe stops counting in the case of re-programming it, by using the (00) option in the unit configuration bits. The MSA, will give the command to stop the counting, after the moment it knows the counting must stop. The counting data received after the stop command is given is ignored. The monitors immediately stop counting, but not in synchronization like their start. Therefore counting data, e.g. from the most distant performance monitors may still arrive at MSA.

The value for the counting interval timer is used at the MSA to infer the maximum number of units, e.g. the maximum number of flits which could have passed one of the monitored link. Note that this number is the same for all the monitored links.

5.5 Analysis

5.5.1 Bandwidth requirements

In general the bandwidth required for a performance monitor to operate is given by:

$$required\_bw_{PM}(N, interval\_counter\_time) = \frac{N * nr\_bits}{interval\_counter\_time * flit\_period} \tag{5.2}$$

where N is the number of links desired to be monitored, nr_bits is the number of monitoring counter output bits per link, $interval\_counter\_time$ is the monitoring period, $flit\_period$ is the duration of the flit clock. For any given network nr_bits, $interval\_counter\_time$, and $flit\_period$ are fixed; N is a design time parameter, and $interval\_counter\_time$ can be configured at run-time;

For the particular Æthereal example with four monitored links, and an *interval_counter_time* of 1024, this becomes:

$$required\_bw_{PM}(4, 1024) = \frac{4 * 12}{1024 * 6} = 7.8\frac{Mb}{s} \tag{5.3}$$

Compared to the transaction monitoring, we can see that the bandwidth requirements of performance monitoring are small.

5.5.2 Area results

Single Performance Monitor

In order to prove that on chip run-time performance monitoring is feasible in resource-constrained NoC designs, we have investigated the area of our hardware performance analysis monitors. For the experimental validation of our NoCMS for performance analysis, we have built a flit accurate SystemC model, and a cycle accurate synthesizable VHDL model of the performance analysis monitors. We have used these models in conjunction with the Æthereal NoC.

We implemented the hardware performance analysis monitors. The area overview of several performance monitors, corresponding to a number of supported links, or N, between 2 and 8, realized in a 0.13μm CMOS technology and the other three parameters fixed to the already mentioned Æthereal NoC values, is presented in Table 5.1. The performance monitor area presented includes the enable/configuration unit allowing to (re)configure the performance monitors at run-time. In general the area of the designed performance monitor for an arity N router can be estimated in mm^2 in a 0.13μm CMOS technology as:

$$A_{PM}(N) = N * 0.0023 + 0.007 \tag{5.4}$$

where N is the number of links desired to be monitored.

Note that in this work area is associated with the arity of the router, because we use the assumption that in order to achieve a full link coverage all routers in the performance monitoring NoCMS must be probed with a performance monitor. This might not always be needed, depending on the monitor, e.g. when performance monitors supporting $2 * arity$ links are placed on the routers in a vertex cover set of the NoC, which also covers all links of the NoC.

In order to see how good these results are, a comparison with the area numbers of the Æthereal router is made, as well as with the transaction monitor of Chapter 4. The results of this comparison are presented in Table 5.2.

The area results for the performance monitor attached to an arity 6 GT/BE router is 0.018mm^2 in an 0.13μm CMOS technology. The area of the corresponding Æthereal arity 6, GT/BE router, presented in [41], is 0.13mm^2 in the same 0.13μm CMOS technology. Compared to this router the performance monitor is not bigger than 14% of its target router, showing an acceptable area cost. Even when compared with the smallest arity 6 Æthereal router, the GT only router

Table 5.1: Performance monitor area results

Nr. of links	***Area (mm^2)***
2	0.009
3	0.011
4	0.014
5	0.016
6	0.018
7	0.021
8	0.023

with an area of 0.033mm^2 [41] in a 0.13μm CMOS technology, the performance monitor is roughly about 50% of its size. The area of the routers employed in a NoC design (and implicitly the area of a single router) does not account for much of the total NoC area, the area of NIs dwarfing the area of routers even for very simple NoC instances.

Table 5.2: Performance Monitor area comparison

Modules	*PM*	*TM*	*R (GT/BE)*	*R (GT)*
Area (mm^2)	0.018	0.020	0.13	0.033
		0.024		
		0.026		

The area of the transaction monitor is roughly constant and does not depend on the number of links being tracked for monitoring, as a single link is selected inside the monitor for identifying transactions. The area results of Chapter 4 show that a transaction monitor offering raw data monitoring capability requires 15% more area compared to the earlier mentioned arity 6 GT/BE router, or 0.020mm^2 in an 0.13μm CMOS technology. Adding connection-based capabilities requires around 18%, or 0.024mm^2. Full fledged on-chip transaction reconstruction is feasible at the cost of 20% of the router area, or 0.026mm^2. The area required for the performance monitor is even less than the area required for the simplest transaction monitor, i.e. the one offering only raw data monitoring capability.

5.6 NoCMS-assisted QoS

This section presents the applications of the NoCMS for performance analysis. The author was involved in this work, his contribution being the monitoring system and how the run-time monitored data can be interpreted in useful information about the network status. The author's claim does not intend to cover the QoS mechanisms, or the congestion-control mechanisms in the form of the

controllers, for which the author fully acknowledges the first authors of [94, 71]. Both techniques are similar, but differ in the layer where the monitoring feedback is employed and the context or the use case. Parts of [94] and to a lesser extent of [71] works are presented in this work, as part of this section, only for the purpose of consistency and self containment, with as the main goal to demonstrate the genericity and versatility of the NoCMS.

5.6.1 Congestion-controlled BE

General Setup

Congestion has negative effects on the NoC run-time performance. [94] proposes a novel congestion control strategy for NoCs. For this purpose, a new communication service, called congestion controlled best-effort (CCBE) service is introduced. This service extends the original Æthereal BE service; this means that three services coexist now in the Æthereal NoC. The highest quality offered service is GT, followed by the CCBE, and the lowest quality service is the regular BE.

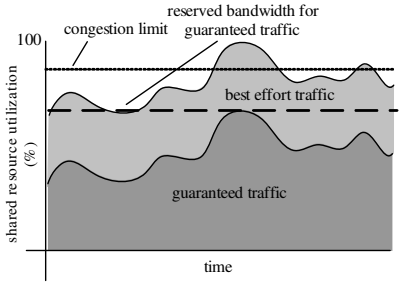


Figure 5.3: Normal BE traffic combined with GT traffic in the NoC

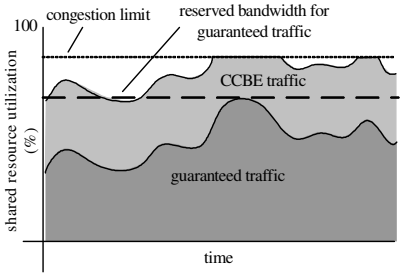


Figure 5.4: CCBE traffic combined with GT traffic in the NoC

The load offered to a CCBE connection is controlled based on multiple router congestion measurements, in the form of router link utilization, inside the mon-

itored NoC; this implies that the NoC-connected IP itself has its own means of controlling its generated traffic; IP load can be controlled by using for instance voltage scaling, degrading audio or video quality, or by partially or completely disabling jobs. All NoC routers are instrumented with the performance monitors previously described in this chapter. Link utilization is chosen to be monitored as a congestion measure, and transported to a centralized MSA which connects to a Model Predictive Controller (MPC). GT connections are used for this, to assure continuous progress of link utilization data in a congested NoC.

The mentioned work also presents a simple but effective model for router link utilization for the model-based predictions, with experimental results showing that the presented strategy is effective and has reaction speeds of several microseconds which is considered acceptable for realtime embedded systems.

The general CCBE principle can be observed in the Figures 5.3 and 5.4. The first figure shows a shared link transporting both constant bit rate regular BE traffic and a variable bit rate GT traffic with the reserved bandwidth depicted with a dashed line. It can be immediately observed that the BE traffic gracefully follows the variations of the GS traffic. The mixture of BE and GT traffic improves resource utilization but at certain moments in time the shared link is congested. The second figure shows the mentioned CCBE service which still follows the GT traffic but will not pass the value set up as the congestion limit.

Prediction Method

CCBE relies on model predictive control (MPC) as the means for controlling the offered IP loads. The general process works as shown in Figure 5.5. The MPC inputs are the measured link utilizations coming from the monitors, and the MPC outputs are the loads allowed by CCBE connections respectively. The desired link utilization is also initially specified as the target where the link utilization should be maintained. BE latency measurement for the Æthereal NoC have indicated 80% of link utilization as the congestion limit. In general, the MPC is not bound to this value. A centralized MPC strategy is employed matching the centralized monitoring service employed. A set of IPs to be controlled are receiving the MPC outputs, and increase or decrease their offered load.

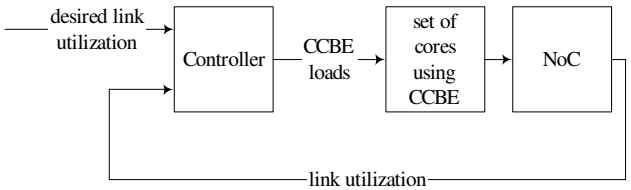


Figure 5.5: Complete CCBE control process

Model predictive control (MPC) is a technique that combines model-based

predictions with actual system measurements, provided by performance monitoring in our case. MPC is an optimal control method. MPC controllers are designed by optimizing a cost function and are known for their ability to deal with varying latencies which is critical for our control problem. MPC distinguishes itself from other optimal control methods by solving the optimization problem at runtime. These optimization problems are typically solved by quadratic programming (QP) [77]. The chosen MPC here is the MPC from the Matlab MPC toolbox [58] which uses Dantzig Wolfes method [26] for QP. MPC allows constraints to be specified for controller inputs and outputs. Minimum and maximum values can be specified, as well as rise and fall speeds (i.e. how many MBytes/s an IP can raise or drop its load per control interval). MPC takes these constraints into account when making control decisions to ensure that the system will not oscillate.

The two most important parameters that affect performance and cost of the MPC are the control interval and the prediction horizon. Each control interval δ the MPC decides on new values for the controlled variables (in our case CCBE allowed loads). An important parameter in deciding an appropriate value for δ for NoCs that use time division multiplexing (TDMA) is the size of the slot wheel, and δ should at least be a multiple of the slot wheel (e.g. five times the slot wheel). During δ, future states are explored over a prediction horizon p. So, each δ, an optimization problem has to be solved by means of QP while considering the effect of decisions over p control steps. In practice, choosing a p value between 1 and 5 gives reasonable performance results. Experiments have shown that a δ between 200 and 1000ns will maintain the reaction speed below 25μs. Measuring link utilization over a small sample period of 1000ns only requires 0.3 MBytes/s which is only 0.015% of the available link bandwidth of 2 GBytes/s. Furthermore, it has been shown that it is possible to run MPC for a realistic control problem on a modest FPGA chip [47]. MPC is therefore well suited for controlling the CCBE traffic.

NoC model

MPC uses a model of the controlled system, in our case the NoC, to iteratively compute future behavior which must be as simple as possible to minimize the amount of computation for the online QP algorithm. In our case link utilization is modeled by taking the sum of the loads of the CCBE connections that share the link. A communication overhead factor k is included for each connection to model the difference between IP load and actual load in the network. For instance, for the Æthereal NoC, BE data is transported through the network as packets. Each packet has a packet header of one word. If a packet size of 36 words is used, k = 1/36. Unit delays following the communication overhead factor model the forward propagation delay from CCBE IP to shared link. The delay from a shared link to the MPC is modeled with a unit delay. By dividing the link load with link bandwidth (2 GBytes/s for Æthereal) we obtain link utilization. Unit delays are used rather than an estimate of propagation delays to keep the model as simple

as possible. Estimates improve controller behavior at the cost of a more complex model. In the model, $\overline{u} = [u1, u2, ..., um]$ is the input vector which represents the loads of the CCBE IPs. $\overline{y} = [y1, y2, ..., yp]$ is the output vector which represents utilization of the links. $\overline{x} = [x1, x2, ..., xq]$ is the state vector where q equals the number of delays in the model.

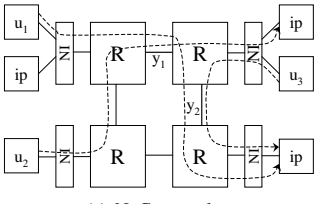


Figure 5.6: NoC Example

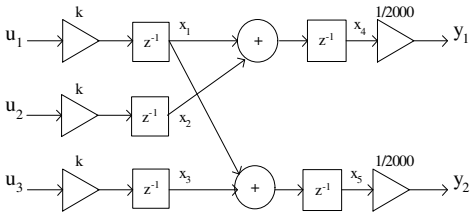


Figure 5.7: Corresponding Flow Graph

An example NoC is shown in Figure 5.6, together with its corresponding model for the two links (y1 and y2) is presented in Figure 5.7. Connections are represented by dotted lines. Link y1 shares u1 and u2, link y2 shares u1 and u3. The state space description of the small example is as follows, where n is the discrete time variable, C is the output matrix, A the state-transition matrix and B the input matrix: $\overline{y}(n) = C\overline{x}(n)$, $\overline{x}(n+1) = A\overline{x}(n) + B\overline{u}(n)$ The A, B, and C matrixes corresponding to the example in Figure 5.7 are shown in Figure 5.8.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & k \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 1/2000 & 0 \\ 0 & 0 & 0 & 0 & 1/2000 \end{bmatrix}$$

Figure 5.8: Corresponding A,B and C

Example

In this subsection the already known MPEG 2x3 example, employed in Chapters 3 and 4, is used as a test bench for the CCBE service. The example is shown in Figure 5.9. This MPEG NoC example has a slot table size of 30 slots (180 ns). The increase in link utilization due to the presence of CCBE is measured together with the reaction speeds of the various congested links in the system.

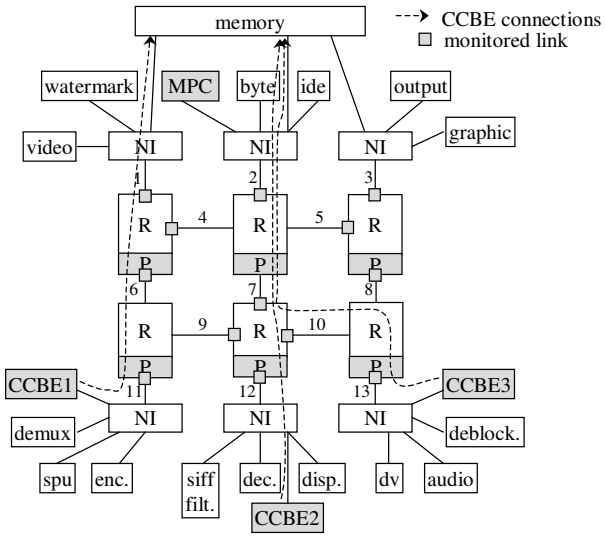


Figure 5.9: CCBE MPEG Example

Overall link utilization, i.e. the sum of all link utilizations averaged over the number of links, is equal to 24% for the original configuration. In order to improve utilization 3 CCBE connections are introduced in the original design, which for example represent connections of another application using the same platform.

Table 5.3: Reaction speeds for various control intervals

	reaction speed μs	
Ctrl. interval (ns)	*MPEG L6*	*MPEG L7*
200	5	4
400	10	8
600	12	10
800	20	12
1000	25	16

These connections follow the paths as displayed in Figure 5.9 with dashed lines. Monitors are attached to each router, each monitoring all router links.

The connection for IP CCBE1 traverses three links, none of which is used by other CCBE connections. CCBE2 and CCBE3 share links L7 and L2. The maximum loads of IP CCBE1 to CCBE3 are chosen in such a way that the NoC reaches a congested state. They are: 1000 MBytes/s, 400 MBytes/s and 400 MBytes/s respectively. Attaching these CCBE IPs to the NoC results in an increase of overall link utilization of 16%. Utilization of link L1-3, L6-7, L10 and L11-13 are increased significantly by these new loads. However, link L6 and L7 are congested (utilization of L6 equals 100% and that of L7 equals 95%).

We have observed that choosing a control interval of five times the slot wheel size typically results in stable system behavior. If this still results in unstable behavior or if smaller control intervals are required, another solution is to constrain rise and fall speeds of CCBE IPs.

We perform a pulse response experiment by applying the aforementioned loads as pulses to the NoC. To obtain a stable system with the slot wheel of 180 ns but with the control intervals from the previous experiment, we constrain rise and fall speed of the loads of the CCBE IPs to -10 and 10 MBytes/s per control interval. The measured reaction speeds for the two congested links are shown in Table 5.3. The reaction speeds for link L7 are better than those for link L6. Link L7 is shared by two CCBE connections. The load of both CCBE IPs are allowed to rise and fall at the same speed as the load of CCBE1 that uses link L6. With the MPEG case we have shown that CCBE is feasible for an example with realistic traffic and for multiple shared links and controlled loads in the system.

5.6.2 Run-time application QoS

To improve the run-time resource utilization or improve and maintain Quality of Service (QoS), run-time monitoring can be employed. Many NoCs practice reservation policies, where resources are provisioned for worst case executions. Monitoring helps in observing what resources are used, giving a run-time overview of resources, in an aggregated form or per traffic class, potentially allowing other applications to use the NoC slack. For example, QoS relies on estimating resource

usages of applications on hardware platform. However, due to the dynamic nature of future applications , e.g. object based MPEG, a great advantage is the coupling of the estimations of resource usage with the concrete observations of resource usage, results obtained by monitoring. This may result in increasing the quality level or a more efficient resource usage. NoC monitoring has the advantage that is being decoupled from the application, while using other application will still benefit from the same monitoring system.

A two layer QoS manager is employed in the [71]. It combines a single global QoS manager with multiple application local Q0S managers. The general setup of this approach is presented in Figure 5.10. The depicted Global QoS manager assigns appropriate resources and quality levels to each application. this assumes that the applications are aware of multiple quality levels. The main part of it is the resource estimator which calculates for each application the amount of required resources for processing the set of new data. The targeted application is a MPEG-4 video decoding, where we decided to lock it to the size of the GOV. The resource request is evaluated with the available resources and the highest quality is set by the Global QoS manager. These resources are reserved for the application until the end of the GOV or any exceptional situation occurs. The Global QoS is aware of all running applications and estimates resources for them, controlling their long-term settings for each application.

The Local QoS manager is responsible for monitoring the prediction model and real resource consumption via monitoring. The LocalQoS controls an individual application and translates the settings into a scalable setting of the tasks. The monitoring of execution is done at fine granularity, in our case at the VOP level. The Local QoS sets the parameters for scalable communication connections and scalable tasks at run-time, based on the resource availability.

The Local QoS connects to run-time monitors of resources; both CPU monitors and network monitors. to have up to date information about the resource utilization status. In fact it is a normal extension form previous work using only computation monitors to include network monitors. An instance of our performance NoCMS is employed which features a fully probed NoC, a 2x4 mesh with 8 NIs and 8 cores, with performance monitors. A centralized approach is used for collection monitoring data to the MSA local to the LocalQoS.

The Local QOS may decide to modifies the functional task execution of its application to a higher level, if processing at a higher quality level is possible; this enables a higher quality level for a short time, for a fragment of the GOV. The Local QoS temporary sets parameters for scalable tasks to a higher quality level for decoding of the next VOP within the actual GOV. This quality level is higher than the quality level that was assigned by the Global QoS manager. Statistical experiments showed that a worst-case approach is not necessary in 80% of the execution time. This effectively means that in general the Global QoS manager is overestimating the required resources. These reserved resources are actually used only 20% of the time and during the remaining time can be used by other applications. If we would have only a reservation-based solution, the system

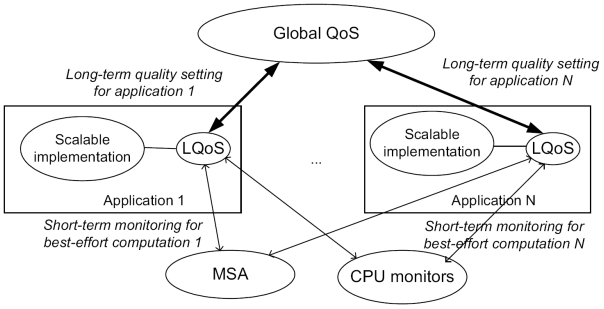


Figure 5.10: Two layer QoS Manager

would have to wait until the next suitable time for changing the quality level and the corresponding reconfiguration (end of GOV). In the new solution, the Local QoS calculates for each VOP the resource requirements of the succeeding VOP and compares it with the run-time information from the network monitors and CPUs monitors. If the centralized MSA reports sufficient available resources for the communication, the Local QoS allows the computation at higher quality level.

More details about the Global QoS and Local QoS and their operation can be found in the original work of the authors [71] and in [72]. The instantiation and inclusion of the performance NoCMS into this setup is the addition made to their work.

5.7 Conclusions

This chapter has shown the versatility of our proposed NoCMS with an instance for performance monitoring. This instance is based on a performance monitor able to count flits, words, or headers. The area of the performance monitor is small compared with the area of a transaction monitor. The bandwidth requirements of the performance monitor are also small compared e.g. with the bandwidth requirements of a transaction monitor employed e.g. in raw mode. The proposed instance has been successfully employed in the provision of a congestion-controlled best effort service by [94] and in a application level QoS manager by [71]. Both approaches have tried to leverage the advantages of monitoring resource utilization of best-effort systems, but in different ways.

Chapter 6

A Monitoring-Aware NoC Design Flow

With the debug instance of the generic NoC monitoring service for transaction monitoring presented in Chapter 4 and the instance for performance monitoring presented in Chapter 5, it is time now to investigate whether there are implications for the NoC design flow, specifically the implications driven by monitoring. As one of the main thesis contributions, this chapter binds the generic NoC monitoring service in general and the NoCMS instances in particular to the NoC design flows, completing the overall monitoring picture. In particular it looks into the means of deploying only the required number of monitors for a given monitoring task, into interconnecting the spatially distributed monitoring probes and into the automation of this task.

After exposing a brief motivation in Section 6.1, this chapter continues with the related work. Section 6.3 presents several scalable alternatives for interconnecting the NoCMS monitors. These alternatives are independent of any specific NoC. All options are based on the reuse of NoC components and (parts of) the NoC design flow. As all are based on a NoC interconnect, they are all scalable solutions. For each of the proposed solutions, we explain the main concepts and the architectural details. We evaluate all the proposed solutions with respect to four aspects: (1) impact on the overall NoC design flow, (2) non-intrusiveness, (3) area cost, and (4) reuse potential of monitoring resources. All our options are exemplified with the Æthereal NoC and design flow.

In the second part of this chapter we propose a concrete monitoring-aware NoC design flow for one of the alternatives, namely the one in which the NoC resources are all shared between the application and the monitoring service. The proposed flow is able to take into account the monitoring requirements at all steps in the NoC design flow. This is done in a generic way, for any NoC, for any kind of monitoring. As an example, we illustrate the proposed monitoring-aware NoC

design flow with a debug driven monitoring case study as presented in Chapter 4 in the now familiar context of Philips' Æthereal NoC, with the corresponding ASIC NoC design flow UMARS.

Simple, area-efficient transaction monitors as detailed and evaluated in Chapter 4, attached to selectively chosen NoC routers, are used to enable debugging of the NoC-based SoC at transaction level. This is one of the most difficult cases, where the monitoring requirements are only known after the path selection step in the NoC design flow. In the context of application specific designs using an application-aware monitor placement method as sketched in Section 6.4, the proposed monitoring-aware NoC design flow of Section 6.5 is able to automatically insert transaction monitors, by determining the number and placement of these transaction monitors and accounting for their communication requirements. The smallest NoC which satisfies the application requirements, as well as the monitoring requirements is generated as a result.

The area implications are quantified and compared to original NoCs without monitoring, and with the area target and initial estimations of Chapter 3. In order to alleviate the NoCMS area-cost woes, the efficiency of the proposed monitoring-aware NoC design flow is shown on several realistic and synthetic examples in Section 6.6. Section 6.7 concludes this chapter with a rehearsal of its main ideas.

The work presented in the first part of this chapter regarding the interconnection of spatially distributed monitoring probes has been published as *"NoC Monitoring: Impact on the Design Flow"*; Calin Ciordas, Kees Goossens, Andrei Radulescu, Twan Basten; In Proceedings of the International Symposium on Circuits and Systems (ISCAS), May 2006. [22]

The remaining of the work in this chapter targeting the seamless integration of monitoring related activities in the NoC design flow has been published as *"A Monitoring-Aware Network on Chip Design Flow"*; Calin Ciordas, Andreas Hansson, Kees Goossens,Twan Basten; In Proceedings of the 9th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools (DSD), August 2006. [23]

An extended version of the latter has been published as *"A Monitoring-Aware Network on Chip Design Flow"*; Calin Ciordas, Andreas Hansson, Kees Goossens, Twan Basten; in Elsevier's Journal of Systems Architecture. [24]

6.1 Motivation

With the need for multiple monitors introduced in Chapter 3, the problem of their interconnection has become clear. The significant challenges of interconnecting the multiple, spatially distributed monitors have also been presented in the same chapter. This has been shown together with the preference for a NoC as a natural fit for monitoring interconnect. In all the previous chapters, it has been tacitly assumed that the same NoC is used for both application data and monitoring data, as the focus was on the general aspects of the NoC monitoring service of Chapter 3

or on the more specialized instances of it: the performance analysis of Chapter 5 and the transaction monitoring of Chapter 4. This is however not necessarily the case; multiple possibilities exist. In the related work of Section 1.6, monitors and the traffic they generate were added into the SoC by using a separate monitoring NoC. A separate NoC for monitoring traffic may not always be desirable for the associated high area costs, and a more efficient solution is to use the same NoC for both monitor data and user data, as already suggested in Chapters 4 and 5. We argue that the options like sharing or not sharing a single (NoC) interconnect for functional and monitoring traffic are key to monitoring system design. These options have to be evaluated to determine which one results in area efficient solutions, and this is where this chapter lays its emphasis in the first sections.

Networks on chip require design flows to aid in design-time decisions, as detailed and exemplified in Chapter 2. The need for NoC monitoring must be accounted for in the design phase, regardless the choice for the mentioned options for the monitoring interconnect; the design of the single shared NoC or of the separated NoCs must be done with the NoC design flow. This is done in any of the debugging or performance monitoring contexts. In this case, the monitoring problem must be solved within or at least coupled with the NoC design process having an impact on the NoC design flow. When monitoring traffic uses an interconnect of its own, it can be dimensioned after the user data NoC is designed. This merely adds an extra step in the design flow. However, when monitor and user data must share the same NoC, the overall design flow must be revised.

As the NoC design flow revision comes into sight, we have to look at the number and placement of monitors and their associated monitoring communication requirements. While the mentioned NoC design flows require as input the communication requirements, some monitoring probe communication requirements are not known beforehand, but only after the NoC to be probed has been designed, or at least some steps in the NoC design flow have been performed. For example, some requirements may be known only after topology generation, such as the number of routers employed in the NoC, which is relevant if all routers or a coverage of routers need to be probed e.g. with router monitors showing link utilization or with transaction monitors. In this case, the number of routers determines the number of probes and their placement, while their communication requirements are fixed, depending only on the number of links being traced. Other monitoring communication requirements may be known after the path selection step in the design flow, e.g. router monitors able to trace a connection, e.g. the functional traffic for debug reasons (or for connection utilization). In this case, assuming a desired full coverage of the connections, the number of probes and their placement is given by the routers in a cover of all the connections. Their communication requirements depend on the number of connections passing the probed router and their sizes in terms of bandwidth.

We observe the existence of two interdependent problems: the one of functional dimensioning of the NoC and mapping of cores while accounting for their communication requirements (as already solved by the NoC design flow), and the

other of monitor placement and monitoring bandwidth specification (which has to be solved in the frame of monitoring). If these two problems are solved sequentially, the monitoring communication requirements can be pre-computed. Note that all the previous chapters tacitly assumes that the monitoring traffic fits on top of the user traffic on the existing shared NoC. In the case of a shared NoC there is also a reasonable chance that the newly created monitoring traffic may not fit directly on top of the existing user traffic on the generated application NoC. In this case, a new NoC must be generated, e.g., by increasing the topology and repeating the process. By increasing the topology, the number of NoC routers may increase. In turn, the mapping, path selection and allocation of resources may change and the number of required monitoring probes may increase as well and their communication requirements may change. Thus, the monitoring problem must be solved within or at least tightly coupled with the NoC design process. The task of placing the specific monitors has to be automated and integrated in the NoC design flow.

6.2 Related Work

As already mentioned throughout this work the problem of how many NoCMS monitors are needed, their automatic placement in the NoC-based SoC by means of a monitoring-aware NoC design flow and the associated area cost implications have not been previously investigated. In this chapter we evaluate a monitoring aware design flow that fully integrates the design of the NoC and its monitoring service, solving all the above mentioned issues, in the context of the ASIC NoC design case.

In a few rare combinations of NoCs and monitoring [69, 63, 1], the use of NoC monitors is proposed. These monitors are employed in real ICs or in FPGA prototypes mainly with the purpose of performance improvement at run-time or at design time. For the run-time, this is done via an operating system controlling the NoC, or a dynamic routing scheme for reducing jitter in the latency of BE traffic. For the design time, it is simply another iteration in the design process.

All this previous work ignores the effects of monitoring on the NoC design flow, the automation aspects of monitoring, e.g. whether monitoring support can be added in an automatic way or not, or what is the required number of monitors to be employed. For example, in the ASIC NoC case and in the context of transaction monitoring, the final target is the automatic insertion of the (minimum number of required) transaction monitors in an automated way based on the known application requirements, with all the effects and costs of this compared to the original NoC. Furthermore, the existing work on the NoC design flow, as presented in Section 2.1, simply ignores all the impact of monitoring on the NoC design flows.

All previous works assumes that, the placement of the monitors is known and the monitoring generated traffic or communication requirements are known

in advance. They also assume that the monitoring/debugging this traffic fits directly on top of the dedicated interconnect for this purpose, dedicated wires, test infrastructure or even a NoC. For NoC monitoring, in general, these assumptions are not valid. Furthermore, one shared NoC may be a more area efficient solution.

In this chapter we propose a monitoring aware design flow that fully integrates the design of the NoC and its monitoring service, solving all the above mentioned issues.

6.3 Monitoring Interconnect Options

In this section, we consider and explore several monitoring options for the interconnection of spatially (NoC-wide) distant monitors which are to be employed for monitoring purposes. For this, we took into account two criteria:

(first) separate or common physical interconnect (NoC) for the user application and for the NoCMS

(second) shared or not shared resources (e.g. router links) for the user application and for the NoCMS

This separation has resulted in three considered options:

(A) Separate Physical Interconnect for the original NoC application and the NoCMS

(B) Common Physical Interconnect but Separate Physical NoC Resources

(C) Common Physical Interconnect and Shared Physical NoC Resources

Note that we cannot have separate NoCs for the user application and for the NoCMS with shared resources.

All these three interconnection options are generic. They are also supported in our reference Æthereal NoC design flow. As a reference example we have chosen an MPEG codec with a 2x3 mesh NoC interconnect as originally presented in [40]. The cost of the original NoC interconnect is 2.35mm^2. In the following subsections, for each option, we explain the concepts, the impact on the NoC design flow, the non-intrusiveness aspect, the reuse potential of monitoring resources for application traffic, and the interconnect area cost. Figure 6.1(a) serves as illustration to differentiate the different options. Each of the options is compared to:

(1) the original NoC, called user NoC in the remainder, as shown in Figure 6.1(a) and

(2) the typical NoC design flow, split in four steps as shown in Figure 6.2(a) and presented in Chapter 2: topology selection, mapping, path selection and slot allocation.

Note that the area numbers which are further presented in this section only account for the interconnect and not for the associated monitors because the area of the probes is the same in all cases. Also, in all forthcoming examples we have used GT connections for monitoring purposes.

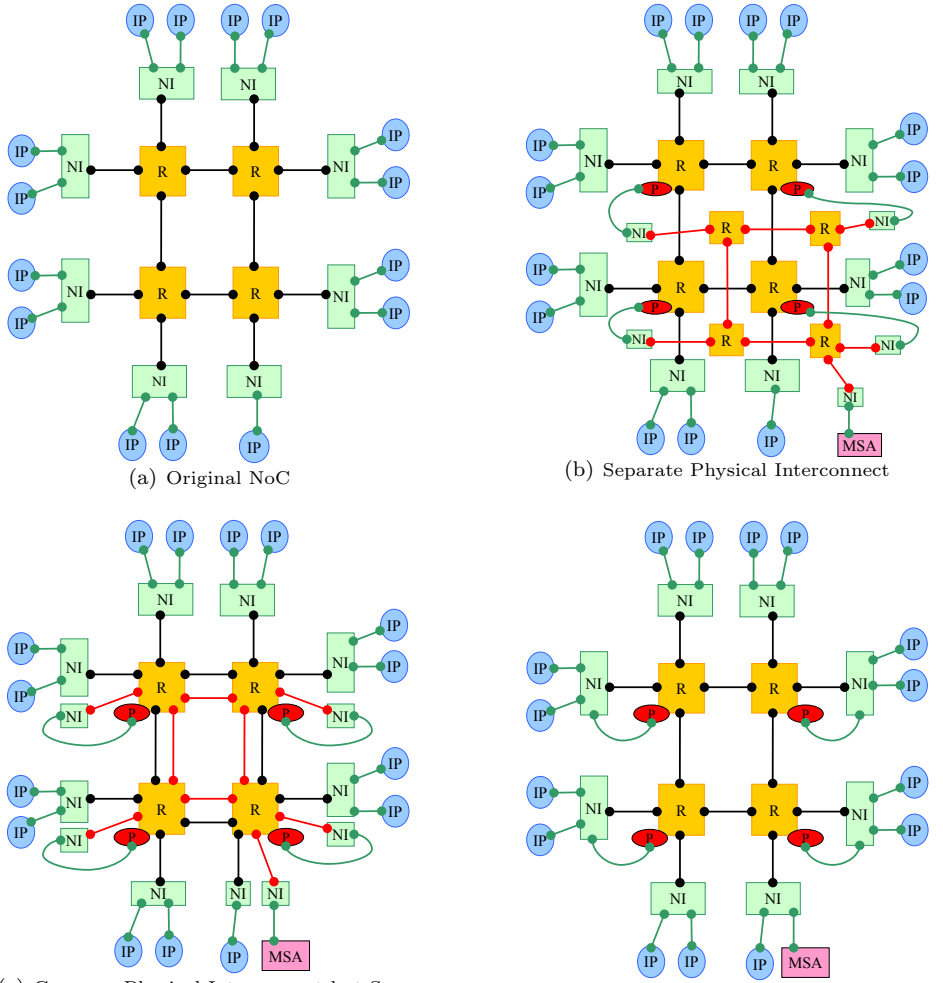
6.3.1 Separate Physical Interconnect

In this case, a separate physical interconnect is chosen for monitoring. Although any interconnect may be used, we have chosen to use a NoC, the monitoring NoC, because it is scalable. Figure 6.1(b) shows the resulting system. The monitoring NoC is used for transporting the monitoring data from probes to the MSA and for monitoring configuration traffic from MSA to the probes. The monitoring NoC can be similar in topology with the user NoC interconnect. For simplicity, we only show a fully probed NoC in Figure 6.1(b), with probes attached to all routers. A more advanced, selective NoC probe placement at routers is possible, e.g. ensuring a coverage of all physical NoC links. In the remainder, only a fully probed NoC is assumed as well. For each of the probed routers, we add a new router and an NI. The NI is used by the probe to connect to the monitoring NoC. Please note that probes can be attached also to NIs or IPs in the system, in which case these will connect to the monitoring router corresponding to the user router these NIs or IPs connect to in the user NoC. Each of the probes and the MSA connect to the monitoring NoC through a separate NI. Optionally, taking into account the monitoring requirements driven e.g. by debugging, some of the monitoring NoC links (in between routers) may be removed, as long as each probe can still connect to the MSA.

Design Flow Impact: During the NoC design process, the NoC design flow is applied twice: (1) for the user NoC, taking into account the user communication requirements as shown in Figure 6.2(a), (2) for the monitoring NoC, taking into account the monitoring communication requirements as shown in Figure 6.2(b). Dimensioning of the monitoring communication requirements and of the number of monitoring IPs (e.g. router probes) required, which are dependent on the user NoC topology, mapping, and path selection, is simple as all these aspects for the user NoC are not influenced in any way by the monitoring system and are already done beforehand. While applying the NoC design flow for the monitoring NoC, topology is already given by the original NoC, and mapping is given by the probe placement in the original NoC, as previously explained. Therefore only the path selection and slot allocation have to be done for the monitoring NoC.

Non-intrusiveness: This solution is non-intrusive because only the monitoring NoC is used for transporting the monitoring data. No interference between monitoring NoC and user NoC is possible, because they are physically disjoint.

Area cost: A total NoC area cost of 3.82mm^2 (2.35mm^2 original + 1.47mm^2 extra) was determined based on the addition of 7 NIs for the 6 probes and one MSA, and of 6 routers.



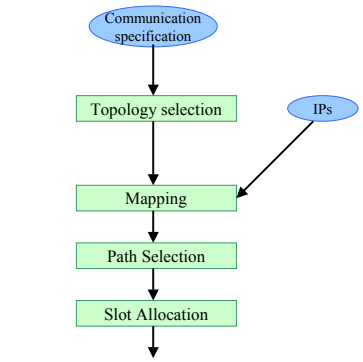
(a) Original NoC

(b) Separate Physical Interconnect

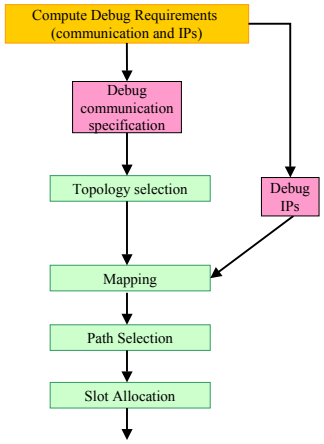
(c) Common Physical Interconnect but Separate Physical Resources

(d) Common Physical Interconnect and Shared Physical Resources

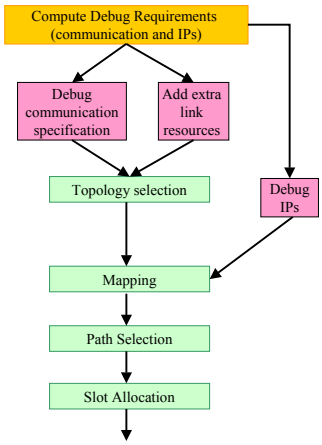
Figure 6.1: Monitoring Transport Options



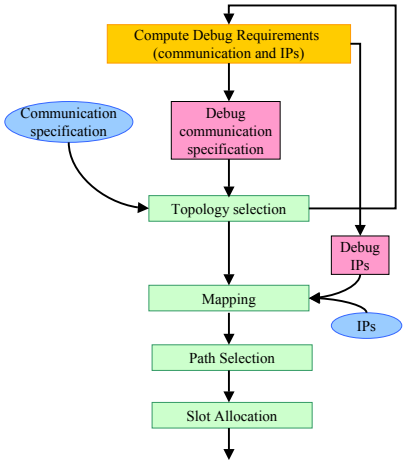
(a) Original NoC



(b) Separate Physical Interconnect



(c) Common Physical Interconnect but Separate Physical Resources



(d) Common Physical Interconnect and Shared Physical Resources

Figure 6.2: Design Flow Modifications

Reuse: No reuse potential due to complete separation of monitoring and user NoCs.

6.3.2 Common Physical Interconnect but Separate NoC Resource

An alternative monitoring option is to have within the user NoC a separate monitoring subnetwork. No new routers are added, but following the NoC topology, separate links and their corresponding router ports are added to the existing NoC. Each probe and the MSA gets its own NI to connect to the NoC. This is visually depicted in Figure 6.1(c).

Design Flow Impact: During the NoC design process the NoC design flow is applied twice: Considering the user requirements, the user NoC is obtained, by going through the NoC design flow. In this way, the topology, mapping, path selection and slot allocation are computed for the user NoC as shown in the reference design flow of Figure 6.2(a). At the second run of the design flow, as shown in Figure 6.2(c), the monitoring communication requirements and the required monitoring IPs, e.g. one probe for every router in the original NoC, are derived. This is done based on the user NoC topology. The user NoC is then extended with the monitoring resources, router links. In the path selection and slot allocation steps, the newly added router links are only scheduled for the monitoring traffic. Optionally, taking into account the monitoring requirements, driven e.g. by debugging, some of the monitoring router ports and links may be removed from the monitoring subnetwork, as long as each probe can still connect to the MSA. Dimensioning of the monitoring communication requirements which are dependent on the user NoC topology, mapping, or path selection is simple as the user NoC path selection and scheduling is not influenced in any way and done beforehand.

Non-intrusiveness: This solution is non-intrusive as only the monitoring subnetwork consisting of dedicated links is used for transporting the monitoring data. Although the routers are shared, the set of user links and the set of monitoring links are disjoint. No interference between the monitoring subnetwork and the user subnetwork is therefore possible. Existing user NoC scheduling in the original NoC (Figure 6.1(a)) is kept also in the new NoC (Figure 6.1(c)).

Area cost: This solution has a high NoC area cost: 3.88mm^2 (2.35mm^2 original NoC + 1.53mm^2 extra). This was due to increasing the arity of all six routers, e.g., from 3 to 6, and the addition of 7 separate NIs, from which 6 for the probes and one for the MSA.

Reuse: One advantage is that after the completion of the monitoring task, e.g. after debugging, some monitoring communication resources (the set of monitoring links) can be used partially or totally for functional user traffic.

Disadvantage: One potential disadvantage of this solution is that the routers are limited to a maximum number of ports.

6.3.3 Common Physical Interconnect and Shared Physical NoC Resources

A third possibility is to use the existing user NoC for the user traffic and also for the monitoring traffic. Both would share all the NoC resources but we keep the NoC user traffic and the monitoring traffic separated. In this way, a virtual NoC for monitoring is created.

Design Flow Impact: Considering the user requirements, the user NoC is obtained, by going through the reference NoC design flow from Figure 6.2(a). In this way, the topology and mapping are computed. After this, the monitoring communication requirements and monitoring IPs are computed and probes are added to the design. Figure 6.1(d) shows that probes are connected to the existing NoC by means of an extra port on the existing user NIs, as opposed to separate NIs for monitoring in the previous two cases. All the links, NI and router links, are considered shared between the user and the monitoring traffic. The mapping of the probes to existing NIs is based on the closest available NI. Path selection and slot allocation is computed together for all the communication requirements: user and monitoring. There are two possible cases:

(1) Everything fits on the existing user NoC. This means that the user NoC can accommodate the monitoring communication requirements on top of the existing user communication requirements. Topology of the NoC will therefore not change. This is exactly the situation shown in Figure 6.1(d). In this case, we have the lowest area cost, as no new NoC components, routers and NIs for monitoring, are added, except the new NI ports to connect the probes to the NoC.

(2) It does not fit on the existing user NoC. In this case, a new NoC must be generated, e.g., by increasing the topology and repeating the process. By increasing the topology, the number of user NoC routers increases and in turn the number of required monitoring probes may increase as well (e.g. if probing all routers is required). This leads to the recomputation of the monitoring communication requirements and monitoring IPs as shown in Figure 6.2(d). However, this process may not converge, i.e., a solution may not be found.

Non-intrusiveness: By sharing NoC resources, non-intrusiveness is potentially not guaranteed and must be enforced. The monitoring traffic can in this case interfere with the user traffic. For our Æthereal examples, this was not needed because we use GT for both functional and monitoring traffic and they cannot interfere. However, in general, extra steps may be required in order to enforce the non-intrusiveness.

Area cost: The total NoC area cost for our example is 2.75mm^2 (2.35mm^2 original + 0.4mm^2 extra). This was based on the addition of seven network interface ports, six for connecting the probes and one for the MSA. The added monitoring traffic fits in the original network.

Table 6.1: Comparison

	A	B	C
Design Flow	++	+	-
Non-intrusiveness	+	+	+/-
Area Cost	-	-	+
Reuse after debugging	-	+	+

Reuse: After completion of the monitoring task, the monitoring communication resources can be used for functional user traffic, e.g. by BE traffic.

6.3.4 Comparison

A brief overview, summarizing the advantages and disadvantages, of each of the proposed solutions is shown in Table 6.1. A, B and C are the solutions proposed in Sections 6.3.1, 6.3.2 and 6.3.3 respectively.

The table shows that having separate NoCs or NoC resources just for monitoring, as A and B, is both non-intrusive and basically straightforward in the NoC design flow; however it shows a high area cost in both cases.

Having shared resources for user traffic as well as for monitoring traffic is a good idea area-wise but may have strong implications on both the NoC design flow and the non-intrusiveness. However, both of these can be alleviated, as previously explained. Furthermore, it enables reuse of the shared resources by the functional traffic after the monitoring task is done.

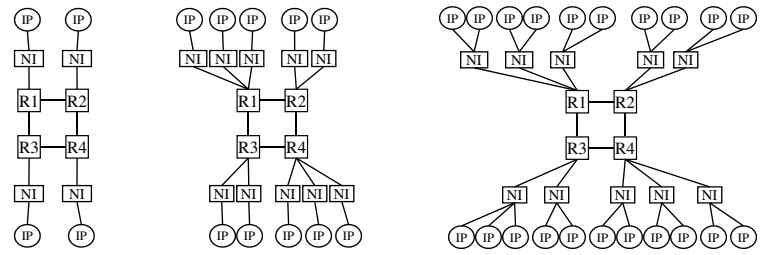
Based on this evaluation, we work out option C in more detail, in the following sections, although the discussion on application-aware placement of monitors in the next section, is valid for the other options as well with only little change. It is the most challenging option from the design-flow point of view.

6.4 Application Aware Placement

Monitoring at routers versus NI versus IP

Three options are possible for attaching monitors in NoCs: to routers, to NIs and to IPs. In Chapter 3 we have presented our choice to attach monitors to routers and not to NIs or IPs. We now elaborate on the choice we have made in this work. This is related with the possible IP-NI-R relations depicted in Chapter 2, which are common in current NoC designs.

Figure 6.3 presents three 2x2 mesh NoC examples which covers the three IP-NI-R relations depicted in Chapter 2: (1) one IP per NI combined with one NI per

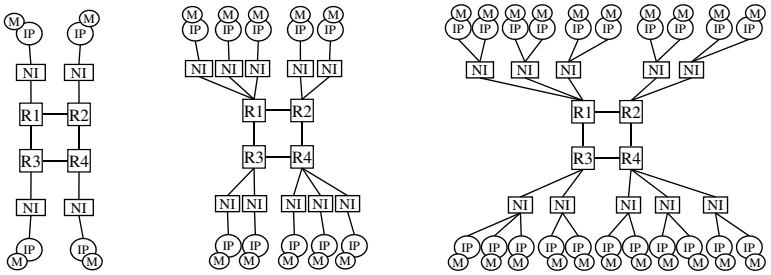


(a) NoC1: IPs/NI one, NIs/R one

(b) NoC2: IPs/NI one, NIs/R more

(c) NoC3: IPs/NI more, NIs/R more

Figure 6.3: Comparison of Monitor Placement: original NoCs (NoC1, NoC2, and NoC3) without monitors

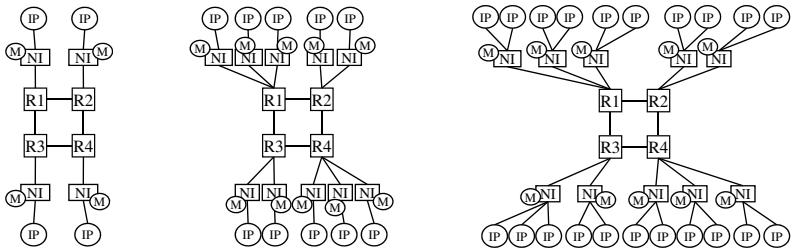


(a) IPs/NI one, NIs/R one

(b) IPs/NI one, NIs/R more

(c) IPs/NI more, NIs/R more

Figure 6.4: Comparison of Monitor Placement: monitors at IPs

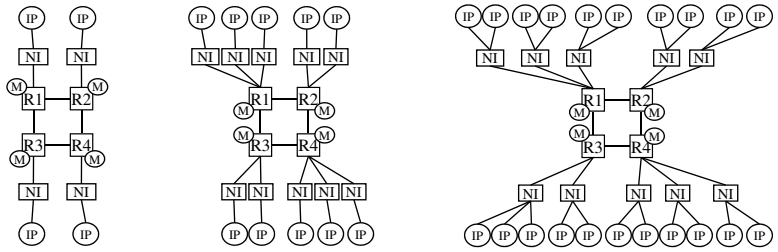


(a) IPs/NI one, NIs/R one

(b) IPs/NI one, NIs/R more

(c) IPs/NI more, NIs/R more

Figure 6.5: Comparison of Monitor Placement: monitors at NIs



(a) IPs/NI one, NIs/R one

(b) IPs/NI one, NIs/R more

(c) IPs/NI more, NIs/R more

Figure 6.6: Comparison of Monitor Placement: monitors at routers

router, (2) one IP per NI combined with more NIs per router, and the most general case (3) multiple IPs per NI combined with more NIs per router. For the sake of comparison we assume that all IPs can be instrumented with ETM-like monitors. Figure 6.4 show the original NoCs instrumented with monitors at IP level, one per IP. Figure 6.5 depicts the same three NoCs, now with monitors attached to NIs, one monitor per NI. The same NoCs of Figure 6.3 are now depicted with router monitors in Figure 6.6. Table 6.2 shows the comparison regarding the required number of monitors for the three NoCs at IP, NI and router level.

- *(probing at IPs)* The IP monitors, e.g. like ETM [5], can be attached to the IPs in the design. We have to point out that there are no generic IP monitors, e.g. ARM uses ETM monitors, and even these ETMs are specific (in architecture, hence in cost) to the ARM processor used; e.g. the ETM7 is used for ARM7 family of processors, while the ETM10 is used for the ARM10 family of processors. Also, some IPs do not posses any IP monitor as this was never provided or made by the IP provider; therefore, these IPs cannot be traced. The IP monitors operate directly at the IP transaction level; their main activity is to track these transactions, although some IP monitors offer more advanced options e.g. watchpointing. Transaction decoding is not required here, as these transactions are directly visible at the IP level (e.g. no packetization did previously occur). Relative to our proposed NoC monitoring interconnect, the monitoring NoC, they can be attached like any other IP. IP monitors can also be attached to NIs, on the IP side, one for each IP involved, but this case is considered here as probing at IPs. It can be seen from the three IP-NI-R options that probing at IP level becomes more expensive (in number of monitors) as more IPs get clustered per NI and more NIs get clustered per router. However, when using one IP per NI and one NI per router, it makes no difference whether the probing is done at IPs or at other levels, NIs or routers.

Table 6.2: Number of NoC Monitors

Designs	*NoC1*	*NoC2*	*NoC3*
IPs	4	10	21
NIs	4	10	10
routers	4	4	4
monitors at IPs	4	10	21
monitors at NIs	4	10	10
monitors at routers	4	4	4

- *(probing at NIs)* NI is the place where the packetization of the IP data takes place. Therefore, monitoring can be done at two levels here: at IP level, on the IP side of the NI, or on the router side of the NI. For the latter, the transaction monitors described in Chapter 4 can be used. The latter is also the NI monitoring case considered in this work, in Figure 6.5. The NI also takes care of handling of multiple IPs with multiple connections; this means that NI is a point where traffic from multiple IPs converges, making it a better point of observation than a single IP. It can be seen from the three IP-NI-R options that instrumenting NIs with monitors is in most cases better than instrumenting IPs with monitors, as IPs get clustered per NI, while it is still worse than instrumenting at routers when multiple NIs get clustered per router.

- *(probing at routers)* the transaction monitors described can be attached to routers. Routers are NoC components where traffic from many NIs converges, which make them more suitable from monitoring than the NIs. The IP level transactions are not directly visible at this level. They should be able to do at least transaction decoding from the router link data stream, which involve the process of intercepting the data stream, the isolating the proper connection and then depacketize its data and rebuild the messages which compose the transactions, as depicted in Chapter 4. This process will bring the capabilities of the router monitor to the IP monitor level. It can be seen from the three IP-NI-R that probing at routers is always the most effective in number of monitors, and tends to be much better when both IPs and NIs get clustered per NI and router respectively.

While this comparison is only quantitative in the number of monitors and not in overall area or monitor hardware design complexity, as previously pointed out, it is important to note that the number of monitors is relevant in many aspects. Each of the employed monitors has to be configured, and reconfigured at run-time; less monitors means less data for configuration and higher overall configuration latency for the monitoring system as each of the monitors has to be configured in isolation. Each of the connections employed has to be handled by the NoC design flow, and the fewer monitoring connections would make it easier. Also, the

integration aspects for fewer monitors must be noted. The number of monitors is also a main factor relative to the overall area.

To conclude, the placement of monitors at routers would reduce the overall number of needed monitors at the cost of a higher monitor complexity, by placing the monitors at routers, the points in the NoC where traffic from multiple IPs via multiple NIs converges.

Placing Transaction Monitors

To work out in detail the previously selected option of sharing a single NoC for monitoring as well as for the application traffic we use the NoC monitoring service of Chapter 3. This was instantiated for debugging with transaction monitors, using centralized monitoring with a single MSA, and only considering transaction monitors attached to routers.

The transaction monitoring problem has been presented in Chapter 4, with the focus on the capabilities of single transaction monitors. The chapter showed how single transaction monitors can ultimately track transactions over any single channel passing any of the router's links, with the tracked channel being (re-)programmed at run-time.

The problem of how many transaction monitors are needed relates to the desired coverage of the user communication flows. In general a full channel coverage is desired; this means that we want to be able to monitor all the possible transactions in the system. However, it is prohibitively expensive to duplicate all traffic in the NoC; therefore the coverage may be full but has to be selective at certain moments in time. This means that the transaction monitors must cover all channels, but not at the same time. At run-time, any (potentially more) of the desired channels can be selected to be monitored. The concurrent observation of multiple channels or the number of simultaneously active transaction monitors in the system is only bounded by the number of transaction monitors deployed, as each transaction monitor can only track a single channel.

As a requirement, at least one transaction monitor is required on the path of each channel, regardless whether it is a request or response channel. This means that any of the existing channels can be probed, achieving a full channel coverage. At run-time, the monitored channels per probe may change by means of programming the transaction monitors. This selectivity is acceptable as usually not all streams are required to be monitored at once.

Since we are considering ASIC-like design, the application is known at design time. For the NoC-based SoC, it means that also the set of connections (all request and response channels) is known at design time. The bandwidth and latency constraints of the channels are determined beforehand by means of static analysis or simulation. This implies the placement of transaction monitors in order to offer a full channel coverage of the system, the placement of the MSA, and the dimensioning of the communication requirements of these monitors, as this data should go to the MSA via the NoC.

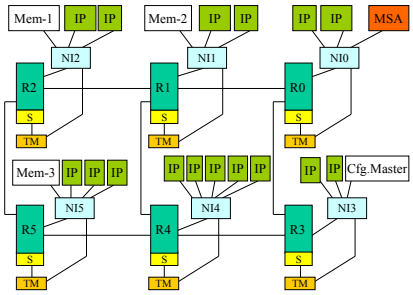
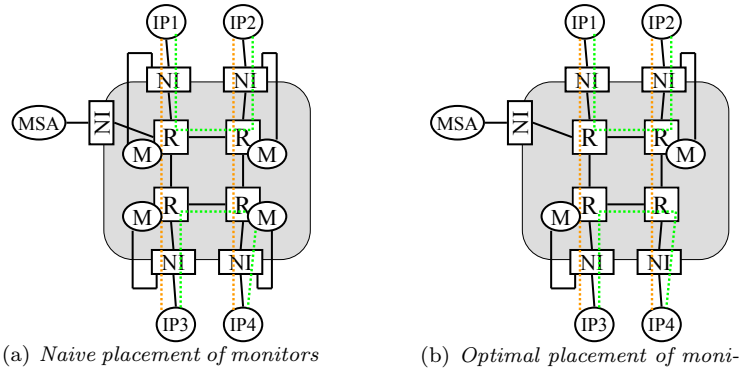


Figure 6.7: NoC Monitoring Service



(a) *Naive placement of monitors*

(b) *Optimal placement of monitors*

Figure 6.8: Placement of Transaction Monitors

In ASIC design, a full coverage of routers with monitors may potentially be avoided while maintaining a full coverage of the user communication flows. See for example the four monitors in Figure 6.8(a) covering each one of the four channels, versus the two monitors in Figure 6.8(b) covering each the two channels passing through. Both offer a full coverage of the flows while only the former offers also a full router coverage. This leads to a reduction of the total monitoring solution area cost. Note that in all the previous chapters a full router coverage has been assumed. Note also that assuming a full coverage of NoC routers with transaction monitors (1) implies a full coverage of the user communication flows, and (2) the communication requirements of these monitors are also not known before the path selection step in the NoC design flow, as we do not know earlier what channels will pass over each of the monitored routers. Therefore, the problem of modifying the design flow to support monitoring constraints cannot be avoided. In order to illustrate the required design flow modifications in the following section, we use the UMARS NoC design flow.

An application aware placement of monitors has a positive impact on the overall monitoring costs. In this particular case, the problem of the cost implications of the monitoring relates to the area of the resulting NoC which supports both the application and monitoring communication requirements, the area of the monitors, and also to the number of transaction monitors involved. The resulting NoC, potentially larger than the original NoC, accounts for the extra NIs, NI ports or enlarged topology to support monitoring in addition to the application communication.

6.5 NoC Design Flow Revisited

An important property of UMARS, as presented in Chapter 2, that we exploit in this work is the fact that channels are naturally allocated ordered on their bandwidth requirements. This ordering assures us that no channel succeeding the one currently being allocated has higher bandwidth requirements.

The proposed monitoring-aware NoC design flow is depicted in Figure 6.9. The coupling of mapping, path selection and time-slot allocation from the original UMARS is extended with the mapping of transaction monitors to routers such that a full coverage of user channels is achieved. Here, we do not discuss the mapping, path selection and slot allocation; for these please refer to [46]

As a *preprocessing* step to the modified UMARS, transaction monitors are virtually added to all routers (as this would be the maximum set of transaction monitors that we consider). These virtual monitors are added to the set of IPs present in the system. They are connected to the closest local NI, attached to the router they monitor.

Due to centralized monitoring used, a single MSA is further added to the set of IPs and it gets its own NI. A single GT connection is assumed from any monitor to the MSA although yet of unknown required bandwidth. We consider monitoring connections as latency insensitive, so no latency constraints are added to them.

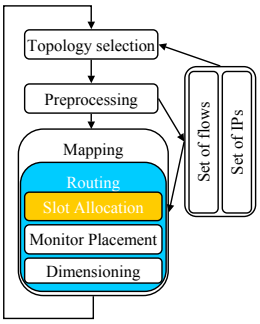


Figure 6.9: Monitoring-aware design flow

Monitor Placement. The loop of Algorithm 2.3.1 is extended with a fourth step, after a channel is allocated. This step is described in Algorithm 6.5.1. First, we check whether we need to insert additional monitoring. If the channel passes through a router that is monitored, we know, as channels are traversed in decreasing bandwidth order, that the monitor is able to monitor also this channel. Hence, nothing changes in this case. However, if none of the routers that the channel passes through are yet monitored, we select one router on the path in Step 1a of Algorithm 6.5.1. We select a router with the highest arity on the channel path, because it maximizes the number of potential observed channels for this monitor. Once we select the router to be probed we are sure that the router will stay in the final set of transaction monitors. Therefore, the virtually probed router is added to the set of probed router.

In Step 1b of Algorithm 6.5.1 we then add a channel from the now monitored router (and its associated NI) to the MSA. This channel is added to the set of unallocated flows.

Dimensioning. The requirements in terms of bandwidth is derived as a function of the channel that mandated the insertion of the probe. Note that the way in which the communication requirements are dimensioned does not impact the overall proposed design flow. For the transaction monitoring example, we set the traffic numbers for the monitoring channels equal to the bandwidth required by the monitored channel. The next channel to be monitored by the same monitor, whose monitoring channel has been allocated, is guaranteed to require a lower bandwidth. As one monitor can only monitor one channel at a time, the previously allocated monitoring channel would be reused. The same holds if the monitoring channels would require, e.g., 10% of the monitored connection bandwidth, due to a higher abstraction power of the monitors.

Algorithm 6.5.1 Step four

1. If the path does not pass a monitored router
   (a) Select a router on the path
   (b) Add a channel from this router to the MSA

The newly added channel is a monitoring channel. The only difference between a genuine user channel and a monitoring channel is that we only want to monitor the user channels. Besides allocating the user and monitoring channels we also take care not to monitor the monitoring channels. Therefore, Step 1b of Algorithm 6.5.1 is only executed for user channels.

Results. If UMARS completes the allocation with success, we have as results the mapping, routing, slot allocation, monitor placement and monitoring dimensioning. After UMARS completes the allocation for all flows, all the routers in the set of probed routers have monitors attached. All the rest of the virtual monitors

are removed, as well as all the unallocated monitoring flows.

Iterations. If an allocation was not found by varying the slot table size till some predefined upper limit, the topology can be increased and the process repeated.

6.6 Experiments

6.6.1 Application Examples

Real Examples

We have used two real applications. *(mpeg)* an mpeg2 encoder/decoder using the main profile (4:2:0 chroma sampling) at main level (720x480 resolution with 15Mb/s) supporting interlaced video up to 30 frames per second. This application consists of 15 processing cores and an external SDRAM, and has 42 channels (with an aggregated bandwidth of 3GB/s), all configured to use guaranteed throughput, as presented in [40].

(audio) this application performs sample rate conversion, MP3, audio-postprocessing and radio. It closely resembles the chip presented in [62]. The application consists of 18 cores and has 66 channels all configured to use guaranteed throughput.

We have combined the two applications into four cases to be used as examples: mpeg (*Design1*), mpeg + audio (*Design2*), 2 × mpeg + audio (*Design3*), 4 × mpeg + audio (*Design4*).

Synthetic Examples

We have also generated synthetic application benchmarks for testing our proposed design flow. These benchmarks are structured to follow the application patterns of real SoCs. We have generated applications into two classes of such benchmarks, as presented in [65]:

(i) Spread communication benchmarks (*Spread*), where each core communicates to a few other cores. These benchmarks characterize designs such as the TV processor that has many small local memories with communication evenly spread in the design.

(ii) Bottleneck communication benchmarks (*Bottleneck*) where there are one or multiple bottleneck vertices to which the core communication takes place. These benchmarks resemble designs using shared memory/external devices such as the set-top boxes.

We have used spread communication of 12 IPs, in which every IP communicates with three others. We have used bottleneck communication with two converging points and 12 IPs. We have generated 500 synthetic application examples with spread and 500 with bottleneck communication.

Table 6.3: Real Examples

Designs	*area*	*inc*	*size*	*mon*	*slot table*
1NI/R					
Design1	5.15	-	2x4	-	21
Design1+M	5.43	+5.5%	2x4	5	27
Design2	8.75	-	3x3	-	30
Design2+M	10.16	+16.1%	3x4	10	27
Design3	12.03	-	3x4	-	44
Design3+M	13.95	+16%	3x4	9	60
2NIs/R					
Design1	4.03	-	1x4	-	21
Design1+M	4.12	+2.2%	2x2	3	20
Design2	7.88	-	2x3	-	20
Design2+M	8.2	+3.9%	2x3	6	20
Design3	10.82	-	3x3	-	22
Design3+M	11.64	+7.6 %	2x4	8	29
3NIs/R					
Design1	3.62	-	1x2	-	30
Design1+M	3.85	+6.3%	1x3	3	18
Design2	6.97	-	1x3	-	27
Design2+M	7.16	+5.4%	1x3	3	30
Design3	10.26	-	2x3	-	21
Design3+M	10.78	+5%	2x3	6	22
Design4	18.45	-	3x4	-	21
Design4+M	19.07	+3.4%	2x4	8	36

6.6.2 Results

Setup

For both the real and synthetic application examples, we have investigated what the original UMARS vs. the monitoring-aware UMARS output is. The original UMARS generates the minimal NoC on which only the application requirements fit, while the monitoring-aware UMARS generates the minimal NoC on which both the application and monitoring requirements fit. To evaluate the performance of our approach, we looked at:

(i) required number of transaction monitors,
(ii) resulting topology size,
(iii) resulting slot table size and
(iv) resulting area.

For each application, we have evaluated all possible meshes, from one by one up to seven by seven. For each of these topologies we have added one, two and three NIs per router, as depicted in Figure 6.10 and evaluated slot table sizes

up to 65 TDMA slots. A larger slot table size mitigates overprovisioning due to granularity, but is often associated with a growth in buffer sizes as network consumption tends to become more bursty. Out of all the configurations for which UMARS finds an allocation, we present the one with lowest total area cost.

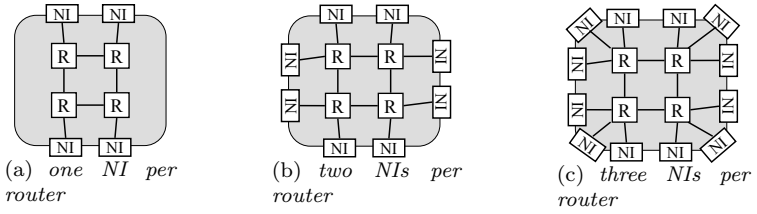


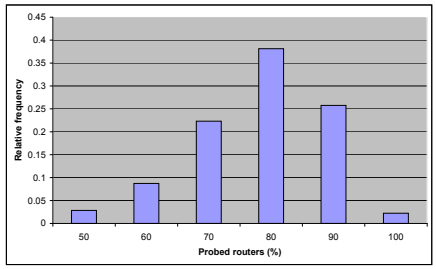
Figure 6.10: NIs per router

Table 6.3 summarizes the results for the real examples, when one, two or three NIs per routers are tried. Due to the large communication demands, and given the constraints on topology and slot-table size we set for our experiments, *Design4* only fits on a topology using three NIs per router. For the synthetic examples, Figures 6.11 and 6.12 summarize the results for bottleneck and spread communication respectively. Each of the four aspects is discussed in detail in the following paragraphs.

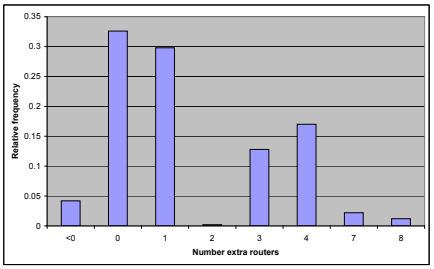
Number of transaction monitors

For the synthetic cases with bottleneck communication, we see that the number of routers needed to be probed for full coverage varies between 50% and 100% with an average of 75%. Figure 6.11(a) displays the distribution. For spread communication, Figure 6.12(a) displays the distribution. We see that the number of routers requiring a probe is higher compared to the bottleneck cases, but that is no surprise as the communication is more balanced (spread out) over the routers. The minimum is 60% while the maximum is 90%. Hence, the interval is narrower than with bottleneck communication, the maximum is actually lower, and coverage of all routers was never required. Looking at the diagrams it is obvious that the number of routers needing probes is focused around the 80-90% bins.

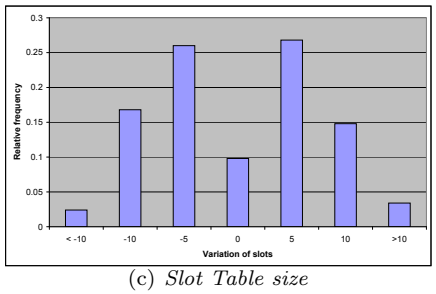
Please note that the number of transaction monitors required is high because the Æthereal NoC allows multiple IPs to be connected to the same NI and multiple NIs to be connected to the same router. Therefore, channels can be very short, e.g., a channel between a master and a slave connected to the same NI will go through the NI starting from the master, then through one router and back to the same NI to the slave, see Section 2.1 for more details. All routers having at least one channel like this passing through will require one transaction monitor. Other NoCs may require a channel to pass through two different NIs, potentially lowering the number of transaction monitors being required.



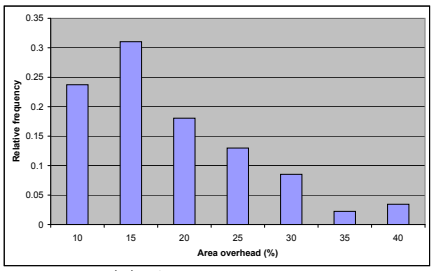
(a) *Number of Probed Routers*



(b) *Topology Increase*



(c) *Slot Table size*



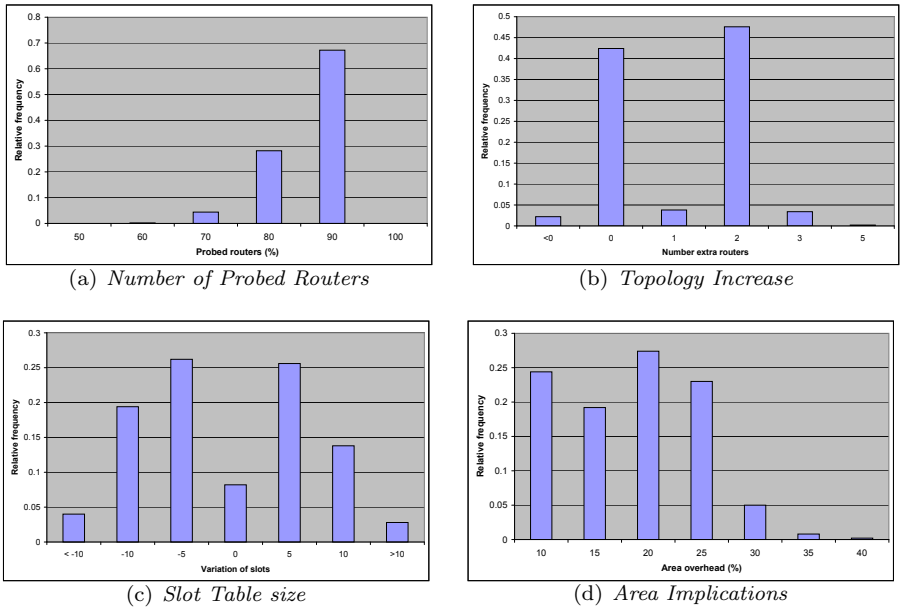
(d) *Area Implications*

Figure 6.11: Bottleneck

For the real examples, see column *mon* in Table 6.3, showing the number of monitors and compare it to column *size* showing the mesh size. On average 87% of the routers need to be probed, but full coverage of routers with probes was required in 60% of the cases. Relating this with the area numbers from the same table, it is interesting to observe that the most area-efficient solutions required all routers probed. Therefore probing all routers must not be associated with area-inefficient solutions, the number of monitoring probes (in our case transaction monitors) being just one component which influences the total area cost of the monitoring solution.

Topology size

For the topology size, we looked at the total number of routers employed. Figures 6.11(b) and 6.12(b) display the distribution for the synthetic examples. On average, topology stays the same (no extra routers required) or one or two extra routers are required. Increases in topology size with more than two routers, but with a maximum of 8, are required in some cases, especially in the bottleneck applications. This can be explained because in bottleneck designs it is harder to accommodate the new monitoring channels due to the existing bottleneck vertices. Interesting is the fact that in 3-4% of the cases the number of routers actually decreased. This we can attribute to the heuristic nature of UMARS and to the



(a) *Number of Probed Routers* (b) *Topology Increase*

(c) *Slot Table size* (d) *Area Implications*

Figure 6.12: Spread

higher number of slots used in the NoCs with monitoring.

For the real examples, we see the number of routers kept constant in six cases, and both an increase and a decrease in two cases. The latter is accountable to an allocation found with a higher slot table size, see column *slot table* in Table 6.3.

In both real and synthetic examples, we see that there is a good chance(30-60%) to find a solution on the same NoC topology, without requiring extra routers.

Slot Table size

Figures 6.11(c) and 6.12(c) display the distribution of the slot table size variation for the synthetic examples. In general, we can see a similar shape for both bottleneck and spread communication examples. In a small number of cases (up to 10%) the slot table size is constant. It varies within a limit of +/- 5 slots on a cumulated 50% of the cases. In roughly 30% of the cases the variation is between 5 and 10 slots, either in the negative or positive part. Higher variations than 10 slots are least visible in the figures.

For the real examples, we can observe the slot table size being constant in one case, bigger in six cases and smaller in three. Clearly, there exists a relation between the NoC topology and the slot table size.

In general, a higher number of slots corresponds to adding the monitoring communication requirements on the same (or eventually smaller) NoC topology

than the one used for user only communication requirements. A lower number of slots corresponds to a bigger NoC topology in the resulting shared NoC. The adapted UMARS design flow tries to balance these aspects.

Area

The total NoC area is derived according to the model in [37] extended with the area of the transaction monitors, $0.026mm^2$ per monitor in 0.13μm CMOS technology. Note that the total area presented includes NIs, routers and probes (transaction monitors). The area of NIs also accounts for buffer sizing in the NIs/NI ports corresponding to the real communication requirements of the users and monitors. The area numbers do not include the area of other IPs in the SoC, but refer to the NoC together with the complete monitoring service.

For bottleneck communication, area wise the cost is continuously below 50% with an average of 15%. Figure 6.11(d) shows the distribution of area overhead over the test cases and it is obvious that most lie in the left half of the span.

For spread communication, Figure 6.12(d) shows the distribution. From an area point of view, the overhead is between 10% and 40%, which again is a narrower interval than for bottleneck communication. In all, it ends up on an average of 15% also for uniform traffic. No major difference in the area overhead is noticeable between uniform and bottleneck communication.

For the real examples, the total area increase, see column *inc* in Table 6.3, amounts to between 2.2% and 16.1%. The area overhead is between 3% and 7% in the most area efficient case of three NIs per router which succeeded for all four designs. We consider the resulting four designs the end results of the monitoring-aware NoC design flow.

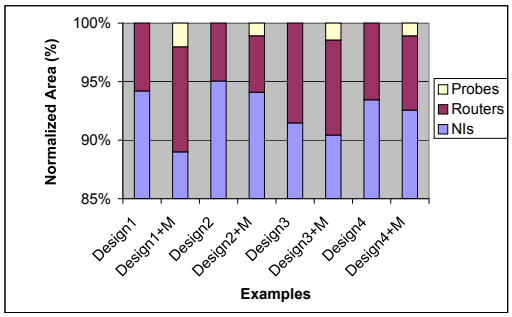


Figure 6.13: Overall distribution of area

It is also interesting to see the overall distribution of this area between NIs, routers and monitors. This is presented in Figure 6.13 for the four designs in their most area-efficient case using *3NIs/R*. For the original designs, the distribution of area between NIs and routers is shown. The main remark is that in all cases

area of the transaction monitors is insignificant relative to the total area of the designs, dominated by the area of the NIs. Furthermore, in all designs, the area of the monitors is even several times lower than the area of the routers involved. These overall area numbers confirm the previously mentioned area numbers as well as the target area and initial estimation of Chapter 3.

6.7 Conclusion

In this chapter, we have presented three architectural options for a NoC monitoring service supporting a chip-wide monitoring system. All options are generic and can be applied to any NoC. Each of the presented options is NoC-based and scalable. Non-intrusiveness, influences on the overall NoC design flow, area, and reuse potential are evaluated for all these options. An interesting trade-off is presented showing that what is good for area and reusability requires efforts in modifying the NoC design flow and in preserving the non-intrusiveness of the monitoring system. Sharing all NoC resources among the application and the NoCMS provides the most area efficient and reusable design, but it has the most impact on the design flow and on the intrusiveness.

Based on the area efficiency, we propose a NoC design flow for the case in which NoC resources are shared between application and NoCMS. The flow takes monitoring into account at design time and it is fully integrated in the flow, automating the insertion of the monitors whenever their communication requirements are known, leading to a monitoring-aware NoC design flow. Our flow was exemplified with the concrete case of transaction monitoring, in the context of the Æthereal NoC and UMARS design flow.

We are the first to quantify the complete cost of the complete monitoring solution accounting for the monitors, extra NIs, NI ports or enlarged topology to support monitoring in addition to the original application communication. Results show an area efficient solution for integrating monitoring in NoC designs. Monitors alone do not add much to the overall area numbers as the designs remain dominated by the area of NIs.

Chapter 7

Conclusions

This last chapter of this work presents the main conclusions. It is the intention of the author to summarize and integrate all chapter conclusions in order to present a consistent overall picture of the achievements. A few remaining open problems and some interesting future research ideas are also detailed here. This is done to show that there are options to continue this work. The chapter ends with the author's personal view on the future of NoC monitoring.

7.1 Contributions

As one of the main contributions of this work we have presented the generic concepts of a NoC monitoring service, the first one described in the scientific literature. This is the basis of our contributions, the rest of them building upon it. The monitoring service offers communication observability at run-time. It is generic and can be instantiated for the monitoring task at hand. We have instantiated it for application and system-level debugging, and for run-time performance analysis in the form of a transaction monitoring system and performance monitoring system respectively.

The monitoring service can be configured and used at arbitrary moments during run-time, offering increased flexibility. The monitoring service is integrated in the NoC and uses the NoC communication services (e.g. guaranteed throughput services) for configuration as well as for the run-time generated monitoring traffic. It can be instantiated automatically together with the NoC, saving design time. The monitoring service consists of monitors attached to NoC components, allowing easy scalability of the service, and monitoring service access points, the points where the monitoring service can be setup and monitored data can be accessed. The generic architectural concepts of the monitor feature a programmable modular design composed of a sniffer, a monitoring network interface and an event generator, providing flexibility to target the service to the monitoring task at

hand. Proof of concept is achieved via implementation for the Æthereal NoC, for all the particular cases presented in this work, although the generic concepts presented allow to re-target our proposed NoC monitoring service to other NoCs.

For debug purposes in general, reconstructing transactions at run-time is essential. The transaction level analysis of the NoC behavior makes the complete MPSoC easier to understand. The NoC is a suitable place to monitor the internals of a SoC at multiple levels of abstraction, including transaction level. We have proposed one specific instance of the generic NoCMS, a transaction monitoring system, in the form a NoC analyzer able to perform run-time NoC transaction monitoring. This NoC analyzer alleviates the run-time observability problem by providing hardware transaction monitors able to work on four different levels of abstraction. They correspond to four analyzer modes, ultimately being able to on-chip reconstruct transactions from low-level monitored router data and abstract them to events.

Due to nonalignment of packets and messages, it is generally difficult to go beyond the raw low-level data (bits), to understand what data means (transactions), in order to (re)construct a transaction-level view from the data stream of a connection. We have conceptually shown how this problem can be solved for all existing packetization schemes. Thus our concepts can be reused for any existing NoC.

The two specific instances of our NoCMS that we developed are both based on an architecture in which the NoC resources are shared between applications and the NoCMS. In total, we explored three architectural options for a NoC monitoring service supporting a chip-wide monitoring system, the one already mentioned, one option that completely separates the application NoC from the monitoring NoC, and one in which the applications and the NoCMS are connected by the same NoC but share separate resources. All options are generic and can be applied to any NoC. Each of the presented options is NoC-based and scalable. Non-intrusiveness, influences on the overall NoC design flow, area, and reuse potential are evaluated for all these options. An interesting trade-off is presented showing that what is good for area and reusability requires efforts in modifying the NoC design flow and in preserving the non-intrusiveness of the monitoring system. Sharing all NoC resources among the application and the NoCMS provides the most area efficient and reusable design, but it has the most impact on the design flow and on the intrusiveness.

Based on the area efficiency, we propose a NoC design flow for the case in which NoC resources are shared between application and NoCMS. The flow takes monitoring into account at design time and it is fully integrated in the flow, automating the insertion of the monitors whenever their communication requirements are known, leading to a monitoring-aware NoC design flow. Our flow was exemplified with the concrete case of transaction monitoring, in the context of the Æthereal NoC and UMARS design flow.

For the area cost for the entire NoC monitoring service, the experiments show that the target of 15-20% of the total NoC area for monitoring only, is realistic in

general. The same NoCMS is remarkably diligent in the two specialized instances we developed, the transaction monitoring and the performance monitoring, the overall monitoring area being dwarfed by the rest of the NoC area, e.g., only 5% of the NoC area is used for the transaction monitoring system for several MPEG/audio SoC case studies.

A transaction monitor for the most difficult packetization scheme was implemented at the cost of one fifth of the router area. A transaction monitor has an area cost of 0.026mm^2 in a 0.13μm CMOS technology. The area results for the performance monitor employed to track link utilization, attached to an arity 6 GT/BE router is 0.018mm^2 in the same 0.13μm CMOS technology.

We are the first to quantify the complete cost of the complete monitoring solution accounting for the monitors (including complex transaction monitors), extra network interfaces and ports, or enlarged topology need to support monitoring in addition to the original application communication. Results show an area efficient solution for integrating monitoring in NoC designs. Monitors alone do not add much to the overall area numbers as the designs remain dominated by the area of network interfaces.

7.2 Open Problems and Future Research

This entire work on monitoring merely provides the knobs and controls for establishing and tuning a structured NoCMS for either transaction-based debugging or performance debugging and optimization. The development of such a methods using the NoCMS is a separate challenge in its own. Therefore, one first open research item is an actual transaction-based debugging method based on or using our transaction monitors and the NoCMS. To effectively debug systems, besides monitoring we also need control; therefore, at least, control will have to be added to our work. Furthermore, in order to achieve a complete SoC monitoring solution future work will have to include the integration of computation observability with the NoC monitoring service

Run-time performance monitoring at a reasonable cost enables the development of QoS-like methods at different levels, e.g. at the application or network manager levels. The development of for example a network manager, that manages all the NoC resources and interacts with one or several application/system QoS managers, is still open.

Methods and the tools based on our NoCMS will actually show the gains of using the NoCMS, and will make the NoCMS an indispensable part of future NoCs.

As opposed to ASICs, a new generation of general chip multiprocessor systems arises. In these systems the set of applications to be mapped onto such a multiprocessor is not known at design time, as is the case for ASICs. For this reason, one has to assume a given (minimal) placement of monitors, e.g. transaction monitors, given certain design requirements and constraints. The development of such

placement methods for large NoCs is an interesting topic for future work. For a given placement of monitors, connections will have to be routed through transaction monitors, and the effects and costs of placement and routing compared to the original NoCs and to the already presented solution (for ASICs) needs to be quantified.

7.3 Personal Opinions

At the end of this work I feel that I finally have the occasion to say what I felt form the beginning of my work, and I think that this is the proper place to do it. I strongly believe that NoC monitoring will become an important issue in the future and it will develop into a standard component for basically all next-generation NoCs, in one form or another. While this is a (strong) personal opinion, I do not try to prove it but I can give a brief two-way reasoning supporting it: why is this not yet the case, and why will it be the case in the future.

The reasons why this is not yet the case are twofold. First, we should consider that current NoCs are fairly small and predictable design of systems based on such small NoCs offering hard timing guarantees is possible. The second reason is that NoCs are just at the beginning of their presumably long life and the research community is still eager to prove their value, by looking at their design, and the design of systems using them, in particular trying to show the advantages over busses or the ease of design via automated NoC design flows. The monitoring for either performance analysis or debugging is therefore not yet a hot topic.

The reasons why monitoring will finally arrive in NoCs are also twofold, a perfect image of the former reasons. First, there is a huge difference between today's small NoCs and tomorrow's required huge NoCs; summarizing, size matters. Such NoCs will only be able to offer overall statistical guarantees, coupled with hard guarantees for small parts of the NoCs. If such situations arise then the monitoring will be in one form or another indispensable for run-time management of such systems. There will be no other way of saying what is going on in the NoC at a certain time. Also with the size of such systems, run-time monitoring at higher levels of abstraction (transactions) will become the only way of debugging.

The second reason is the personalization. This is because we move towards the time when NoC-based SoCs are incorporated into devices surrounding us which tend to be intelligent and personifiable. Therefore the set of applications running onto them will be dependent of us, not known in advance, and probably unique for each person and learning our habits, as opposed to today's devices that run the same software/firmware for everyone, the trend of personalization just arising. This will make monitoring the only means to enable the run-time resource and quality management.

Bibliography

[1] D. Andreasson and S. Kumar. Improving be traffic qos using gt slack in noc systems. In *NORCHIP Conference*, volume 23, pages 44–47, Nov. 2005.

[2] ARM. http://www.arm.com/products/CPUs/families.html.

[3] ARM. *Coresight*. http://www.arm.com/products/solutions/CoreSight.html.

[4] ARM. *Embedded Trace Buffer*. www.arm.com, 2002.

[5] ARM. *Embedded Trace Macrocell*. www.arm.com, 2002.

[6] ARM. *AMBA AXI Protocol Specification*, June 2003.

[7] Arteris. www.arteris.com.

[8] Arteris. A comparison of network-on-chip and busses. White paper, 2005. Available on www.arteris.com.

[9] A. A. Bayazit and S. Malik. Complementary use of runtime validation and model checking. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 1052–1059, Washington, DC, USA, 2005. IEEE Computer Society.

[10] L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.

[11] Bindesh Patel, Sean Smith. http://www.design-reuse.com/articles/12790/transaction-based-debug-of-pci-express-embedded-soc-platforms.html, 2008.

[12] T. Bjerregaard. *The MANGO clockless network-on-chip: Concepts and implementation*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2006.

[13] T. Bjerregaard and J. Sparsø. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1226–1231, Mar. 2005.

[14] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2–3):105–128, Feb. 2004. Special issue on Networks on Chip.

[15] S. Carta, A. Acquaviva, P. G. D. Valle, D. Atienza, G. D. Micheli, F. Rincon, L. Benini, and J. M. Mendias. Multi-processor operating system emulation framework with thermal feedback for systems-on-chip. In *GLSVLSI '07: Proceedings of the 17th ACM Great Lakes symposium on VLSI*, pages 311–316, New York, NY, USA, 2007. ACM.

[16] I. Cidon and K. Goossens. Network and transport layers in networks on chip. In G. De Micheli and L. Benini, editors, *Networks on Chips: Technology and Tools*, The Morgan Kaufmann Series in Systems on Silicon, chapter 5, pages 147–202. Morgan Kaufmann, July 2006.

[17] C. Ciordas, T. Basten, A. Rădulescu, K. Goossens, and J. van Meerbergen. An event-based network-on-chip monitoring service. In *Proc. Workshop on High-Level Design Validation and Test (HLDVT)*, pages 149–154, Nov. 2004.

[18] C. Ciordas, T. Basten, A. Rădulescu, K. Goossens, and J. van Meerbergen. An event-based monitoring service for networks on chip. *ACM Transactions on Design Automation of Electronic Systems*, 10(4):702–723, Oct. 2005. HLDVT'04 Special Issue on Validation of Large Systems.

[19] C. Ciordas, K. Goossens, T. Basten, A. Rădulescu, and A. Boon. Transaction monitoring in networks on chip: The on-chip run-time perspective. In *Proc. Symposium on Industrial Embedded Systems (IES)*, Oct. 2006.

[20] C. Ciordas, K. Goossens, and A. Rădulescu. *Electronic device and method of communication resource allocation.* KONINKLIJKE PHILIPS ELECTRONICS N.V., May 2006. Patent Application WO2006051471.

[21] C. Ciordas, K. Goossens, and A. Rădulescu. *Electronic device, system on chip and method for monitoring a data flow.* NXP, BV., Jan. 2008. Patent Application WO2008004188.

[22] C. Ciordas, K. Goossens, A. Rădulescu, and T. Basten. NoC monitoring: Impact on the design flow. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*, pages 1981–1984, May 2006.

[23] C. Ciordas, A. Hansson, K. Goossens, and T. Basten. A monitoring-aware NoC design flow. In *Proc. Euromicro Symposium on Digital System Design*, Aug. 2006.

[24] C. Ciordas, A. Hansson, K. Goossens, and T. Basten. A monitoring-aware NoC design flow. *Elsevier Journal of Systems Architecture*, 54(3-4):397–410, March-April 2008. Special Issue with selected best papers of DSD2006.

[25] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. Design Automation Conference (DAC)*, pages 684–689, 2001.

[26] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[27] G. De Micheli. *Networks on Chips: Technology and Tools (Systems on Silicon).* 2006.

[28] J. A. de Oliveira and H. van Antwerpen. The Philips Nexperia digital video platform. In G. Martin and H. Chang, editors, *Winning the SoC Revolution.* Kluwer Academic, 2003.

[29] R. Drechsler. Towards formal verification on the system level. *Proc. Int'l Workshop on Rapid System Prototyping*, pages 2–5, 2004.

[30] J. Duato. *Interconnection Networks: an engineering approach.* 1997.

[31] S. Dutta, R. Jensen, and A. Rieckmann. Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. *IEEE Design and Test of Computers*, pages 21–31, Sept-Oct 2001.

[32] First Silicon Solutions. *BusNavigator.* http://www.fs2.com/busnavigator.html.

[33] First Silicon Solutions. *SiliconBackplane Navigator.*

[34] First Silicon Solutions. *Trace Instrumentation and Architectures for OCP busses.*

[35] S. Gheorghita, T. Basten, and H. Corporaal. Application scenarios in streaming-oriented embedded system design. In *International Symposium on System-on-Chip (SoC)*, pages 175–178, 2006.

[36] S. González Pestana, K. Goossens, A. Rădulescu, and R. Thid. Framework and performance metric definitions: A first step towards network-on-chip benchmarking. Technical Note 2006/00003, Philips Research, Jan. 2006.

[37] S. González Pestana, E. Rijpkema, A. Rădulescu, K. Goossens, and O. P. Gangwal. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 764–769, Feb. 2004.

[38] K. Goossens and C. Ciordas. *Electronic device, system on chip and method of monitoring data traffic.* NXP, BV., Jan. 2008. Patent Application WO2008004187.

[39] K. Goossens, C. Ciordas, and A. Rădulescu. *Electronic device, system on chip and method for monitoring data traffic.* NXP, BV., Jan. 2008. Patent Application WO2008004185.

[40] K. Goossens, J. Dielissen, O. P. Gangwal, S. González Pestana, A. Rădulescu, and E. Rijpkema. A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1182–1187, Mar. 2005.

[41] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):21–31, Sept-Oct 2005.

[42] K. Goossens, O. P. Gangwal, J. Röver, and A. P. Niranjan. Interconnect and memory organization in SOCs for advanced set-top boxes and TV — Evolution, analysis, and trends. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 15, pages 399–423. Kluwer, 2004.

[43] K. Goossens, B. Vermeulen, R. van Steeden, and M. Bennebroek. Transaction-based communication-centric debug. In *Proc. Int'l Symposium on Networks on Chip (NOCS)*, pages 195–206, May 2007.

[44] S. L. W. Group. *Functional Specification for SystemC 2.0.* www.systemc.org, 2001.

[45] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 250–256, 2000.

[46] A. Hansson, K. Goossens, and A. Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 75–80, Sept. 2005.

[47] M. He and K.-V. Ling. Model predictive control on a chip. In *(IEEE) Conference on Control and Automation (ICCA)*, 2005.

[48] J. J. H.S. Bhullar, R. van den Berg and F. Zegers. Serving digital radio and audio processing requirements with sea-of-dsps for automotive applications the philips way. In *IEEE ISPC GSPx*, 2004.

[49] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 688–693, 2003.

[50] A. Jalabert, S. Murali, L. Benini, and G. De Micheli. XpipesCompiler: A tool for instantiating application specific networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2004.

[51] Y. Jiang, C.-K. Tham, and C.-C. Ko. Challenges and approaches in providing qos monitoring. *International Journal of Network Management*, 10(6):323–334, 2000.

[52] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, Sept. 2002.

[53] N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, and P. T. Wolkotte. A virtual channel network-on-chip for gt and be traffic. In *ISVLSI '06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, page 211, Washington, DC, USA, 2006. IEEE Computer Society.

[54] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[55] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *Proc. Symposium on VLSI*, 2002.

[56] R. Leatherman and N. Stollon. An embedded debugging architecture for socs. *IEEE Potentials*, pages 12–16.

[57] M. Mansouri-Samani and M. Sloman. A configurable event service for distributed systems. In *Proc. Int'l Conference on Configurable Distributed Systems*, pages 210–220, 1996.

[58] Mathworks. Matlab model predictive control toolbox. http://www.mathworks.com/access/helpdesk/help/.

[59] I. Matta and A. Bestavros. A load profiling approach to routing guaranteed bandwidth flows. In *IEEE INFOCOM*, volume 3, pages 1014–1021, Mar 1998.

[60] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proc. Int'l Conference on VLSI Design*, pages 693–696, 2004.

[61] H. min Kyung, G. ho Park, J. W. Kwak, W. Jeong, T.-J. Kim, and S.-B. Park. Performance monitor unit design for an axi-based multi-core soc platform. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1565–1572, New York, NY, USA, 2007. ACM.

[62] A. Moonen, R. van den Berg, M. Bekooij, H. Bhullar, and J. van Meerbergen. A multi-core architecture for in-car digital entertainment. In *GSPx*, 2005.

[63] R. B. Mouhoub and O. Hammami. Noc monitoring hardware support for fast noc design space exploration and potential noc partial dynamic reconfiguration. In *IES*, 2006.

[64] S. Murali, M. Coenen, A. Rădulescu, K. Goossens, and G. De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *Proc. Design Automation Conference. Asia and South Pacific (ASPDAC)*, pages 146–151, Jan. 2006.

[65] S. Murali, M. Coenen, A. Rădulescu, K. Goossens, and G. De Micheli. A methodology for mapping multiple use-cases on to networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 118–123, Mar. 2006.

[66] S. Murali and G. De Micheli. SUNMAP: A tool for automatic topology selection and generation for NOCs. In *Proc. Design Automation Conference (DAC)*, June 2003.

[67] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2004.

[68] S. Murali and G. De Micheli. An application-specific design methodology for STbus crossbar generation. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2005.

[69] V. Nollet, T. Marescaux, and D. Verkest. Operating-system controlled network on chip. In *Proc. Design Automation Conference (DAC)*, pages 256–259, June 2005.

[70] OCP International Partnership. Open core protocol specification, 2001.

[71] M. Pastrnak, P. H. de With, C. Ciordas, J. van Meerbergen, and K. Goossens. Mixed adaptation and fixed-reservation QoS for improving picture quality and resource usage of multimedia (noc) chips. In *Proc. Int'l Symposium on Consumer Electronics (ISCE)*, pages 1–6, June 2006.

[72] M. Pastrnak, P. Poplavko, P. H. de With, and J. van Meerbergen. Novel qos model for mapping of mpeg-4 coding onto mp-noc. In *Proc. Int'l Symposium on Consumer Electronics (ISCE)*, pages 93–98, 2005.

[73] K. Peterson and Y. Savaria. Assertion-based on-line verification and debug environment for complex hardware systems. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*, pages 685–688, 2004.

[74] Philips. *Nexperia PNX8550 Home Entertainment Engine*, Dec. 2003.

[75] Philips Semiconductors. *Device Transaction Level (DTL) Protocol Specification. Version 2.2*, July 2002.

[76] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. pages 63–72, 2003.

[77] J. Rossiter. Model based predictive control, a practical approach. CRC Press, 2002.

[78] A. Rădulescu, J. Dielissen, S. González Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 24(1):4–17, Jan. 2005.

[79] A. Rădulescu and K. Goossens. Communication services for networks on chip. In S. S. Bhattacharyya, E. F. Deprettere, and J. Teich, editors, *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*, pages 193–213. Marcel Dekker, 2004.

[80] W. Savage, J. Chilton, and R. Camposano. Ip reuse in the system on a chip era. In *Proceeding of ISSS*, pages 2–7, 2000.

[81] Semiconductor Industry Association. *The International Technology Roadmap for Semiconductors, Design.* 2005.

[82] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proc. Design Automation Conference (DAC)*, pages 667–672, June 2001.

[83] G. J. M. Smit, A. B. J. Kokkeler, P. T.Wolkotte, P. K. F. Holzenspies, M. D. van de Burgwal, and P. Heysters. The chameleon architecture for streaming dsp applications. *EURASIP Journal on Embedded Systems*, 2007.

[84] B. Sprunt. Pentium 4 performance-monitoring features. *IEEE Micro*, 22:72–82, 2002.

[85] ST - TA305 TCEHNICAL ARTICLE . Nomadik - open multimedia platform for next generation mobile devices.

[86] N. Stollon and R. Leatherman. Integrating on chip debug instrumentation and eda verification tools. In *DesignCon East*, 2005.

[87] Synopsys. http://www.synopsys.com/products/designware/designware.html, 2007.

[88] S. Tang and Q. Xu. A multi-core debug platform for noc-based systems. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 870–875, San Jose, CA, USA, 2007. EDA Consortium.

[89] S. Tang and Q. Xu. A debug probe for concurrently debugging multiple embedded cores and inter-core transactions in noc-based systems. In *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pages 416–421, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[90] Texas Instruments. Omap platform - www.ti.com/omap3.

[91] C.-K. Tham, Y. Jiang, and C.-C. Ko. Monitoring qos distribution in multimedia networks. *International Journal of Network Management*, 10(2):75–90, 2000.

[92] The Nexus 5001 Forum. *Nexus Standard*.

[93] R. van den Berg and H. Bhullar. Next generation philips digital car radios, based on a sea-of-dsp concept. In *IEEE ISPC GSPx*, 2004.

[94] J. W. van den Brand, C. Ciordas, T. Basten, and K. Goossens. Congestion-controlled best-effort communication for networks-on-chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 948–953, Apr. 2007.

[95] J. W. M. van den Brand. Run-time networks on chip performance monitoring. Master's thesis, Technical University Eindhoven, Aug. 2005.

[96] G. J. van Rootselaar and B. Vermeulen. Silicon debug: Scan chains alone are not enough. In *Int'l Test Conference (ITC)*, pages 892–902, 1999.

[97] B. Vermeulen. Functional debug techniques for embedded systems. *IEEE Des. Test*, 25(3):208–215, 2008.

[98] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas. Bringing communication networks on chip: Test and verification implications. *IEEE Communications Magazine*, 41(9):74–81, Sept. 2003.

[99] B. Vermeulen, K. Goossens, R. van Steeden, and M. Bennebroek. Communication-centric SOC debug using transactions. In *Proc. European Test Symposium (ETS)*, May 2007.

[100] B. Vermeulen, S. Oostdijk, and F. Bouwman. Test and debug strategy of the pnx8525 nexperia digital video platform system chip. In *Int'l Test Conference (ITC)*, pages 121–130, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[101] B. Vermeulen and G. van Rootselaar. Silicon debug of a co-processor array for video applications. In *Proc. Workshop on High-Level Design Validation and Test (HLDVT)*, pages 47–52, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[102] D. Wingard. Socket-based design using decoupled interconnects. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 15, pages 367–396. Kluwer, 2004.

About the author

Calin Ciordas was born on 4th of April 1976 in Oradea, Romania. In 2000 he obtained the Master Degree in Computer Science from the Technical University of Cluj Napoca, Romania. After a short software development experience in Romania, he joined the Information and Communication Technology TWAIO programme from the Stan Ackermans Institute of the Eindhoven University of Technology. He obtained his MTD degree (nowadays PDEng) in 2003 and continued as a research assistant in the Electronic Systems (ICS/ES) group of the Eindhoven University of Technology. He worked on monitoring networks on chip and collaborated with Philips Research in a network on chip project. Since March 2008 Calin is a researcher at Irdeto.

List of Publications

Journal Papers

1. *"A Monitoring Service for Networks-on-Chip"*;
   Calin Ciordas, Twan Basten, Andrei Radulescu, Kees Goossens, and Jef van Meerbergen; ACM Transactions on Design Automation of Electronic Systems, 10(4):702-723, October 2005. Special Issue on Validation of Large Systems. [18]

2. *"A Monitoring-Aware Network on Chip Design Flow"*;
   Calin Ciordas, Andreas Hansson, Kees Goossens, and Twan Basten; Elsevier Journal of Systems Architecture, 54(3-4):397-410, March-April 2008. Special Issue with selected best papers of DSD2006). [24]

3. *"Bringing Communication Networks On Chip: Test and Verification Implications"*;
   Bart Vermeulen, John Dielissen, Kees Goossens, and ***Calin Ciordas***; IEEE Communications Magazine, 41(9):74-81, September, 2003. [98]

Conference Papers

4. *"A Monitoring-Aware Network on Chip Design Flow"*;
   Calin Ciordas, Andreas Hansson, Kees Goossens, and Twan Basten; In Proc. 9th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools (DSD), pages 97-104, August 2006. IEEE, 2006. [23]

5. *"NoC Monitoring: Impact on the Design Flow"*;
   Calin Ciordas, Kees Goossens, Andrei Radulescu, Twan Basten; In Proc. Int'l Symposium on Circuits and Systems (ISCAS), pages 1981-1984, May 2006. IEEE, 2006. [22]

6. *"Transaction Monitoring in Networks on Chip: The On-Chip Run-Time Perspective"*;

Calin Ciordas, Kees Goossens, Twan Basten, Andrei Radulescu, and Andre Boon; In Proc. IEEE Symposium on Industrial Embedded Systems (IES), October 2006. IEEE, 2006. (Best paper award) [19]

7. *"An Event-based Network-on-Chip Monitoring Service"*;
Calin Ciordas, Twan Basten, Andrei Radulescu, Kees Goossens, and Jef van Meerbergen; International High Level Design Validation and Test Workshop (HLDVT), pages 149-154, November 2004. IEEE, 2004. [17]

8. *"Combined Reservation and Adaptation QoS for Improving Picture Quality and resource usage of multimedia (NoC) chips"*;
Milan Pastrnak, Peter de With, ***Calin Ciordas***, Jef van Meerbergen, and Kees Goossens; In Proc. Int'l Symposium on Consumer Electronics (ISCE), September 2006. IEEE, 2006. [71]

9. *"Congestion-controlled best-effort communication for networks-on-chip"*;
Jan-Willem van den Brand, ***Calin Ciordas***, Kees Goossens, and Twan Basten; In Proc. Design Automation and Test (DATE), April 2007. IEEE, 2007. [94]

Published Pending Patent Applications (Philips and NXP)

10. (Patent WO2006051471) *Electronic device and method of communication resource allocation*;
Calin Ciordas, Kees Goossens, Andrei Rădulescu
KONINKLIJKE PHILIPS ELECTRONICS N.V., may 2006. [20]

11. (Patent WO2008004185) *Electronic device, system on chip and method for monitoring data traffic*
Kees Goossens, ***Calin Ciordas***, Andrei Rădulescu
NXP BV., january 2008. [39]

12. (Patent WO2008004187) *Electronic device, system on chip and method of monitoring data traffic*
Kees Goossens, ***Calin Ciordas***
NXP BV., january 2008. [38]

13. (Patent WO2008004188) *Electronic device, system on chip and method for monitoring a data flow*
Calin Ciordas, Kees Goossens, Andrei Rădulescu
NXP BV., january 2008. [21]