# Resource virtualization for mapping multiple applications on the same platform

van Hintum, M.M.W.

*Award date:*
2007

Marc van Hintum

# RESOURCE VIRTUALIZATION FOR MAPPING MULTIPLE APPLICATIONS ON THE SAME PLATFORM

# RECOURCE VIRTUALIZATION FOR MAPPING MULTIPLE APPLICATIONS ON THE SAME PLATFORM

MARC VAN HINTUM



MSc

Embedded Systems
Computer Science & Engineering
Technical University Eindhoven

Jan 2007

From an engineering perspective, there are two ways to make something bigger: One is to make it physically bigger, (and human beings spent a lot of time making things physically bigger, working out ways to deliver more power to systems, working out ways to actually build bigger buildings, working out ways to expand territory, working out ways to invade other cultures and take over their territory, etc.) But there's another way to make things bigger, and that's to make things smaller. Because the real size of a system is not how big it actually is, the real size is the ratio between the biggest part of a system and the smallest part of a system. Or really the smallest part of a system that you can actually put to use in doing things.

— Seth Lloyd, Moore's Law and the Ultimate Laptop

Marc van Hintum: *Recource Virtualization for Mapping Multiple Applications on the Same Platform,* MSc, © Jan 2007

SUPERVISORS:
prof. dr. ir. Jef van Meerbergen
ir. Marino Strik

LOCATION:
Eindhoven

TIME FRAME:
Jan 2007

I dedicate this thesis report to my loving grandfather,
who died unexpectedly during the time I was writing
this work.

## ABSTRACT

The increasing complexity of systems-on-chip enabled by technology scaling drives the increase in design effort. Changes are required in system-on-chip development to drive improvements in effectiveness to reduce (non-recurring engineering) cost and improve time-to-market.

To deal with these requirements a platform-based approach is advocated. Intellectual Property blocks are reused to reduce design effort. Furthermore complete platforms are reused to evolve product emphasis once performance requirements have been satisfied in a certain application domain.

However, scaling a design causes a shift in performance bottleneck from computation toward communication. Arbitration to shared resources (i.e. remote memory) can cause serious latency issues as it is a non-scalable bottleneck in the architecture. Furthermore, resource sharing also causes inter-dependencies between jobs that are mapped on the architecture. Integration or evolvement of resources changes all temporal behavior, where complete system verification is required to validate end-to-end behavior.

This thesis contributes to these issues by introducing a hardware platform, based on the Æthereal network-on-chip, that is optimized for streaming applications. Virtualization of the platform enables a compositional mapping of jobs on the architecture such that all inter-dependencies between these jobs are completely removed. By removing all inter-dependances the strictest form of compositionality is obtained. All additional costs for the strictest form of compositionality are accepted for this thesis. The resulting system allows for individual verification of jobs instead of extensive simulation of all possible mappings on the architecture.

# ACKNOWLEDGMENTS

I would like to express my gratitude for all the help and support I have received throughout this graduation project. In special I would like to thank my supervisors Jef van Meerbergen and Marino Strik for giving me the opportunity to work at NXP Semiconductors and for giving me advise and suggestions for improvement during the period of my thesis.

I would like to thank Andreas Hansson for his constant support and encouragement. For giving me advise, suggestions, critique and directions to successfully fulfill my thesis assignment.

Furthermore I would like to thank all members of the Hydra and Æthereal project within NXP research, especially Kees Goossens, Marco Bekooij and Martijn Coenen. I would like to thank the members of the PreMaDona project within Eindhoven University of Technology for the opportunity to join them in many interesting meetings.

# CONTENTS

# LIST OF FIGURES



# LIST OF TABLES



# ACRONYMS

AHB Advanced High-performance Bus

AMBA Advanced Microcontroller Bus Architecture

AXI Advanced eXtensible Interface

BE Best-Effort

CMOS Complimentary Metal Oxide Semiconductor

CPU Central Processing Unit

DDF Dynamic Data Flow

DMA Direct Memory Access

DSM Deep Sub-Micron

DSP Digital Signal Processor

DTL Device Transaction Level

FFT Fast Fourier Transform

FIFO First In First Out

FSM Finite State Machine

GALS Globally asynchronous and locally synchronous

GS Guaranteed Service

IP Intellectual Property

IR drop Voltage drop in the supply lines

ISO-OSI International Standard Organization's Open System Interconnect

ITRS International Technology Roadmap for Semiconductors

MMIO Memory-Mapped Input/ Output

MPSoC Multi-Processor System on Chip

NI Network Interface

NoC Network-on-Chip

NRE Non-Recurring Engineering

NuMa Non-Uniform Memory Access

OCP Open Core Protocol

PMAN Circuit-based interconnect for video streaming

QoS Quality of Service

RC delay Delay in interconnect lines

RISC Reduced Instruction Set Computer

ROM Read Only Memory

RTL Register Transfer-Level

SDRAM Synchronous Dynamic Random Access Memory

SoC System-on-Chip

SRAM Static Random Access Memory

TCL Tool Command Language

TDMA Time-Division Multiple Access

TDMC Time-Division Multiplexed Circuit-Switching

VLIW Very Long Instruction Word

XML eXtensible Markup Language

*"Cost (of design) is the greatest threat to continuation, of the semiconductor roadmap"*

— *ITRS [10]*

# 1 INTRODUCTION

To meet the computational requirements of modern day applications in domains as multimedia and automotive, the processing power of *systems-on-chips* (SoC) has to increase. Single general-purpose processors are typically not efficient enough to suffice these high computational demands, and would lead to unacceptable power dissipation. In these cases the use of scalable and power efficient *multiprocessor system-on-chip* (MPSoC) platforms is advocated.

This is supported by the so-called *design productivity gap*, which states that the expected scaling towards smaller dimensions, as predicted by *Moore's law*, will continue for the foreseeable future but our ability of designing embedded systems cannot keep up with this technology scaling. In other words, *deep sub-micron* (DSM) technology enables designers to build more *complex systems*[1], but the design effort is not a linear function of the size of the design. *Platform-based* SoC design promises to boost productivity by minimizing the effort to add components to a given design.

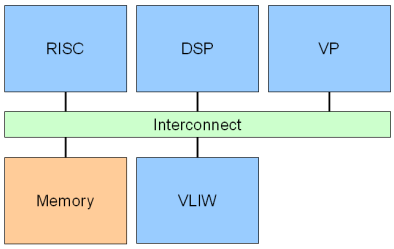


Figure 1. System-on-Chip

The importance in this is the fact that the manufacturing *non-recurring engineering* (NRE) cost of designing these complex systems rapidly becomes the greatest threat to continuation of the semiconductor roadmap, as stated by the *International Technology Roadmap for Semiconductors* (ITRS) [10]. The designer is faced with extreme challenges that arise with the additional complexity of SoC design caused by the consumers demand for increased functionality and lower cost. These challenges diverse from DSM problems (i.e. timing, power consumption and yield)

1 Referring to the exponentially increasing transistor counts enabled by smaller feature sizes making it possible to integrate complete systems with many hardware resources including not only storage and processing functions but also peripherals and interfaces on a single chip

via architectural problems and application software all the way to verification.

Rapid technology change shortens product life cycles and makes time-to-market a critical issue for semiconductor customers [10], while NRE costs are in the order of millions of dollars (table 1). To avoid an explosion of the design costs reuse is advocated. The abstraction level is raised from standard cells to complete *Intellectual Property* (IP) blocks i.e. CPU, DSP, memory and interconnect. Platform-based design shortens the SoC design process by making use of reusable groups of cores to form a complete hardware platform. This enables integration of reusable and configurable IP blocks unspecific for a certain application domain.

IP reuse is only one of the challenges accompanied with the attempted closure of the design productivity gap. The ITRS lists several more of these challenges, they include [10]:

- *Reuse* — support for hierarchical design, heterogeneous SoC integration (modeling, simulation, verification, test of component blocks) especially for analog/mixed-signal
- *Verification and test* — specification capture, design for verifiability, verification reuse for heterogeneous SoC, system-level and software verification, verification of analog/mixed-signal and novel devices, self-test, intelligent noise/delay fault testing, tester timing limits, test reuse
- *Cost-driven design optimization* — manufacturing cost modeling and analysis, quality metrics, co-optimization at die-package-system levels, optimization with respect to multiple system objectives such as fault tolerance, testability, etc.
- *Embedded software design* — predictable platform-based electronic system design methodologies, codesign with hardware and for networked system environments, software verification/analysis
- *Reliable implementation platforms* — predictable chip implementation onto multiple circuit fabrics, higher-level handoff to implementation and
- *Design process management* — design team size and geographic distribution, data management, collaborative design support, "design through system" supply chain management, metrics and continuous process improvement

Most products fit in a larger family of products. The members of such a product domain share a lot of functionality and features. It is attractive to share implementations, designs et cetera between those members to increase the efficiency of the entire company. Once performance requirements have been satisfied in a particular application domain, in order to remain viable and competitive, the product emphasis must evolve. An example is the mobile phone, were bluetooth, a digital camera and MP3 encoder are integrated in the more advance products within the family, while the basic functionality is the same. Next to IP

reuse complete hardware platforms are reused and migrated across a family of products.

In practice many difficulties pop-up when product developments become coupled. Integration of IP blocks to enhance or extend functionality creates new dependencies as shared resources such as interconnect and memory often do not scale with product evolvement. Interference at these resources results in major latency issues when scaling the design. In combination with the increasing cost of wires due to the silicon process evolution the bottleneck in system performance shifts from computation to communication. Consequently, the contention at shared resources puts high demands on arbitration to ensure system performance requirements. When a certain requirement is established in a application domain, integration or evolvement of resources change data traffic patterns and hence it must be re-ensured that performance requirements are met.

Many applications, such as for audio and video compression, have hard real-time timing requirements (i.e. latency and throughput) to guarantee a certain quality. If these requirements are not met the quality of the audio or video may not suffice to consumer standards. Arbitration can cause timing requirements of one job to be met at the cost of missing the requirements of other jobs. Consumers expect a predictable *quality-of-service* (QoS) of systems. Therefore, timing requirements must be verified before a product is made available on the market.

| Feature dimension (transistor count) | Cost for typical IC design | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0.18µm (30M) | | 0.13µm (47M) | | 90nm (54M) | | 65nm* (60M) | |
| | $M | % | $M | % | $M | % | $M | % |
| Generation of spec and spec to RTL | 0.283 | 10.0 | 0.428 | 9.6 | 1.143 | 12.6 | 3.023 | 16.3 |
| RTL to netlist (verification) | 1.621 | 57.1 | 2.419 | 54.0 | 4.178 | 46.1 | 7.187 | 38.7 |
| Netlist to GDSII | 0.639 | 22.5 | 1.132 | 25.3 | 2.535 | 28.0 | 5.534 | 29.8 |
| Validation | 0.295 | 10.4 | 0.498 | 11.1 | 1.207 | 13.3 | 2.807 | 15.1 |
| TOTAL | 2.838 | 100.0 | 4.477 | 100.0 | 9.063 | 100.0 | 18.551 | 100.0 |

Note:
* Estimate.

Source:**GLOBAL SYSTEM IC (ASSP/ASIC) SERVICE MANAGEMENT REPORT, IBS INC. July 05

Table 1. Design cost

Meeting design constraints can require many design iterations (*design closure*). This becomes even more true with the increasing flexibility of MPSoCs, where the enormous amount of possible combinations of applications and there non-deterministic behavior make it very difficult to valide the timing requirements. To come back at the example of the mobile phone, an MPSoC-based mobile phone may execute an MP3 decoder to produce music, while the user also writes a text message concurrently to downloading a new ring tone in the background. The user should not experience significant quality drops or delays when

activating or de-activating applications.

Interference between integrated IP blocks causes unpredictable behavior, which makes it difficult if not impossible to verify that timing requirements are met at design-time. Simulation can only be used to demonstrate that timing requirements are met for a particular set of input stimuli and therefore does not guarantee correct behavior. Moreover, simulation is time-consuming. Predictable platform-based system design is advocated to reduce design cost and time-to-market by enabling system analysis at design-time to ensure QoS.

Note that predictability is not only endangered by interference at shared resources, but by all uncertainties introduced in the design process (i.e. unpredictable IP behavior). The essence of QoS is therefore the offering of a predictable system behavior to the consumer. In order to predict the behavior of a complete system the behavior of all resources must be predictable and hence every resource must provide QoS.

## 1.1 PROBLEM DESCRIPTION

Increased functionality and heterogeneity of MPSoC enables efficient parallel processing of jobs on a system. This increases computational power and minimizes power dissipation but on the other hand evokes a new set of problems. Dynamic market behavior results in short product life cycles and consequently high NRE design costs. These NRE design costs largely result from verification effort (see table 1). Uncertainties in resource behavior make it difficult to guarantee that a certain QoS is met. This motivates the emphasis on the analysis and prediction of the behavior of an application instead of the need for extensive simulation. Therefore, it is essential to enable a predictable mapping of the application onto the architecture.

Predictability (3.1.1) is a valuable asset that deals with uncertainties that are introduced by resources. This can be obtained by either removing or bounding these uncertainties. Predictability is a non-trivial issue and motivation of many research groups. In order to close the design productivity gap the emphases should be on predictable platform-based system design methodologies [10].

One of the causes of unpredictable behavior is due to interference at the shared resources (computation, communication and storage). Interference between resources make system behavior unpredictable. Jobs become dependant on each others temporal behavior as they share resources, and put high demand on the arbitration of contention points. A more relaxed but valid requirement that can be established is that once the temporal behavior of a job mapped on a system is predicted or simulated, that behavior is still valid for any composition of other jobs in the system. This deals with the compositionality of an architecture.

Compositionality (3.2.1) makes a design more reliable as it guarantees a certain amount of shared resources to be available for each job. Compositionality relates to the uncertainties in the availability of shared resource and thus is a necessary prerequisite for predictability, as is shown in section 3.2.2. This allows the designer to do individual analysis or simulation of jobs instead of exhaustive simulation of all

possible mappings of the system.

## 1.2 CONTRIBUTION

This report tries to contribute to the closure of the design productivity gap by introducing the first steps towards predictable platform-based design. This report presents a compositional platform-based design methodology. First latency issues are addressed to minimize the effect of performance bottlenecks caused by resource sharing. Resource sharing creates dependances between jobs as they content for the same resource. Resource contention points put high demands on arbitration to guarantee performance requirements, which requires many design iterations. Therefore secondly compositionality is introduced, which decreases verification effort by enabling independent subsystem verification. The aim for this thesis is to prove the concept of compositionality in its strictest form, which is that the behavior of a job is not effected by other jobs on a cycle-true level. For this thesis all the costs for obtaining the strictest form of compositionality are excepted.

Compositionality is achieved by means of *virtualization* (3.3.1). This essentially means that resources are virtually divided in such a way that the temporal behavior of one job does not effect the temporal behavior of another job. This gives a job the illusion that it is running on its own virtual platform. This report thus introduces resource virtualization for mapping multiple applications on the same MPSoC based platform.

## 1.3 ORGANIZATION

The rest of the thesis is organized as follows. Chapter 2 looks into the trends in SoC design and addresses latency issues caused by the scaling of designs enabled by DSM technologies. Based on these findings it introduces an architecture that is optimized for streaming applications. In Chapter 3 predictability and compositionality are defined and virtualization is introduced as a new design methodology. In chapter 4 the concept of virtualization is applied on the proposed architecture. Chapter 5 specifies the IP blocks that form the actual hardware platform. Chapter 6 discusses the implementation detail needed to obtain the hardware platform. Experimental results and analysis of the results are shown in chapter 7. Chapter 8 summarizes the conclusions of this thesis and chapter 9 ends with directions for future work.

*It has become appallingly obvious that our technology has exceeded our humanity*

*— Albert Einstein*

# 2 FUTURE TRENDS TOWARDS COMPLEX SYSTEMS

As predicted in 1965 by Gordon E. Moore , co-founder of Intel, the complexity of integrated circuits roughly doubles every 2 years. The original statement is the following:

> The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer.
>
> *Gordon E. Moore - "Cramming more components onto integrated circuits", Electronics Magazine 19 April 1965*

In the foreseeable future this trend is expected to continue (figure 2). This enables the design of complex SoCs, though as stated in the introduction the downside is that the design effort for such complex systems rapidly becomes the bottleneck for continuation of the semiconductor roadmap.

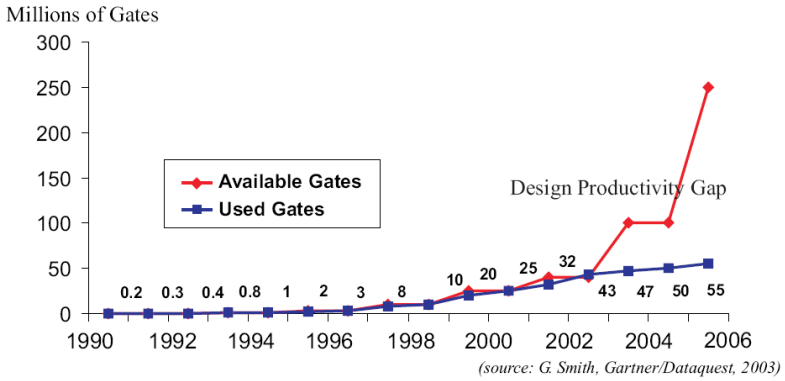


Figure 2. Design productivity gap [18]

Challenges have to be met to overcome these issues resulting in the increase of complexity. This leads to requirements for new methods

and techniques to enhance productivity, controllability and quality of hardware components. To get a better insight in these issues this chapter focuses on the root causes for the increasing complexity of SoC design. This means approaching the problem from a deep sub-micron (DSM) perspective. Furthermore, the trends in resolving these issues are discussed.

## 2.1 DEEP SUB-MICRON PROBLEMS

This section is about issues related to deep sub-micron problems. Deep sub-micron technology relates to the decrease in feature sizes towards only fractions of a micron, allowing for billions of transistors on a single die, possibly running at gigahertz frequencies. Improved microprocessor performance results largely from this technology scaling, which lets designers increase the level of integration at higher clock frequencies [26]. Though deep sub-micron technologies enhance possibilities in semiconductor integration, its complexity hampers the verification and test process and face the designer with several major technology challenges. These consequences become even greater when scaling the design towards higher dimensions.

In this report the DSM problems are divided into three groups, namely *timing, power* and *yield*. The following sections show how these DSM problems propagate through several architectural levels all the way to the software applications that are ran on such a systems. It introduces a new design methodology that raises the abstraction level from these DSM problems to complete hardware blocks. Furthermore, latency issues are addressed to minimize the effect of shared resource bottlenecks.

### 2.1.1 *Timing*

In the early days of SoC design the main challenge was to optimize a design for speed. This issue was mainly addressed by advantages in IP performance. However, with the increase of the number of IP blocks on a chip the length of the wires connecting them (interconnect) also has increased. With the silicon process evolution the RC-delay of wires becomes much larger than the gate delay and thus the dominant factor in chip performance. Consequently, timing has become one of the most important DSM challenges and the main driver for performance limitation.

In [33] there are listed several consequences of this change in the main driver for performance limitation. First the placement and routing of the wires becomes important and hence many layout iterations are needed. This is the problem of *timing closure*. Secondly, RC delays in supply lines leads to voltage loss or so-called IR-drop associated with current peaks. Third, similar effects happen in clock lines, which causes skew problems. The arising wire delays to connect IPs now become a dominant factor. To minimize this effect, global wires should be decoupled from the local wires of an IP, i.e. clock lines become too long and consequently too slow for a SoC to be synchronized with only one clock. The future trend is now towards *globally asynchronous and locally*

*synchronous* (GALS) [25].

As the bottleneck changes from computation to communication, communication becomes a central concern. This causes a shift from a computation to a communication-centric approach [5, 7]. Reuse of communication structures is advocated which supports the use of *Network-On-Chip* (NoC)[5, 6, 18, 33], since on-chip networks are scalable, flexible and reusable communication structures that decouple local and global wires so that it allows for GALS.

### 2.1.2 *Power*

The second group of problems are power related. Power dissipation is a major concern especially for mobile and wireless (nomadic) applications, where dissipation constrains can be as low as 1 Watt. This has formed the main reason to drop the supply voltage. However, for CMOS technology the drop in supply voltage typically has a negative effect on the performance. This effect on performance can be limited somewhat by scaling the threshold voltage but this again results in increased static power dissipation.

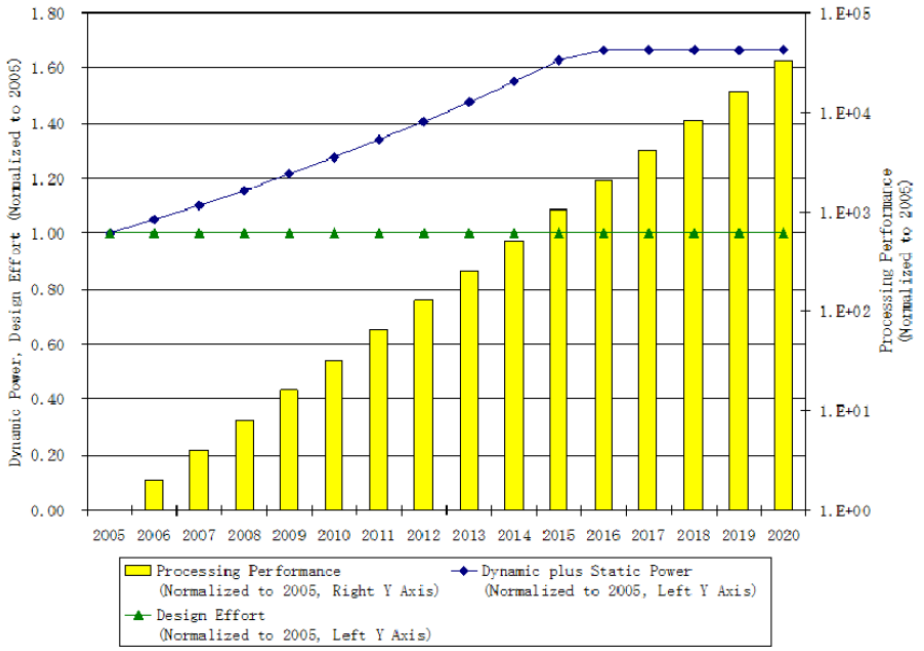


Figure 3. Trends for power efficient SoC [11]

In [11] several trends for power efficient SoC drivers are given. They are partly based on the model created by the Japan Semiconductor Technology Roadmap Design Working Group and are shown in figure 3. The application domain is "*Mobile Consumer Platforms*". This is a domain that is rapidly evolving and the application domain specific to this thesis. The ITRS lists several aspect of the model including [11]:

- Its typical application area is electronic equipment categorized as "Mobile Consumer Platforms" because this application area will make rapid progress in the foreseeable future across semiconductor technology generations.
- From the typical requirements of this type of SoC ("Mobile Consumer Platforms") an explosive increase of processing power is required under some upper bound for battery life.
- As a result, the requirement for processing power will be 1000x in the next ten years, while the requirement for dynamic power consumption will not change noticeably.
- The life cycle of "Mobile Consumer Platform" products is short, and will stay short in the future. Therefore, the design effort cannot be increased - it needs to stay at the current level for the foreseeable future.

Note that the different requirements coincide. As shown in the figure there is a need for an explosive increase in processing power while power consumption and design effort stay near constant. With the already mentioned design productivity gap (see figure 2) and timing closure this seems as an impossible task. Solving this asks for a completely new design methodology. The future trend is now towards platform-based MPSoC design. Multiprocessors offer a cost-efficient high-performance platform to meet the throughput and latency requirements of this application domain by virtue of parallel processing. Platform based design simplifies the SoC design process by making use of reusable groups of cores to form a complete hardware platform.

### 2.1.3 *Yield*

Finally there is the problem of yield. Currently inspection systems are expected to detect defects of sizes scaling down in the same way or even faster as feature sizes required by technology generations. Noise and variability have negative impact on yield and the problem of finding yield relevant defects becomes a major issue. In [12] it is stated that the signal-to-noise ratio for defect inspection tools was identified by the community as the most important challenge for yield enhancement. Noise can come from sources as the supply, the substrate, from inductive effects, capacitive coupling between neighboring wires (crosstalk), from alpha particles and other forms of radiation that injects charges [33]. *Variability* [27] is a major concern for predictability. It relates to the fact that identical transistors can show different electrical behavior. This causes the need for reliable systems to be designed using unreliable components. Abstraction to a higher granularity is one of the methods for solving this issue, making the design error tolerant. Though the importance of this subject, it is not within the scope of the thesis and is therefore not discussed in the continuation of this report.

## 2.2 MEMORY

With the evolvement of MPSoC and hence the increase in processing performance one of the pressing obstacles is the disparity between memory and processor speed. This gap is referred to as the *memory gap*. On-chip memory blocks are reasonably fast but with the increase in capacity this advantage is quickly disappearing, where also the amount of chip area it occupies is increasing significantly. Off-chip memory can have virtually no capacity limitations, but is slow and needs a high amount of interface pins and therefore is often limited to one. Although a remote (shared) memory provides a large amount of available memory to a processor, communication latency is often an issue. This can mostly be hidden by caches, but still the combination of multiple processors and a scarcity of - relative slow - memory blocks leads to resource sharing and hence high demands on communication and memory resource arbitration. Shared memory becomes a non-scalable bottleneck in the architecture.

To give an example, if there is a 3% cache miss rate and every cache miss resolves into an extra 20 clock cycles penalty then 60% of the time the processor would stall. When fetching instructions and data from shared memory this poses several questions:

- Is it possible to reduce[1] the number of low-latency requests to external SDRAM? *Relax vs. remove*
- Does the problem come from the architecture or is it a fundamental requirement from the application domain? *Architecture vs. application*

The designer of a specific architecture should be aware of this problem and the posed questions. Interference at shared resources such as memory should be minimized to decrease the number of stall cycles due to arbitration. Trends are towards *Non-Uniform Memory Access* (NuMa) architectures [9] where embedded local memories are introduced that are kept close the the processor units. Distributing memory decreases the amount of sharing and furthermore the memory access time to on-chip local memory is smaller than the access time to a remote off-chip memory, thus decreasing latency and the number of accesses to the remote memory.

The memory issue appears throughout many of the application domains and is considered an architectural problem. In the following sections the requirements of the application domain are investigated to identify whether the latency requirements can be relaxed. The consequences of the architecture are discussed and an architectural template is introduced to address this issue.

## 2.3 APPLICATION DOMAIN

In this chapter the application domain "Mobile Consumer Platforms" is introduced. This application domain is rapidly evolving and dominant in many technology offices [11]. The reason not to choose for a general purpose approach is that this statement may be too strong. By narrowing the scope to a more select application domain the architecture can

1 Relax the timing requirements of low-latency requests or remove low-latency requests

be optimized for this, as is shown later on. However, making the scope too small would again lead to high - NRE - design and mask costs.

Applications in this domain often have high requirements on power consumption. As the application domain is focussed on nomadic applications, the system is often battery driven. To increase battery lifetime, power consumption should be as low as possible. Furthermore consumer demands raise computational requirements to enable applications with high demands on processing power.

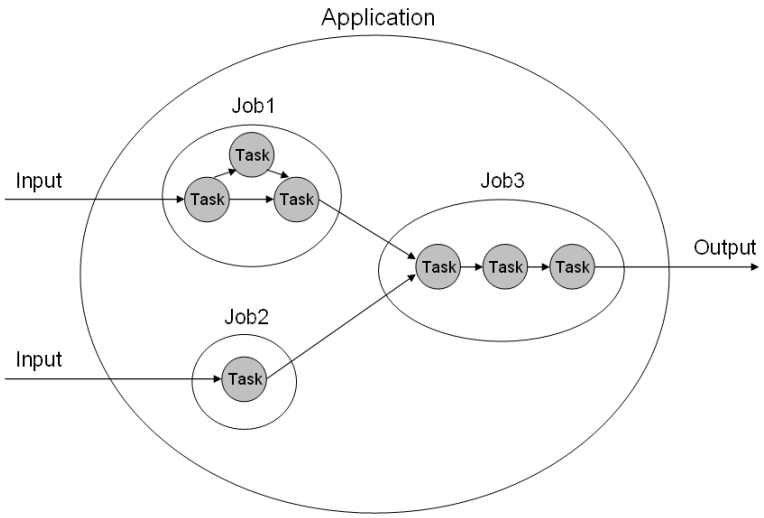


Figure 4. Application graph

To execute these applications they are mapped onto an architecture. In order to more accurately specify applications, the description of applications may require a somewhat more extended view. This description is used throughout the report. An application that is ran on a system can consist of several jobs. These jobs, in turn, can consist of several tasks possibly executed in parallel (see figure 4), which are mapped onto a certain *Processing Element* (PE). It is up to the designer to make the most optimal mapping of the application onto the architecture.

The support of a complete product family such as for "Mobile Consumer Platforms" requires flexible SoCs, which advocates the use of programmable multiprocessor systems. The advantage of the flexibility introduced by MPSoC is that multiple applications can be mapped on the same platform. Single general-purpose processors are typically not power efficient enough to suffice the high computational demand in this application domain and would lead to unacceptable power dissipation.

Often different tasks can have different requirements (i.e. computational and memory requirements). This advocates the use of different PEs, and thus a *heterogenous MPSoC*. However, the designer has to make the tradeoff between the amount of flexibility and performance of such a SoC. The use of dedicated hardware increases performance

but decreases the flexibility of the SoC. As explained this tradeoff is also dependent on the power dissipation limit, which for the "Mobile Consumer Platforms" is one of the driving constraints.

### 2.3.1 *Streaming applications*

So far the only assumption that is made is that the application domain is "Mobile Consumer Platforms". As briefly discussed efficiency can be gained when a system has a specific behavior characteristic for the application domain. Note that optimization of a system for a specific behavior does not imply that the system does not work anymore for other applications. When analyzing system behavior two sorts of data streams are identified, namely *low-latency* and *latency-tolerant* data streams.

Low-latency data streams are data streams for which late arrival of the data will result in many processor stall cycles. Data/ instruction misses have typically low-latency requirements because access latency of remote memory reads causes processor stall cycles, thereby making latency to SDRAM critical. As mentioned in section 2.2 latency can be somewhat hidden by caches, but when a next instruction is not available in the cache (read miss) the data still has to be fetched from a higher level memory (i.e. remote memory) where access times may exceed tolerable timing requirements.

Latency-tolerant data streams are less dependant on latency, e.g. the processor does not stall if the latency is somewhat bigger. Sometimes techniques can be applied where low-latency data streams can be made latency-tolerant. Examples are write streams where latency is hidden by using buffers; as long as the consumer of data has enough data left in a buffer the producer can be slowed down.

A characteristic behavior of systems within the application domain "Mobile Consumer Platforms" is the use of data streams. Applications in this domain are often referred to as *streaming* applications. In [33] a more specific description of streaming applications is given.

> The main characteristic is a repetition of the same function over and over again on new input data. The function can show data dependent behavior. Streaming applications are conveniently represented as graphs, e.g. dataflow graphs. An important characteristic is an explicit separation of variables in internal variables, which are local to the computation in the node and external variables, which are communicated over the edges. Data abstraction in the form of tokens is possible. For example for video applications tokens can be pixels, blocks, lines, stripes, frames, etc... Tokens are produced and consumed in fifo order. Random access is still possible within a token and also to the local state. Dynamic applications and bursty behavior can be modeled using dynamic dataflow (DDF). As a result many dynamic applications can be modeled, e.g. coding, motion compensation, graphics, etc... It is not streaming anymore if random access is needed to a dataset, which is too large to fit in local memory and thus must be stored in SDRAM. An example is H264 where motion vectors must be detected in 5 frames.

Streaming data allows for optimization due to the explicit separation of variables in internal variables. Streaming data can thereby be produced independent of the consumption of the data, where data can be buffered if the production rate is higher than the consumption rate. The producer thereby does not stall if the consumer is not able to process the data immediately. If the buffers are kept close to the consuming processor, i.e. in a local on-chip memory instead of a off-chip shared memory, the consumer has low-latency access to that data. In this case latency is hidden by the buffers, making the data stream latency-tolerant.

## 2.4 ARCHITECTURE

In the previous sections several trends and challenges are discussed, summarized they include: deep sub-micron effects, computation versus communication, heterogenous multiprocessor SoC, global asynchrony, design productivity gap and the memory gap. These trends and challenges ask for a completely new design methodology.

The design of complex systems requires a lot of effort and specific knowledge. This not only implies complying to functional requirements but also means that the non-functional requirements, i.e. timing, power and area, have to be met. In order to minimize effort for designing these complex systems one needs to abstract from the DSM problems and emphasize on a higher level of abstraction were complete IP blocks are reused. Hence, the level of abstraction should change from standard cells to complete IP blocks such as CPUs, DSPs and memories. Reusability has been recognized as a basic principle for enhancing productivity and quality of engineering products [10].

With the evolvement of the application domain "Mobile Consumer Platforms" next to timing also power dissipation becomes an important non-functional requirement. We have seen that the lowering of the supply voltage does not suffice to solve this problem. As discussed in the previous section MPSoCs become an interesting alternative to single general-purpose processors.

As the introduction of MPSoC is accompanied with the shift towards a communication-centric design approach, the reuse of standardized communication structures is advocated. Reuse not only addresses complete hardware blocks, but also integration and verification of composed systems. Standardized communication structures and interfaces support reuse, since IPs with standardized interfaces can be easily integrated and exchanged and also the communication structure itself is reused. NoC is a promising solution offering interfaces to integrate IPs and is suitable for GALS to minimize the effect of long wires.

In [15] a MPSoC template is proposed, which is adopted for this report. The template is shown in figure 5. The architecture corresponds to the technology trends mentioned in this report. The platform-based design abstracts from most of the DSM problems which minimizes design effort, where the on-chip network enables decoupling of computation and communication and allows for heterogeneous multiprocessor and GALS. This is further explained in chapter 4.

The architecture has a large remote - shared - memory space capable

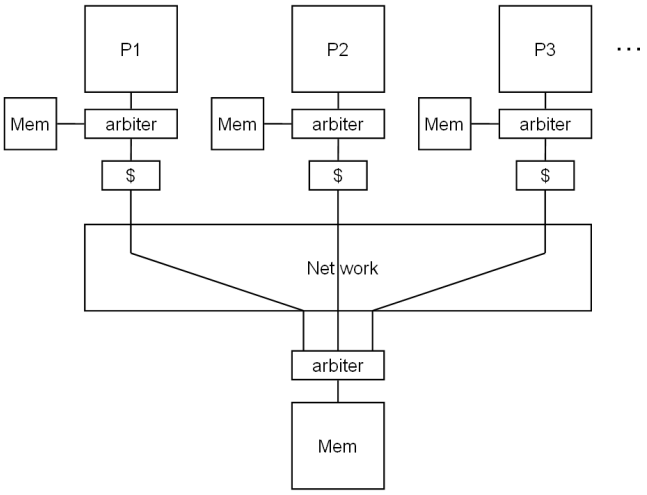


Figure 5. Multiprocessor design template

of storing all data, like instructions and local state. To minimize latency due to interference caused by multiple processors accessing the shared memory caches are introduced. In [16] a consistency model is proposed, which ensures this system to be software cache coherent and memory consistent in order to solve inter-task interference in the cache.

Furthermore, the architecture is a NuMa architecture where on-chip local memory is available, which enables remote memory access optimization and provides shorter access times by bypassing the global interconnect. This is explained in the previous section and is applied in this design. The local memory can be used for buffers, buffer administration and possible some data and state variables but might be too small to store all this information.

### 2.4.1 *Low-latency*

In section 2.2 two questions are stated, namely whether it is possible to reduce the number of low-latency requests to external SDRAM and if the number of accesses to shared memory is a problem that comes from the architecture or whether it is a fundamental requirement from the application domain. In section 2.3 we have seen that by choosing a more specific application domain hardware optimizations can be made to reduce latency. In this section these optimizations are discussed.

As a first optimization, efficiency of the system is improved by allowing latency-tolerant data streams to bypass the cache. This is especially useful for streaming applications, where the written data is not used anymore by the producer. In this way the cache is used more optimally because the extra available cache lines can now be used for low-latency data.

To reduce the number of low-latency request to off-chip shared

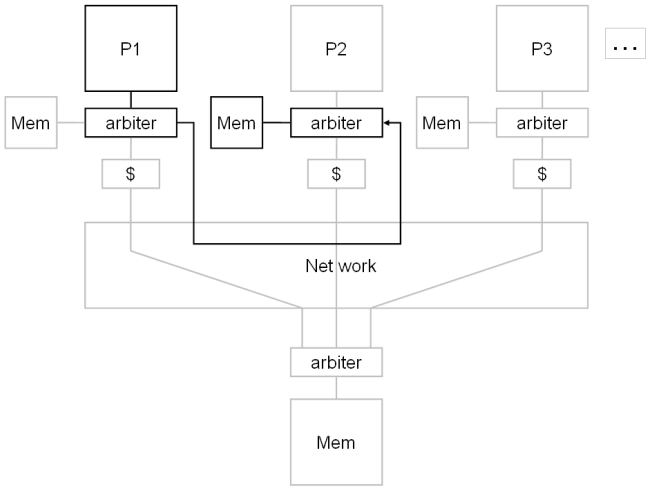


Figure 6. Peer-to-peer streaming

memory, [15] introduces buffers in on-chip local memory to enable peer-to-peer streaming. In this way a processor that produces data can directly write this data in the local memory of the consuming processor (see figure 6). This is not possible when only caches are used because caches do not allow a write from the network [15]. The advantages of this so-called *push architecture* are discussed in the next section. To enable peer-to-peer streaming point-to-point connections must be provided by the network and the platform must contain local memories for consuming PEs.

The proposed architecture template suits perfectly for the application domain "Mobile Consumer Platforms", as tokens can be produced by a streaming application on one PE and consumed by another PE. This allows for peer-to-peer communication, as produced streaming data is often not reused by the producer and therefore can be buffered in the local memory of the consuming party. Data is thus pushed to the consumer which has positive effects on efficiency.

### 2.4.2 *Push architecture*

The advantage of pushing (writing) data instead of pulling (reading) it from the producer is that pushing data does not have to be effected by the latency of the network. When pulling data, first a request is done to the producer. Then data is send from the producer to the consumer. In this way the network is traversed twice, doubling the latency. Request can be pipelined for optimization, but this is not always possible (i.e. due to incorrect branch prediction).

For pushing data the network only is traversed once, where latency can be hidden by buffers. No request is done but the producer constantly tries to push data to the consumer. Note the requirement for a buffer in the local memory of the receiving party in order to store

the data that cannot be processed immediately and flow control to ensure that data is not send if the buffer is full [2]. The consumer then has low-latency access to the data, as the data buffer is stored in the local on-chip memory and not in remote off-chip memory. Hence, latency-tolerant write streams are send to the on-chip local memories of consuming PEs and low-latency read requests access local on-chip memory.

For protocols such as AXI, where also for write commands response messages are generated, writes have to be posted. Normally, the write response is generated by the slave party. This makes the behavior of a write identical to that of a read, where for a read the response message is simply the read data coming back. For posted writes, the response data is generated by the network, thereby removing the latency needed to send the request over the network and the response back.

*Prediction is very difficult, especially about the future*

*— Niels Bohr*

# 3 A COMPOSITIONAL DESIGN APPROACH

The previous chapters show that closing the design productivity gap involves tackling some challenges. Platform-based design is advocated, which benefits the design of more complex systems. However, when scaling the design new problems appear. Bottlenecks shift from computation to communication as latency becomes an important performance limiter. The previous chapter introduced an architecture designed to minimize latency.

Contention at shared resources not only results in latency, also the behavior of a system becomes difficult to predict due to the uncertain load of the contention point. This makes the effect of other jobs on the behavior of a certain job uncertain. A predictable system allows for analysis instead of simulation, which result in decreased verification time and actual proof of system behavior. System analysis requires predicable arbitration and QoS of all resources. Trends in future SoC design envision predictability as a valuable asset to the architecture [10].

The goal of this thesis is to introduce a compositional platform-based design methodology and give proof of concept, as stated in section 1.1. Compositionality is part of any predictable design (3.2.2) and allows for individual subsystem simulation instead of complete system simulation. This chapter gives an overview of the definitions related to compositionality.

## 3.1 PREDICTABILITY

To be able to reason about the timing behavior of a system, i.e. to prove that throughput and latency requirements are met, the system should provide a predictable QoS. Predictability enables the designer to do analytical reasoning about the end-to-end behavior of a system and thus derive the minimum hardware to meet these timing requirements, which decreases the number of design iterations caused by intensive simulation and verification.

Unpredictability obviously is not a desired property. A simple example is that when a certain job gets more resources assigned one would predict the system to perform better. Unfortunately with current systems this is not always the case. Predicability enables the designer to make a tradeoff between the percentage of resources assigned to a certain job and the quality (i.e. the execution time or the number of missed deadlines) of the design. An increase in the percentage of available resources for a subsystem could result in a higher quality but decreases the available resources and thus quality of other subsystems.

In this report predictability is defined as follows:

**Definition 3.1.1 (Predictability)** *A system is defined as predictable if bounds on the temporal behavior of one job can be derived at design time*

This definition should be clarified. If one could reason about the behavior of a system, and guarantee that a system will end up in a certain state within a certain moment in time, the system is predictable. This statement is a bit vague, for instance the meaning of the word "certain". This can be made more specific by defining "certain" as "bounded". Bounding uncertainties enables predictability. If the temporal behavior of a system can be guaranteed to be within fixed bounds, the system is predictable. The key is in the size of the bounds which should be as small as possible, i.e. nobody is interested in the prediction that a system finishes within an infinite amount of years or with a 100% miss rate of deadlines.

Deriving these bounds can be done by modeling the application with data flow graphs [1]. Predictability of the system allows for a predictable mapping of an application on the architecture, which enables the derivation of best-case as well as worst-case performance and derivation of the minimum hardware requirements in order to meet these requirements.

Predictability deals with uncertainties of resources which should either be removed or at least bounded in order to provide predictable QoS. The uncertainties that endanger predictability are introduced by the hardware (i.e. cache line replacement policy and resource arbitration), the software application (i.e. conditional branches) and the environment (i.e. user input). These uncertainties make it difficult to reason about the temporal behavior of a job. In [24] uncertainty in the resource supply is distinguished from the uncertainty in the resource demand. The uncertainty in the resource supply is due to resource arbitration in the hardware of the multiprocessor system. The uncertainty in the resource demand is due to the data value dependent processing in the application and external events from the environment. In [24] also a solution is provided to bound these uncertainties. They are listed in the following two paragraphs.

The uncertainty in resource supply is bounded by making use of predictable hardware arbitration schemes. An arbitration scheme is predictable in the case it is known how long it maximally takes before a resource becomes available. The minimal time that the resource stays available must also be known. An example of a predictable arbitration scheme for resource access is *Time-Division Multiple Access* (TDMA). In this scheme, it is guaranteed that the resource can be obtained after a fixed amount of time and that during a fixed amount of time the resource stays available. The specifics of this scheme are discussed later.

The uncertainty in the resource demand is bounded by making use of admission control and hardware resource budget enforcement. Admission control takes care that sufficient hardware resources are available when a job is started. If there are insufficient resources available in the system then the user is notified that the job is rejected. Resource budgeting guarantees that a task of a job gets a certain amount of memory, bandwidth and processor cycles. By enforcing budgets it becomes impossible for a job to claim more resources than its budget. This scheduling technique is referred to as *non-work-conserving* scheduling.

Due to these enforced budgets the interference of tasks of different jobs is bounded. With enforced resource budgets, it looks for a job as if it runs on its own private hardware. In other words, resource budget enforcement creates for each job its own virtual platform with private hardware resources.

Optimizations can be achieved by allowing for *work-conserving* scheduling. This essentially means that if part of a budget is not used, it can be used by any other job. Hence, budgets are only enforced when multiple jobs claim the same budget. In this way budget utilization is optimal: resources are only idle when there is no traffic send. Note that it is still possible to derive worst-case behavior, but the average behavior is expected to be better [34].

A predictable system can only be achieved when all resources provide QoS. If one resource in a chain of predictable resources does not provide QoS, this removes the predictability and thus the QoS of the complete system.

## 3.2 COMPOSITIONALITY

Compositionality is a subset of predictability and deals with uncertainties introduced by sharing resources. Sharing resources introduces dependencies (contention) between jobs, which make jobs influence each others behavior making the composed system unpredictable.

In [20] a system is defined as compositional with respect to a specific property if the system integration will not invalidate this property once the property has been established at the subsystem level. This statement is too general as in this thesis we focus on the temporal behavior of jobs.

Therefore, the compositional property specific for this thesis is temporal behavior. Hence, the mapping of a job on an architecture is compositional if its temporal behavior is not effected by the temporal behavior of other jobs. More formally compositionality can be defined as followed:

**Definition 3.2.1 (Compositionality)** *A system is defined as compositional if the temporal behavior of one job is completely independent of other jobs [20]*

Compositionality does not require the behavior of a job to be predictable, hence uncertainties in i.e. conditional branches in the software and user input from the environment do not endanger compositionality (a job may harm itself as long as it does not harm other jobs in the system). Although compositionality does not make a system predictable, it does give several important properties to the system, i.e. if the behavior of one job can be verified, its behavior is identical (at a cycle-true level) if composed with any number of other jobs that are concurrently run on that system.

With the parallelism introduced by MPSoC many of these jobs can be active at the same time. When designing such a job the resource usage of the other jobs is often a unknown factor. Furthermore, the flexibility of MPSoCs causes the possibility of the number of sets of jobs that can be active at the same time, referred to as a use-cases, to be enormous. For example, when a certain application contains 10 jobs that can be

ran in parallel, the total amount of use-cases corresponds to an order of $2^{10}$. Obviously it becomes impossible to verify the correct behavior of all these use cases. Not even mentioned here are the transactions between use-cases, which for this example could possibly explode to $2^{10^9}$. It is desired only to verify the behavior of the 10 individual jobs and so cover all the use-cases. Again this confirms with the design productivity gap, which states that the design effort should be linear to the size of the design.

Compositionality is a necessary condition for predictability, the proof of this is given below.

**Proof 3.2.2 (Compositionality $\subset$ predictability)** *Assume: Task* A *can effect the temporal behavior of task* B *and task* A *is not known at design time Proof: Task* B *depends upon the the behavior of task* A *but the behavior of task* A *cannot be predicted and thus the temporal behavior of task* B *cannot be predicted*

Temporal compositionality is endangered by uncertainties due to interference at shared resources like processing elements, shared memory and the interconnect. The uncertainties can be bounded by making it mandatory for a job to reserve these resources at startup. Reservation then bounds the time it takes for a resource to become available (resource supply) and the time is stays available for that job (resource demand), without effecting the resource utilization of other jobs. As the aim for this thesis is to prove compositionality in its strictest form, there is not any variance in these bounds. Bounding these uncertainties makes the behavior of a shared resource predictable, and hence provide QoS. Therefore a compositional system can only be achieved when all shared resources provide QoS.

## 3.3 VIRTUALIZATION

Reservation bounds interference of shared resources and gives each job the illusion as if it is acting on its own virtual platform, thus introducing compositionality. This implicitly assumes that the uncertainties in resource supply and demand are bounded, which can be achieved in several ways using different concepts and techniques. One of these concepts is to virtually present a resource as several separate independent resources [32]. This method to implement compositionality is referred to as *virtualization* and is defined as;

**Definition 3.3.1 (Virtualization)** *Virtualization is the process of presenting the complete set of resources in such a way that the temporal behavior of one job does not effect the temporal behavior of another job. Each job thus obtains its own virtual platform.*

Virtualization requires the arbitration of resource contention points to provide predictability and thus bound resource demand. The arbiter has to ensure that the shared resource is accessed in such a way that jobs only use assigned budgets (or unused budgets for work-conservative scheduling), where assumptions have to be made or techniques have to be implemented to ensure that budgets are enforced. For this thesis

report we opt for the strictest form of compositionality, where there is not any variance in bounds. Jobs only use assigned budgets and unused budgets are not exploited. In this way jobs do not influence each other on a cycle-true level. This is not true for work-conserving scheduling, for which more refined techniques are needed to show that the property of compositionality is obtained [24].

Shared resources have to deal with inter-dependencies of different jobs, as they are not allowed to influence each others behavior. Therefore, every shared resource has to guarantee that when the resources is reserved by a certain job, the resource provides a *guaranteed service* (GS) to that job, disregarding the behavior of other jobs. This GS bounds the uncertainties in resource supply. The combination of resource reservation and GS enables predictability and hence ensures QoS of the shared resources.

With the decision to virtualize the system the designer has to accept that there are consequences with respect to probable cost constraints, i.e. timing and power consumption. This due to the fact that guaranteed service require resource reservation for worst-case scenarios, which can be expensive. For example, guaranteeing throughput for a stream of data implies reserving bandwidth for its peak throughput, even when its average is much lower. As a consequence, when using guarantees, resources are often underutilised. As the arbitration is decided to be non-work-conserving unused budgets are not assigned to pending requests of other jobs. One has to consider whether the increases in speed and energy consumption even out the increase in scalability and analysability. For this thesis all extra cost is excepted for obtaining the strictest form of compositionality.

In chapter 6 the implementation details of virtualization of the hardware platform are explained. In the following chapter the conceptual details of the compositional hardware platform are discussed.

*A system is computational just in case adopting the computational stance to that system offers useful generalizations and predictions about the operation of the system, over and above those generated by not adopting such a stance.*

*— Istvan Berkeley, Re: Searle's challenge, The Monist Interactive Issue*

# 4 COMPOSITIONAL PLATFORM-BASED SYSTEM-ON-CHIP DESIGN

In chapter 2 several problems are listed that contribute to the increased complexity for the design of a hardware platform. They concerned DSM problems as they propagate through the design phases. This has set the context of a new design methodology. A MPSoC platform is introduced that complies with the new technology trends. In chapter 3 virtualization is introduced as a method to enable compositionality. In this chapter virtualization is adopted to the proposed MPSoC platform to obtain a compositional platform design.

## 4.1 DESIGN DECISIONS AND CONSTRAINTS

Several design decisions and constraints are made based on the observations in chapter 2 and 3 that influence the platform. These decisions include:

- Resource virtualization requires resource reservation at every resource contention point. Therefore, every shared resource must be scheduled with a predictable hardware arbiter and has to provide guaranteed services (hence a shared resource must provide QoS). Shared resources are scheduled based on a non-work-conserving scheduling technique
- The architecture can provide a small amount of local memory to a PE. However, the size is too small to store all instruction, state variables and data. Instructions are therefore stored in remote memory
- The architecture allows for streaming optimization, because it is a characteristic for many of the applications within the "Mobile Consumer Platforms" application domain. Therefore the interconnect has to support point-to-point connections

The latter two items are based on the support of streaming applications to address latency issues. The first item is key in obtaining a compositional platform. This requires the arbitration to shared resources to be predictable and all shared resources to provide GS. For now it is assumed that the memory is chosen such that it provides QoS and the PEs are not shared by jobs. Note that as PEs are not shared

they do not have to provide QoS. What remains is the interconnect that must arbitrate contention points with a predictable hardware scheduler and provide GS to the connected IPs.

It is not realistic to make the assumption for the interconnect. As the platform design is communication-centric the interconnect plays an important role in the design process. Based on the mentioned decisions and constraints there are several requirements for the interconnect. These are further analyzed in the following sections.

## 4.2 INTERCONNECT

As discussed in chapter 2 wires rapidly become the bottleneck in SoC design. The communication architecture becomes a key element in the design flow. Currently, bus based interconnects are most frequently used. Advantages of a bus based interconnect is that it is a simple architecture, has low area cost and is easily extendable [6, 21, 18, 4]. However, despite bus evolvement in order to address drawbacks of bus based SoC design (i.e. ARM *AMBA AXI* protocol and multilayered bus design), bus based interconnects still have some disadvantages.

One of these disadvantages is scalability. With the increasing complexity of SoC design more IPs have to be connected resulting in increasing wire lengths. Intrinsic parasitic capacitance and resistance can become quite high, where the increased propagation delay may exceed a specified clock domain. As discussed in the section 2.1.1 SoC design concerns the deep sub-micron aspect, in particular the local and global physical wires on a chip. Local wires of an IP and the global wires connecting it to other IPs are often not distinguished in current day SoC design. As a result, the timing correctness of different IPs is inter-dependent: correcting a timing violation in one IP may invalidate the timing of another. Hence, the process of verifying the timing of the SoC as a whole (*global timing closure*), does not necessarily converge to a solution, and is specific for each design.

Communication should be decoupled from computation in order to solve this. The use of NoC is advocated [3, 4, 5, 6, 13, 18, 19, 21, 31, 33] as NoC can provide decoupling between computation and communication and furthermore the reusable communication structure scales very well for large designs.

An issue related to the use of on-chip networks which should be noticed is that current bus-like interconnects often use communication protocols such as AHB, AXI and PMAN. These protocols assume the use of respectively one master and one-to-many slaves and one slave and one-to-many masters, as can be seen in figure 7. This is not optimal for NoC design, since NoC provides the resources to establish communication between one-to-many masters and one-to-many slaves. Thus for future improvement this report envisions the importance that any interconnect, but especially NoC, provides flexibility with respect to communication protocols.

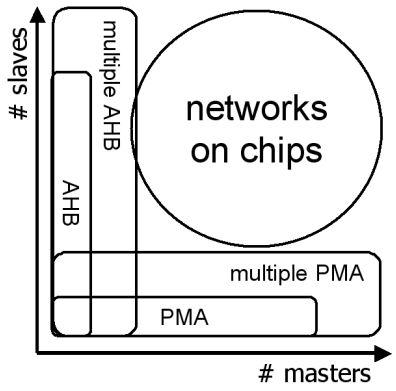


Figure 7. Busses are limited by the number of masters that can be active simultaneously and concentrators like PMAN by a limited number of slaves. Network on Chip allow the combination [33]

### 4.2.1 *Network-on-Chip*

In the previous sections we identified that the NoC could help in improving scalability, integration and reusability of a SoC. The reason for this can be deducted from the properties of a NoC, namely that NoCs [13, page 2]:

A.) Structure and manage wires in deep sub-micron technologies,

B.) Use wires efficiently through sharing,

C.) Scale better than busses,

D.) Are programmable for multiple and new task graphs, and

E.) Decouple computation from communication through well-defined interfaces, enabling IP blocks and interconnect to be designed in isolation, and to be integrated more easily.

As mentioned in item "E" the communication goes through well-defined interfaces. This ensures that the implementation details of the interconnect are hidden and computation and communication can be decoupled (figure 8a). The reason for this is that NoCs are traditionally designed using layered protocol stacks, where each layer provides a well-defined interface which decouples service usage from service implementation. This is shown in figure 8b.

The goal of this thesis is to introduce a compositional architecture. This effects the interconnect as it has to provide compositionality to the connecting IPs. As mentioned in chapter 3 this requires a guaranteed service from the interconnect.

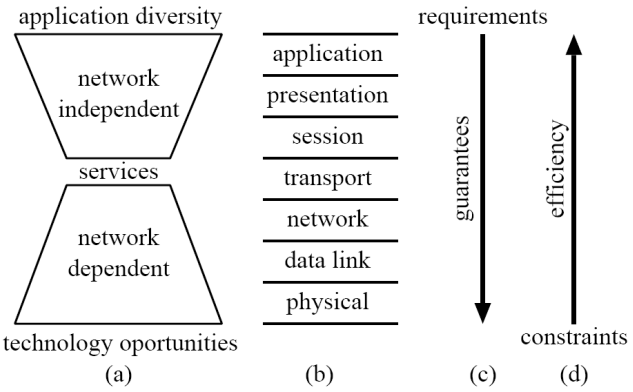


Figure 8. Network services: a.) Hide the interconnect details and allow the construction of diverse applications on top of them b.) Are built using a layered approach c.) Are driven by the application requirements d.) Their efficiency relies on technology and network organisation [8]

Most of the current interconnects, as well as NoCs have been built to offer *best-effort* (BE)[1] communication services [28]. BE communication infrastructures are not analyzable because the behavior of IPs and interconnect may be inter-dependent[2] as it does not take the communication of other tasks in account. Therefore, they require simulations to verify if the specified requirements are fulfilled. Because for complex chips the interconnect is a central component in the system, complete system simulations are required for system verification.

As mentioned in the previous chapter, covering worst-cases for all configurations is not possible through simulations, because they are based on sample (demanding) stimuli, which are never guaranteed to cover worst-case and corner cases. If any change, the system has to be resimulated again. In [28] three main problems with such systems are indicated: 1) long simulation times at each change, 2) numerous changes because of inter-dependencies which lead to change side effects, and 3) worst-case behavior is not necessarily covered.

To solve these problems, the use of throughput and latency guarantees is advocated [8]. The guarantees can be seen as requirements from the application (figure 8c). These guaranteed services make sure each IP module (i.e. computation and memories modules) can be designed in isolation, because the interconnect requirements are made explicit. As the communication has a guaranteed behavior, the composed system will function according to the specifications provided all IP modules meet their specifications. If IP modules have predictable behavior, the system behavior can be formally verified, without the need of simulations. If IP modules do not have predictable behavior, providing guarantees in the interconnect is still useful. This is because of the

1 As the name suggest BE performs a communication action as soon as this is possible, thus if the connection is not used for another communication actions

2 To avoid congestion in the lay out, the number of global wires should be minimized. This means that wires must be shared and hence bandwidth becomes a shared resource

system compositionality resulted from offering guarantees: the system does not need to be simulated as a whole, but simulating only IP modules is enough. Moreover, there are no inter-dependencies and therefore modifying parts of the system does not affect other parts of the system. Hence, by definition guaranteed services result in virtualization of the on-chip network.

However, as is explained in the previous chapters resource virtualization, e.g. network virtualization, has consequences with respect to probable cost constraints (figure 8d), which have to be considered by the designer.

## 4.3 DEGREES OF COMPOSITIONALITY

The platform is now adopted such that all shared resources provide predictable QoS, thereby achieving compositionality by means of virtualization. To show that the platform is compositional, an application is mapped on the hardware platform.

In section 2.3 a more detailed view on applications is given, where applications are divided into jobs that can consist of multiple tasks. The designer must map the tasks to PEs in such a way that an optimal system is obtained. Figure 9 shows an example task mapping of an application onto the proposed architecture template.

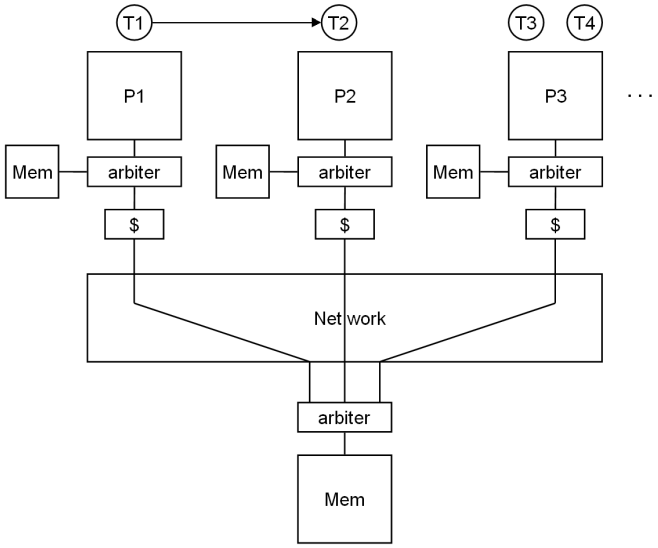


Figure 9. Task mapping

The application consists of three independent jobs, where job 1 is composed out of tasks T1 and T2 and jobs 2 and 3 are composed out of respectively T3 and T4. To prove compositionality of this system the behavior of each job must be shown to be independent of the behavior of other jobs. In this report three degrees of compositionality are identified. They are listed in table 2.

| Degree | Description | Status |
|---|---|---|
| First degree | Tasks are independent, at most one task per PE | supported |
| Second degree | Tasks may be inter-dependent, at most one task per PE | supported, not optimized |
| Third degree | Tasks may be inter-dependent, multiple tasks per PE possible | not supported |

Table 2. The three degrees of compositionality

The first degree of compositionality can be obtained if in figure 9 the dependency between tasks T1 and T2 is removed and T4 is left unmapped. Thus, three independent tasks, T1, T2 and T3, are mapped to respectively PE1, PE2 and PE3. Proof of first degree compositionality can then be given by showing that the behavior of the three tasks is independent on a cycle-true bases.

The figure shows that jobs may contain tasks that are mapped on different PEs (T1 and T2). This places dependencies on the behavior of the involved PEs, i.e. one task T1 may produce data for task T2 on a different PE. Proof of second degree compositionality is then given be showing that the behavior of the dependant PEs is not influenced by the behavior of the rest of the PEs. Thus, the behavior of T3 (T4 is not allowed since shared PE is not supported) is independent of the behavior of tasks T1 and T2. Optimizations are addressed in section 2.4.1, i.e. by writing data directly from producer to consumer, off-chip memory is not used for communication. The hardware for this optimization is available, only the buffers that are needed in the local memories are not [2].

The figure also shows that T3 and T4 share a PE. From section 3.3 can be concluded that sharing the PE endangers compositionality. In order to obtain third degree compositionality the PEs must provide QoS (i.e. predictable scheduling) to the mapped tasks, which is assumed not to be the case. Furthermore, task switching is complicated and requires i.e. caches to be either split or flushed before the next task is executed. These inter-task dependencies are addressed in [16]. For this architecture sharing of the PEs makes all resources shared resources. For reasons of complexity PEs are not shared in this design. The mapping of task T4 is therefore not allowed.

The goal of this report will be to proof first degree compositionality. Therefore, the architectural template is implemented. This is explained in the upcoming chapters.

*The chessboard is the world, the pieces are the phenomena of the universe, the rules of the game are what we call the laws of nature.*

*— T.H. Huxley, A Liberal Education, 1868*

# 5 PLATFORM SPECIFICATION

In the previous chapter an architecture template is proposed that enables compositionality. In order to give proof of concept this template is implemented by creating the corresponding hardware platform. Therefore, the building blocks of the platform are specified. This chapter introduces the compositional MPSoC hardware platform.

## 5.1 HARDWARE PLATFORM

To minimize the design effort IP blocks are reused as much as possible. This results in a hardware platform where only two blocks have to be implemented, namely an AXI shell (adapter) for Æthereal and an arbiter providing compositionality (discussed in chapter 6). In figure 10 the hardware platform is shown. The figure contains the following blocks;

- ARM11, Model of the ARM1176JZ-s. The ARM11 has separate caches for instruction and data. The ARM11 has four ports, an instruction port, data port, DMA port and peripheral port (not visible in the figure).
- Æthereal, fully connected instance of Æthereal (point-to-point connections instantiated at startup), where the NIs provide an AXI compliant communication interface by means of AXI shells
- ip_1036, AXI Bus-based interconnect that multiplexes the data on the ports depending on the address. Each ip_1036 has a arbitration block inside. Except for the arbitration in the ip_1036 at the remote memory side, where the arbiter is changed to provide compositionality, the arbitration block is standard round-robin as deployed from the reuse database. Arbitration is performed per (mirrored) slave port.
- ip_2114, AXI memory controller used to access embedded SRAM
- Mem, Model of embedded SRAM

Each of the master ports of the ARMs are connected to a mirrored master port of an ip_1036. A static address decoder ensures that data is multiplexed to the correct mirrored slave port. These slave ports can either be connected to the local memory, the remote memory or one of the local memories of the other ARMs.

Except for the port to the local memory, the mirrored slave ports of the ip_1036 are connected to the slave ports of Æthereal. In Æthereal

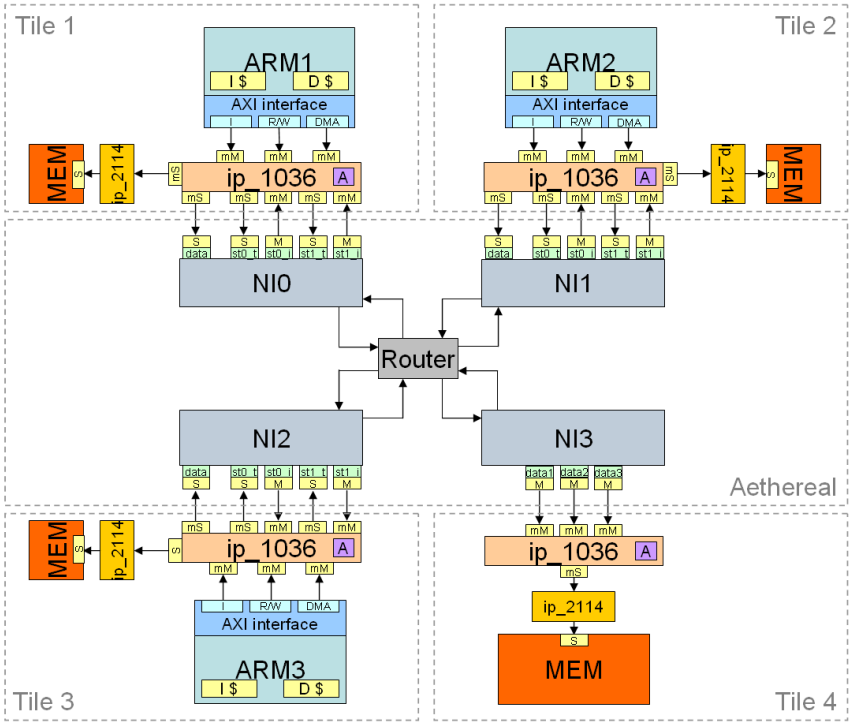


Figure 10. Hardware platform

these ports are dynamically connected to one of the master ports. Point-to-point connections are setup at runtime by a configuration master (discussed in section 6.5.2). These master ports are connected to one of the ip_1036s of the other tiles.

The ip_1036 at the remote memory multiplexes data from the corresponding master ports of Æthereal and arbitrates them with a predictable scheduling mechanism. There are several advantages and disadvantages of this platform, which are listed below:

- Reuse, a lot of hardware components can be reused, which decreases implementation effort
- The platform is uniform. As few different components as possible are used in the design.
- The cost of a uniform hardware platform is that possible hardware optimizations are not addressed. Therefore the platform has more cost overhead
- For streaming data the address is not necessarily required as the memory controller can determine itself where to load and store data. However, for this implementation the address is used to determine the location of the data in the memory. This is a more simple solution at the cost of extra bandwidth to send the address

In the following sections the IP blocks are discussed in more detail.

## 5.2 ÆTHEREAL NETWORK-ON-CHIP

A NoC satisfying the requirement of having QoS communication is the *Æthereal* NoC. Æthereal is a combined BE and GS infrastructure, meaning that it supports both guaranteed service communication (uncorrupted, lossless and ordered data transfer, and both latency and throughput over a finite time interval [8]) and best effort communication. As explained in the previous chapter, for now, there is no interest in the later, because the requirement for compositionality. In this section implementation details of Æthereal relevant for this thesis are explained. Further details can be found in [3, 8, 19]. This section summarizes these papers.

### 5.2.1 *Contention resolution*

Compositionality is endangered by interferences at shared resources. When a router attempts to send multiple data items over the same link at the same time contention is said to occur. As only one data item can be sent over a link at any point in time, a selection among the contending data must be made; this process is called *contention resolution*. As discussed before arbitration of data must be made predictable in order to enable compositionality. In circuit switching, contention resolution takes place at setup at the granularity of connections, so that data sent over different connections do not conflict. Thus, there is no contention during data transport, and time-related guarantees can be given. In packet switching contention resolution takes place at

the granularity of individual packets. Because packet arrival cannot be predicted contention cannot be avoided. It is resolved dynamically by scheduling which data items are sent in turn. This requires data storage in the router/ NI and delays the data in an unpredictable manner which complicates the provision of guarantees [8].

To implement guarantees, Æthereal uses contention-free routing, which is based on a *time-division multiplexed circuit-switching* (TDMC)[1] approach, where one or more circuits are setup for a connection. Circuits are created by reserving consecutive slots in consecutive routers/ NIs. This is, the circuits are pipelined, in the sense that if a circuit is set from router $R$ to router $R'$, and slot $s$ is reserved at router $R$, then slot $s+1$ must be reserved at router $R'$. In a slot $s$ at most one block of data can be read/ written per input/ output port. In the next slot, $(s+1)\%S$, the read blocks are written to their appropriate output ports. Blocks thus propagate in a store-and-forward fashion. The latency a block incurs per router is equal to the duration of a slot, and bandwidth is guaranteed in multiples of block size per $S$ slots. On these circuits, data received in one slot will be forwarded to the next router/ NI in the next slot. By setting up circuits, it is ensured that data is transported without contention. In this way throughput and flit latency are guaranteed [17]. Slots are therefore not only used for avoiding contention on a link, but also to divide up bandwidth per link between connections and to switch data to the correct output.

**Example 5.2.1 (Contention free routing)** *The entries of the slot table map outputs to inputs for every slot* $T(s,o) = i$*, meaning that blocks from input* $i$ *(if present) are passed to output* $o$ *at times* $s + kS$, $k \in \mathbb{N}$*. An entry is empty when there is no reservation for that output in that slot.*

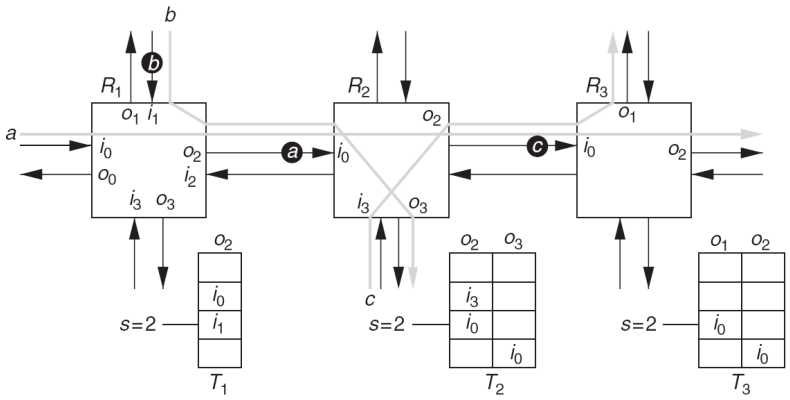


Figure 11. Contention-free routing [19]

*No contention can arise in table* $T$ *because there is at most one input per output for each slot. Sending a single input to multiple outputs (multicast) is possible. Figure 11 illustrates the operation of contention-free routing. It shows a snapshot of a router network with three routers* R1, R2 *and* R3 *at slot*

1 TDMA principle applied to circuit switching

$s = 2$, *indicated by the arrows pointing to the third slot in the table (recall that slots are numbered from 0). The size of the slot tables is* $S = 4$ *and only the relevant columns are depicted. Three connections,* a, b *and* c, *are shown with the gray arrows. The black circles represents packets on the connection with the corresponding letter. Packets* a *and* c *were switched from the input of the network to their output links in slot 1. In slot 2, shown in Figure 11, packet* b *is switched from input* $i_1$ *to output* $o_2$ *in router* R1, *as indicated by the slot table* $T_1(2, o_2) = i_1$. *Packets* a *and* c *are switched similarly by the network. The slots reserved for a block along its path from source to destination increase by one (modulo* S*). If slot* s *is reserved in a router, slot* $(s + 1)\%S$ *must be reserved in the next router on the path as mentioned earlier. The assignment of slots to connections in the network is an optimization problem. [8, 19]*

### 5.2.2 *Network Interface*

To enable reuse of existing IPs, there has to be a smooth transition from busses to NoCs. Æthereal can provide this property, since it uses a shared memory abstraction (e.g. read and write) to the IP module [3]. Communication is performed using a transaction-based protocol, where master IP modules issue request messages (e.g. read and write commands at an address, possibly carrying data) that are executed by the addressed slave modules, which may respond with a respond message (i.e. status of the command execution and possibly data). A transaction is thus defined as a request message possibly followed by a corresponding response message.

In the Æthereal NoC, all signals are sequentialized in request and response messages, which are supplied to the NoC. There they are transported by means of packets. Sequentialization is performed to reduce the number of wires, increasing their utilization and to simplify arbitration. Packetization is performed by the NI and is thus transparent to the IP modules.

In this way the internal Æthereal protocol can provide backwards compatibility to existing on-chip communication protocols like AXI, OCP and DTL but also allows future protocols better suited to NoCs.

The Æthereal NI provides network services at the transport layer or above in the ISO-OSI preference model. This model describes 7 layers, e.g. the physical, data link, network, transport, session, presentation and application layer. Since the transport layer is the first layer where offered services are independent of the network implementation this is the key ingredient in decoupling between computation and communication, which advantages were described in previous sections. Hence, it also provides these services to the application layer which helps us with reasoning about end-to-end behavior of software application to software application.

The Æthereal NoC offers its services on connections, which can be point-to-point (one master, one slave), multicast (one master, multiple slaves, all slaves executing each request) and narrowcast (one master, multiple slaves, a request is executed by only one slave). Connections are composed of unidirectional point-to-point channels (between a single master and a single slave), where at both sides of the channel a source and destination queue is located. To each channel, properties

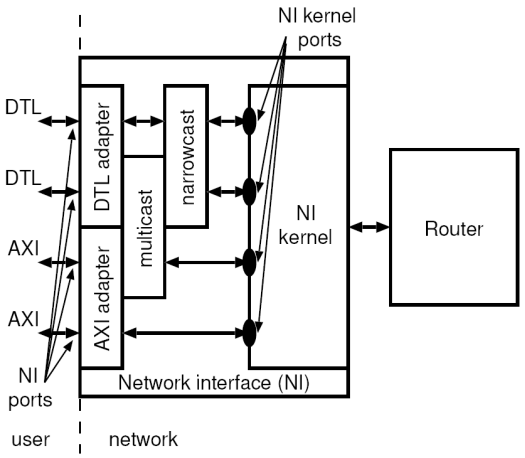


Figure 12. NI architecture [3]

are attached, such as guaranteed message delivery or not, in order or un-ordered message delivery and with or without timing guarantees.

Real-time communication is achieved by the use of TDMC. This ensures that guaranteed services as throughput, latency and jitter can be provided. Throughput guarantees are given by the number of slots reserved for a connection. Slots correspond to a given bandwidth: $B_i$, and, therefore, reserving N slots for a connection results in a total bandwidth of $N * B_i$. The latency bound is given by the waiting time until the reserved slot arrives and the number of routers data passes to reach its destination. Jitter - on the flit level - is given by the maximum distance between two slot reservations.

In the architectural view of the Æthereal NoC the design of the network interface is split in two parts: a) the NI kernel, which implements the channels, packetizes messages and schedules them to the routers, implements the end-to-end flow control, and the clock domain crossing, and b) the NI shells, which implement the connections (e.g., narrowcast, multicast), transaction ordering for connections, and other higher-level issues specific to the protocol offered to the IP (see figure 12).

*NI kernel*

The NI kernel (see Figure 13) receives and provides messages, which contain the data provided by the IP modules via their protocol after sequentialization. The message structure may vary depending on the protocol used by the IP module. However, the message structure is irrelevant for the NI kernel, as it just sees messages as pieces of data to be transported over the NoC.

The NI kernel communicates with the NI shells via ports. At each port, point-to-point connections can be configured, their maximum number being selected at NI instantiation time. A port can have multiple connections to allow differentiated traffic classes, in which case there are also `connid` signals to select on which connection a message is

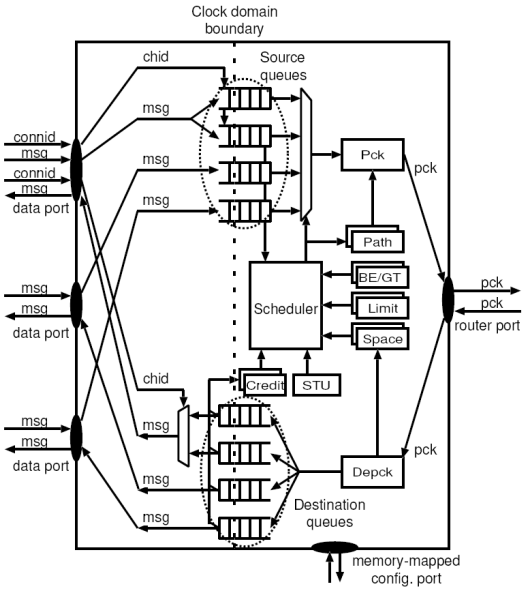


Figure 13. NI kernel [3]

supplied or consumed. In the NI kernel, there are two message queues for each point-to-point connection (one source queue, for messages going to the NoC, and one destination queue, for messages coming from the NoC). Their size is also selected at the NI instantiation time.

Counters `credits` and `space` are used for end-to end flow control. From the source queues, data is packetized (`Pck`) and sent to the NoC via a single link. A packet header consists of the routing information (NI address for destination routing, and path for source routing), remote queue id (i.e., the queue of the remote NI in which the data will be stored), and piggybacked credits. There are multiple channels which may require data transmission, the `scheduler` is used to arbitrate between them.

*NI Shell*

With the NI kernel described in the previous section, point-to point connections (i.e., between one master and one slave) can be supported directly. These type of connections are useful in systems involving chains of modules communicating point-to-point with one another (e.g., video pixel processing). For the future it is envisioned that also more complex types of connections, such as narrowcast or multicast and to provide conversions to other protocols, are supported by the shells. As an example, in figure 12, a NI is shown with two DTL and two AXI ports. All ports provide point-to point connections. In addition to this, the two DTL ports provide narrowcast connections, and one DTL and one AXI port provide multicast connections. Note that these shells add specific functionality, and can be plugged in or left out at design time according to the requirements. NoC instantiation is simple, as an XML

description can be used to automatically generate the VHDL code for the NIs as well as for the NoC topology.

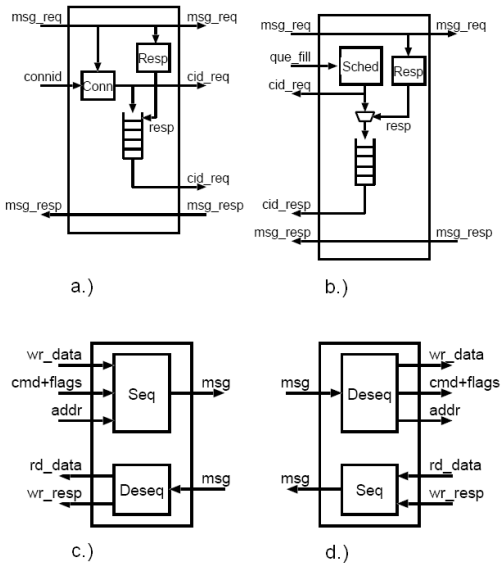


Figure 14. NI shell examples [3]: a.) Narrowcast shell b.) Multi-connection shell c.) Target shell d.) Initiator shell

Examples of shells are shown in figure 14. Figure 14a shows the narrowcast shell. The narrowcast connection is implemented as a collection of point-to-point connections, one for each master-slave pair. Within a narrowcast connection, the slave for which the request is destined is selected based on the address (Conn block). The address range assigned to a slave is configurable in the narrowcast module. To provide in-order response delivery, the narrowcast must also keep a history of connection identifiers of the transactions including responses (e.g., reads, and acknowledged writes), and the length of these responses. In-order delivery per slave of request messages is already provided by the point-to-point connections.

When a slave using a connectionless protocol (e.g., DTL) is connected to a NI port supporting multiple connections, a multiconnection shell must be included to arbitrate between the connections. A multi-connection shell (see Figure 14b) includes a scheduler to select connections from which messages are consumed, based e.g., on their filling. As for the narrowcast, the multi-connection shell has a connection id history for scheduling the responses.

In Figures 14c and 14d, a target and initiator shell are shown that implements a simplified version of a protocol such as AXI. The basic functionality of such a shell is to sequentialize commands and their flags, addresses, and write data in request messages, and to de-sequentialize messages into read data, and write responses.

## 5.3 PROCESSING ELEMENTS

As the network interface of Æthereal provides moderate protocol conversion shells, a PE can in essence be any DSP, VLIW or RISC. In order to prove that the concept of compositionality holds for an actual design, any selection of (possibly the same) PEs is possible. For this report the PE used is a model of the *ARM1176JZ-s*. The ARM1176JZ-s processor incorporates an integer core that implements the ARM11 ARM architecture v6 [23]. The ARM11 has an AXI interface, used to connect to the four different ports, namely; the instruction fetch port, the data read/write port and the DMA port all using a 64 bit AXI interface and the peripheral port using a 32 bit AXI interface.

The ARM11 has separate caches for instruction and data, which optimize the performance.

The data traffic behavior generated by the ARM is assumed to be unpredictable, and thus does not provide QoS. The ARM can stall at any moment in time, which should not effect the property of compositionality. However, the ARM complies with AXI and the AXI ordering model as stated in [22].

## 5.4 MEMORY

The memory controller used for this use-case is the ip_2114, which is an AXI embedded SRAM/ROM controller that provides an interface to embedded SRAM/ROM memory instances through an AXI compliant communication network in an AXI subsystem. Various AXI masters in the network can communicate with the embedded SRAM/ROM memories through this controller. The basic functionality that is incorporated in the controller is to derive memory commands (e.g. read and write) from AXI commands and to transfer the data from AXI to memory or vice versa based on the type of command [30].

Because AXI allows for parallel read and write requests and the memory is single-ported the memory slave port is arbitrated. Arbitration of the memory controller is round-robin and is on an AXI burst granularity. The write data interleaving depth of the controller is one, which means that the memory controller can not accept interleaved write data from different requestors. Hence, an AXI write transaction has to be completed before a next request is accepted (*single-threaded*). More information about the memory controller can be found in [30].

| Request | Latency (cycles) | Throughput (words/cycle) |
|---|---|---|
| Read request | 0 | 1 |
| Write request | 1 | 1 |

Table 3. Memory controller, throughput and latency

The advantage of using this memory and memory controller is that its behavior can be predicted and thus provides QoS. The cycle latency from AXI to memory command and throughput are fixed (see table 3), thereby making it possible to make assumptions on the QoS the

memory controller provides. This is necessary in order to determine budgets in such a way that they can be implicitly enforced. This is explained in the next chapter.

Since, unlike PEs, memory is a shared resource it has to provide QoS in order to ensure that budgets are not exceeded. From a performance perspective, this quality should be as high as possible. For this project all memories are embedded SRAM and are connected by a SRAM/ ROM memory controller with AXI interface. The author recognizes that the use of embedded SRAM for remote memory is not realistic, but it suffices for a first proof.

## 5.5 ARBITER

From the architectural template can be concluded that the arbiter does more than just arbitration of contention points at slave ports, it also acts as a multiplexer. Based on the provided address the arbiter has to determine which target port to access. The following possibilities are implemented in order to comply with the architectural template proposed in [15]:

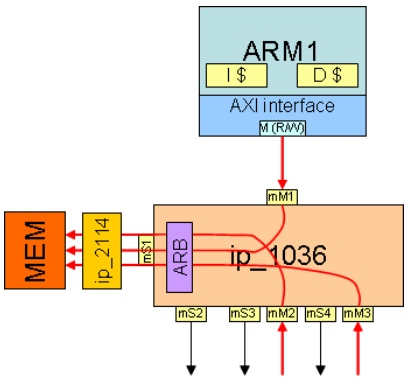


Figure 15. Multiplexing

- CPU must be able to read/ write the local memory of CPU $(mM1 \Rightarrow mS1)$
- CPU must be able to write the local memory of CPU' $(mM1 \Rightarrow mS3 \wedge mM1 \Rightarrow mS4)$
- CPU may be able to read the local memory of CPU' $(mM1 \Rightarrow mS3 \wedge mM1 \Rightarrow mS4)$
- CPU' must be able to write the local memory of CPU $(mM2 \Rightarrow mS1 \wedge mM3 \Rightarrow mS1)$
- CPU' may be able to read the local memory of CPU $(mM2 \Rightarrow mS1 \wedge mM3 \Rightarrow mS1)$
- CPU must be able to read/ write the remote memory $(mM1 \Rightarrow mS2)$

For a first implementation the local memory of each PE is not used. Therefore arbitration to the local memories can be round robin (does not endanger compositionality since the memory is not shared anymore). Arbitration is performed per slave port. The other slave ports of the multiplexer at the processor are not shared as they have a dedicated master and therefore do not have to be arbitrated (can be parallel connections in case of a multilayer bus). Arbitration to the remote (shared) memory however is arbitrated in such a way it provides compositionality to the connecting IPs. By making assumptions on the worst-case latency of memory transactions, budgets can be derived. The arbitrator in the memory controller does not effect compositionality anymore, since arbitration is already done in the IP block above. Since the PE has an AXI interface, the interfaces of the arbiter should be AXI compliant.

In order to minimize design effort, also for the thesis assignment, IP blocks are reused as much as possible. The arbiter is decided to be implemented using a standard bus-based AXI interconnect (ip_1036), where for the instance at the remote memory the arbitration is changed from the standard implemented round-robin arbiter to a arbiter that provides compositionality. The ip_1036 AXI interconnect is a simple multi-layer AXI switch matrix, optimized for low area and low latency. It does not contain packet buffers to store intermediate data words. It does not perform re-packetization or data reordering, other than optional write data interleaving from different masters. All data buffering is expected at AXI masters, AXI slaves or data adaptor units [29].

This simple architecture imposes a few constraints to external AXI masters and system performance behavior. The following constraints must be considered :

1. The ip_1036 does not support masters with a write interleave depth greater than 1, since such a master can supply interleaved write data. When such a master would address a slave that is not capable of handling this interleaved write data, the interconnect is responsible for the packet storage and data reordering. This is presumed to be a too complex task for the ip_1036.

2. An AXI master can have a pending read or write request to a single slave only. When a master issues a read or write command request to second slave, it will not be granted by the interconnect. When the interconnect would allow masters to perform read or write request to multiple slaves, of which at least one is capable in read data or write response reordering, a deadlock situation can occur.

   When the interconnect would allow masters to perform read or write request to multiple slaves, of which at least one is capable in read data or write response reordering, a deadlock situation can occur as is shown in figure 16. The read data (or write response) reorder deadlock problem can be explained as follows :

   - Master 1 sends read address to slave A (with ID=0)
   - Master 1 sends read address to slave B (with ID=0)
   - Master 2 sends read address to slave B (with ID=0)
   - Master 2 sends read address to slave A (with ID=0)

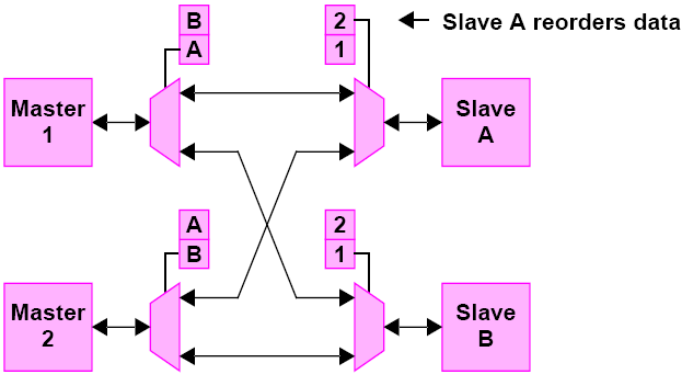


Figure 16. Re-ordering deadlock [29]

- Master 1 is waiting for read data from slave A first (read data must come in order)
- Master 2 is waiting for read data from slave B first (read data must come in order)
- Slave A has re-ordered its read data and is trying to send it to Master 2
- Slave B is trying to send its read data back to Master 1

3. An AXI master can perform consecutive (non-pipelined) write bursts only. This constraint is caused by the AXI ordering rule as defined in section 8.5 of [23], "The order in which a slave receives the first data item of each transaction must be the same as the order in which it receives the addresses for the transactions." When a slave would accept a second write command from a master the write data interleaving process would be blocked as is shown in Figure 17.

   This ordering rule requires that the write data for the second transfer of master 1 must be presented to slave A before the write data of masters 3 and 4. But this write data cannot be presented since master 1 can only generate in order write data (see the first item). The write data for masters 3 and 4 can therefore not be interleaved to slave A, although the slave port was capable of doing so. The ip_1036 AXI interconnect will de-pipeline the pipelined write commands as supplied by the AXI master.

4. The ip_1036 can contain combinatorial paths between inputs and outputs on the master interfaces of the address write, address read and write channels of the interconnect. In principle this is a violation of the AXI protocol, that does not allow these combinatorial paths. However for a zero latency interconnect these combinatorial paths are inevitable. The ip_1036 provides possibilities to ensure that there will be no combinatorial paths on the master interfaces of the interconnect.

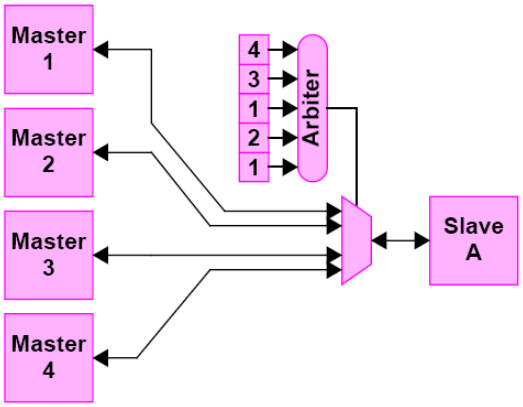


Figure 17. Write-interleave blocking [29]

## 5.6 IMPLEMENTATION CONSTRAINTS

In the previous sections some assumptions and constrains were made to the platform. They are recapitulated in the following list;

- PEs are not assumed to provide predictable scheduling. This constraints the possibilities for compositionality, as this predictability is used for deriving worst-case budgets. Therefore PEs are not shared by jobs, a job only runs on one PE.
- All instances of the memory controller are assumed to be single-threaded and memory is assumed to be single-ported
- Off-chip SDRAM is simulated as an on-chip SRAM. The size of such a SRAM is not considered feasible for actual implementation but suffice for a first proof of concept
- Latency and throughput requirements from AXI to memory command are assumed to be fixed. This guarantees the service rate of the memory required to derive worst-case budgets
- The word size of Æthereal is assumed to be 32 bit
- Data size of AXI is assumed to be 64 bit
- Each tile is connected to only one NI. This means that physical links in the NoC are shared by multiple connections.

*Strive for perfection in everything you do.*
*Take the best that exists and make it better.*
*When it does not exist, design it*

*— Sir Henry Royce*

# 6 IMPLEMENTATION

In the previous chapters the conceptual details of a compositional design platform are explained. The basic blocks for building the platform are specified. This chapter focusses on the implementation details in order to give proof of concept.

## 6.1 VIRTUALIZATION OF THE HARDWARE PLATFORM

As discussed in section 3.3 shared resources must be reserved in order to prevent contention and provide compositionality, which can be achieved by a predictable arbitrator based on budgets. This requires bounding shared resource supply and demand. Bounds on resource supply can be achieved by *time-sharing* using techniques such as time-division-multiplexing. In the previous chapter it is explained how the NoC ensures the property of compositionality. In this chapter it is explained how the rest of the platform is adapted to remain this property. Shared memory must be reserved for each task in a job. For this implementation a TDMA arbiter ensures that time is divided in slots (budgets) and each time-slot is assigned to one and only one task. Hence, the platform is non-work-conserving. Tasks that require more resources can be assigned to more, not necessarily consecutive, slots. This gives the job the illusion that it does not share the resource, since every task of the job has a fixed percentage of access-time to the resource which is not effected by other tasks.

Note that excepted AXI transactions cannot be interrupted and continued at another moment in time, unless stalling the rest of the transactions [22]. The reason for this is explained later. By reserving slots it is assumed that the assigned slot-time can be used effectively, i.e. reserving slots that cannot be used because a previous task from another job is still pending is not useful. Therefore, uncertainties in resource demand must be bound. Implementation can be achieved by obligating the resource to provide guaranteed service. This allows for techniques that ensure that a task will not exceed assigned slot boundaries, since QoS enables worst-case execution times to be derived at startup. Slots are thus sized for the biggest non-preemptive transaction in the system. In this way uncertainties in resource demand can be bounded by using admission control, where budgets are implicitly enforced because admission control ensures that no tasks are admitted which worst-case execution time could exceed the slot boundary.

The size and assignment of these slots highly effects the performance of the system. In this thesis slots are allocated statically, which means that slot reservation is done at startup and during execution of the

system this does not change. Although applications might be unknown at design time, the maximum number of concurrent tasks running on a system is known. To give each subsystem the virtue of having its own hardware platform each task must have at least one slot of the TDMA time wheel. Depending on the resource usage of the task multiple slots can be reserved, resulting in a better quality (not necessarily a better overall quality). It is up to the designer to make the best tradeoff in the reservation of slots.

Virtualization with static slot allocation requires all available resources to be distributed over the maximum amount of concurrent tasks, thus the available resources are divided over the maximum amount of concurrent tasks in the system. The obtained system can now be very inefficient for several reasons. First, when fewer jobs are active than predicted the extra available resources cannot be exploited and are wasted. Secondly, slots might not be completely used effectively because they are sized based on worst-case execution times and hence they will be grossly over-dimensioned [34].

Time-slots are thus sized for worst-case scenarios. In practise however, there may be only few (or even no) occasions where this worst-case scenario occurs, thus wasting a lot of slot-time. For performance reasons this must be optimized. For this thesis this is achieved by not only allowing transactions to be processed at the beginning of each time slot, but also at any other moment of time within that slot. Note that this is not free of risk, as a large write request can now be granted at the end of a slot, thus exceeding slot bounds (AXI transactions cannot be explicitly enforced by pre-emption). Therefore, the arbiter only grants access to the resource when the next transaction can be completed within that same time-slot. This is shown in figure 18.

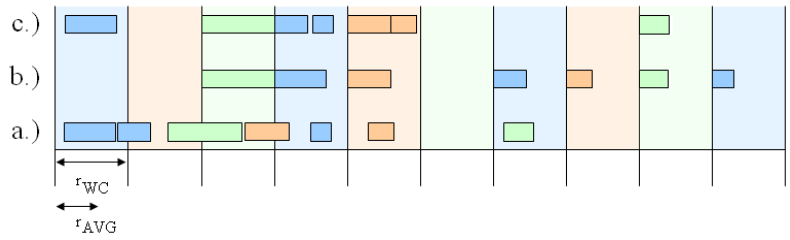


Figure 18. Guaranteed services require worst-case ($r_{WC}$) resource reservations that can be significantly higher than the average ($r_{AVG}$). a.) Without admission control transaction can be served when possible and thus exceed slot-times ($r_{WC}$) and use unassigned slots. b.) Admitting transaction at the beginning of each assigned slot prevents this at the cost of high amounts of unused slot-time. c.) This can be optimized by admitting transactions when enough slot-time is left to process them.

Note that there is still room for improvement. Dynamic slot allocation would take task switches (change in use-case) and resource utilization into account resulting in better performance (work-conserving versus non-work-conserving). The behavior of an application should be

predicted in advance and resources should be allocated accordingly.

This however requires a complex and dynamic resource manager, since with every change in use-case the resource demand changes and hence should also the resource supply. This re-configuration must be unnoticeable for the user and also this event itself should be predictable. Furthermore, work-conserving scheduling makes it difficult to prove that compositionality is obtained because not all dependencies between jobs are removed but they are bounded. The design of a more complex resource manager is beyond the scope of this thesis as for this thesis it is sufficient to prove the concept of compositionality.

Also TDMA may not be the best technique to virtualize the platform, as the main communication protocol is AXI. In AXI data can be transferred in burst, where only one command and address word are needed to send multiple data words. This hampers the *pre-emption* (pre-empt a transaction and start with another transaction before continuing it) of data transactions, since somebody has to keep track of the data and the corresponding command and address. Slots can be sized for average-case behavior, but then a worst-case transaction does not fit in one slot anymore. In that case, these transactions are distributed over multiple slots and consequently hard real-time deadlines can be missed. Therefore, either smaller transactions must be presented to the resource or the resource is *multi-threaded* to enable pre-emption of transactions. As AXI transactions are not split for this thesis and the memory controller specified in section 5.4 does not provide multi-threading, transactions must be processed in one assigned slot in order not to stall other jobs. This involves some more problems that are resolved in the following sections.

## 6.2 TRANSACTION-BASED VALID SIGNALLING

Due to uncertainties in the behavior of the ARM and the scheduling of Æthereal transaction data is scattered over time. Hence, the *arrival curve* of data is unpredictable. This endangers compositionality as the memory controller is single-threaded, hence it is not possible to pre-empt transactions. In AXI every transaction must have the number of transfers specified in the command. No component can terminate a burst early to reduce the number of data transfers [22]. When data scattering results in exceeding slot boundaries compositionality with hard real-time deadlines is lost, as following slots cannot be used due to the pending transactions. Budget enforcement is implemented implicitly and is not forced, which requires bounding data scattering to be able to provide a predicable *service curve*.

Currently, there is no implementation available of a multi-threaded memory controller. This is due to the complexity of the implementation and since TDMA arbitration is not used nowadays in memory controllers.

Bounding data scattering to prevent data interleaving is implemented by buffering data transactions (see figure 19). Data transactions are not allowed to pass a contention point when it is uncertain that the complete transaction can pass that contention point before the end of the assigned time-slot. If this would be allowed the situation described

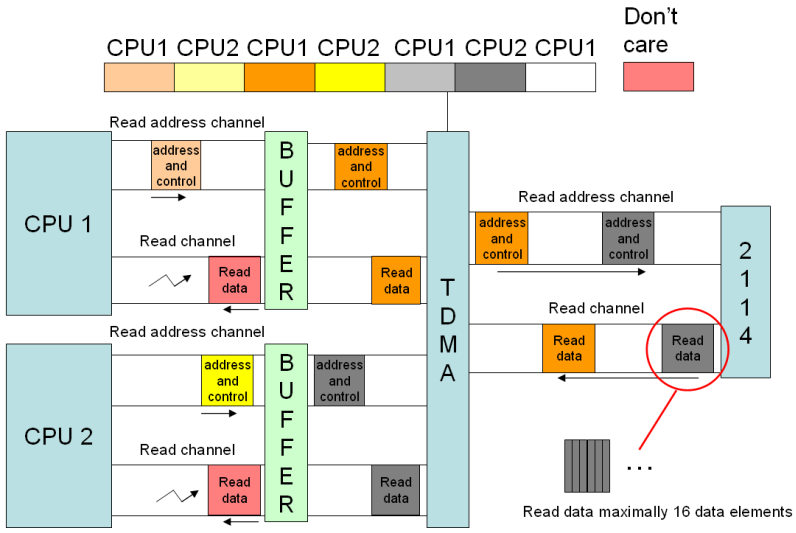


Figure 19. Read transaction scheduling: transactions are only scheduled by the TDMA arbiter when the complete request message is available in the buffer and there is enough space left in the buffer to receive the response message. In this way bounds can be determined on the temporal behavior of a transaction.

in the previous section could occur, where transactions are crossing slot boundaries influencing the behavior of other tasks.

Note that transactions have to be buffered in both directions, as AXI read and write transactions also include response data coming back. If the NoC is not able to process response data i.e. because the network is flooded the same situation could occur, where now the memory controller is waiting until it can send the response data to the network. If this waiting time exceeds slot boundaries other transactions can be influenced.

Figure 19 shows an example of a read transaction. If a complete AXI request is available in the buffer and there is enough space left in the buffer to receive the complete response data, a transaction-valid signal is send out to the arbiter. This signal indicates that the transaction is ready to be scheduled. The predictable arbiter then waits until the assigned time-slot of the corresponding task is active before granting the request, thus providing a predictable service curve. The size of the slots must ensure that that memory controller has finished the transaction, implicitly bypassing the arbitration in the memory controller. This is because it cannot be the case that the memory controller is busy at the end of a slot (formally, predictable arbitration implies memory controller arbitration). Slot sizes are chosen such that if the correct slot is active, the transaction can be completed within the same slot-time.

Restrictions on AXI itself prevent the buffers and slot-sizes of becoming unfeasible large, as a data burst can maximally contain 16 data

elements (see figure 19) [22]. Buffers are made large enough to contain the 16 data elements and slot-sizes are big enough to process the complete transaction. The latter can be done because the fixed latency of memory transactions between memory controller and memory (see table 3). The length of an AXI burst can be determined by snooping the correct signals of the read/ write address channel.

Buffers typically have a negative effect on performance and area cost and therefore have to be limited. An optimization could be to disallow data burst, each data element should be preceded by a command and address (*Memory-Mapped Input/ Output* (MMIO)). This minimizes the buffer size, but has negative effect on performance and puts concessions on AXI and is not considered for now. However, some optimization is achieved by reusing the source and destination queues of Æthereal (figure 20).

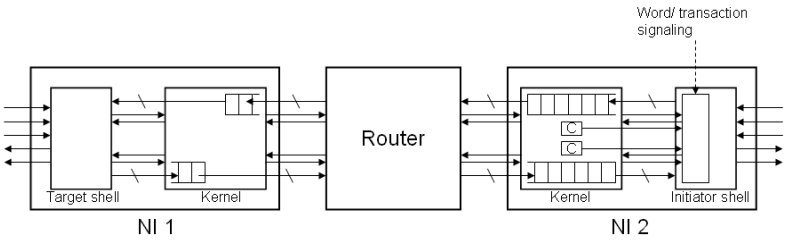


Figure 20. Transaction signalling: the NI kernel contains queues which are used to store transactions. A counter indicates the current filling of the queues. A separate module in the shell is capable of polling the counter signals to allow for transaction-based valid signalling

The buffer shown in figure 19 is thus split into a source and destination queue as shown in figure 20, where the depth of the queue at the slave party is chosen sufficiently large to contain a complete write message in the source queue and a complete read message in the destination queue. Note that a complete transaction is only buffered here. A counter is implemented per queue to indicate the current filling of the queue. As shown in the figure current implementation of the counter is crossing clock boundaries, which makes it not possible for the current implementation to operate in differen clock domains. In order to create a clean cut between protocols the shell at the slave party implements this transaction-valid signalling. This is explained in section 6.3.3.

Note that as the source and destination queue are in the NoC, the protocol is not AXI but a native Æthereal protocol. In section 5.2 it is explained how AXI signals are converted into message packets which are send over the network and converted back to AXI when exiting the network. Implementation details of the shells bridging between protocols is explained in the following section.

## 6.3 AMBA AXI SHELL

In section 5.2.2 the Æthereal NI shells are introduced. As the protocol used by the IPs is AXI, the NoC must be deployed with AXI shells to enable communication between connected IPs.

Unfortunately, such AXI shell was not available and has to be implemented. This means that an AXI target shell is implemented to bridge AXI to native Æthereal and an AXI initiator shell to bridge between native Æthereal and AXI.

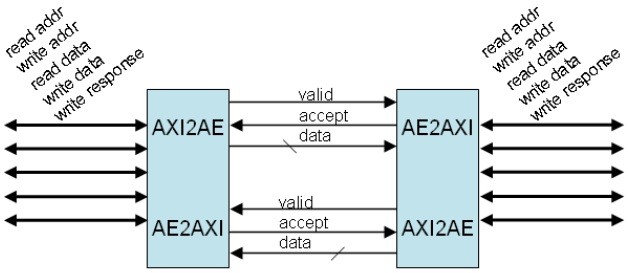


Figure 21. Combined AXI shells

Figure 21 shows a cascading of the AXI target and initiator shell. The outside wires show the five AXI channels at each shell. In [22] an overview is given of the signals in each channel. Every channel has a valid and accept signal used for the handshake protocol. Native Æthereal also has handshake signals and a data bus for both directions (read and write data). The target shell is connected to the master IP and the initiator shell to the slave IP.

As the Æthereal NoC is based on message passing, all AXI signals are sequentialized in request and response messages, which are supplied to the NoC. There they are transported by means of packets. Sequentializing AXI signals into messages is one of the main tasks of a shell. How the signals are sequentialized is described in the *message format* that is shown in figure 22. In the next section the message format for AXI is introduced.

### 6.3.1 *Message format*

In this message format the read and write signal group are fit in the command word. It is assumed that the data width of Æthereal is 32 bits, which requires the 64 data bits of AXI to be split over two words. Also, the word size of Æthereal makes it not possible to fit the write strobes needed by AXI (eight bits per 64 bit of data) in the same word as the data. Therefore next to the two data words another word is added for the write strobes.

Note that this extra word for the write strobes could be avoided by increasing the word size of Æthereal to 36 bits (four bits per 32 bits of

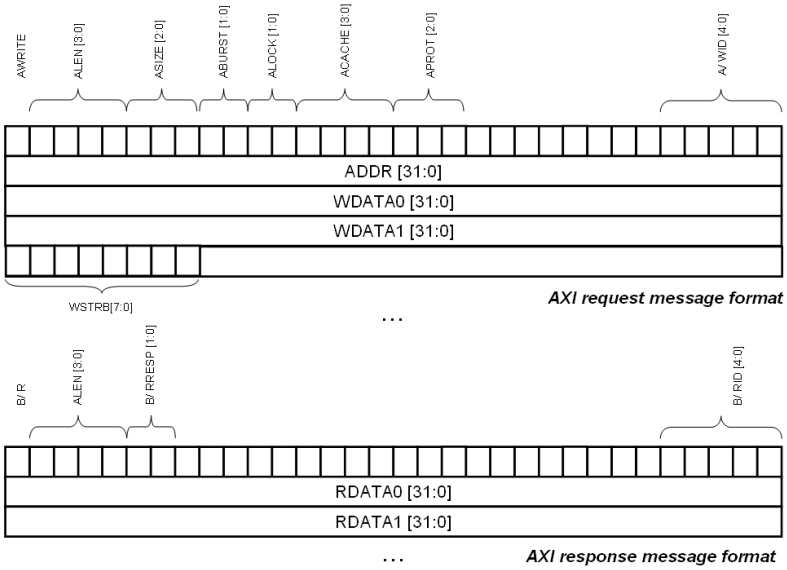


Figure 22. Example of an AXI message format

the AXI data). This however, would make Æthereal more dependant on AXI and is not considered as a clean cut between protocols.

### 6.3.2 *AMBA AXI target shell*

When a master IP issues a read or write request via the network, first the signals are bridged to a native Æthereal protocol. This protocol is based on message passing. It is based on a read and write data bus combined with a valid and accept signal per bus used for the handshake protocol. It also contains a write flush signal, which for this implementation is short-circuited and is not mentioned anymore in the continuing of the report. Furthermore the AXI target shell:

- Sequentialize AXI commands and their flags, addresses and write data in request messages
- Down-scaling 64 bit AXI data to 32 bit Æthereal data for write requests
- Up-scaling 32 bit Æthereal data to 64 bit AXI data for read response
- Include transaction-based round-robin arbitration for contention between read and write requests
- Post normal access write requests
- Pipeline read requests
- De-sequentialize response messages into read data

*Request*

As the word-size of Æthereal is 32 bit, AXI request data is sequentialized in 32 bit words. The request message format shown in figure 22 is used for this implementation. All AXI command signals are packet in the first Æthereal word. This command contains information about i.e. whether it is a read/ write request, the length of the receiving/ sending data transaction and an identification of the message.

The next word contains the AXI address. As AXI uses a 32 bit address this fits in the second word. For a read request this is sufficient data. The shell now waits for a response message to be returned. This is explained later.

If the request is for a write transaction, write data is send along the command and address. In this implementation the width of AXI data is 64 bit and hence does not fit in a single word. Therefore, AXI data is down-scaled to a 32 bit Æthereal word. Note that this effects latency as Æthereal processes one word per cycle and AXI data thus takes two cycles.

Furthermore, a write strobe is send with every 64 bits of data. Write strobes enable sparse data transfer on the write data bus. Each write strobe signal corresponds to one byte of the write data bus. When asserted, a write strobe indicates that the corresponding byte lane of the data bus contains valid information to be updated in memory. There is one write strobe for each eight bits of the write data bus [22]. As this thus requires eight bits for the 64 data bits, it does not fit in the command word. Therefore, a separate word is added to transport the write strobes. As mentioned in section 5.2.2 there are other solutions possible, but in order to keep Æthereal independent of AXI this solution is preferred.

As AXI is capable of sending data bursts-based transaction where only the start address is issued, there can be multiple of these data elements per transaction. The number of data elements is derived from the AXI write data channel. Again note that down-scaling takes an extra cycle and also per 64 bits of data an extra cycle is needed for sending the write strobes.

As AXI supports read and write request to be performed in parallel and Æthereal does not allow for this, read and write transactions are arbitrated in a round-robin fashion. Note that this is only done when a read and write request are performed in the same cycle and does not harm compositionality as this only effect the behavior of the transactions of the corresponding task. Arbitration is then performed on a AXI transaction granularity.

*Optimizations*

As the hardware platform supports optimizations for streaming data, writes are made posted. This means that the write response is not provided by the slave party, i.e. the memory controller, but by the network itself. This is taken care of in the target shell. If this would not be implemented, latency of the network is still an issue, because as the ip_1036 does not support write interleaving a next write request would stall until the write response is returned from the memory controller

thus adding twice the latency of the network[1].

AXI supports four types of response signalling all causing different response types, namely normal access, exclusive access, slave error and decode error. The shell only supports the first. With a correct implementation errors do not occur and are used for debugging only. Exclusive access is not supported yet as the shell standard returns a normal write access okay response.

The ip_1036 supports pipelining a maximum of eight read transactions. This is also supported by the target shell and is implemented by using a different *Finite State Machine* (FSM) for request and response messages. Identification of the different messages ensures that different messages are not interchanged.

*Response*

The slave party generates a response message depending on the received request. For both read and write requests a command word is send (see figure 22), indicating i.e. the length and the identification of the read/ write request. The target shell de-sequentializes this message and asserts the correct write response channel signals.

A read response also contains the read data. As the AXI data width is 64 bit, two words are send for the data. AXI supports data burst, thus multiple data elements can be send in one transaction. The amount of AXI data elements is indicated by the AXI read address channel. The target shell up-scales two data elements to one 64 bits AXI data element and asserts the correct signals on the AXI read data channel.

### 6.3.3 *AMBA AXI initiator shell*

The initiator shell de-sequentialized the messages generated by the target shell and asserts the correct channel signals at the slave party. Furthermore, it implements the following features:

- De-sequentialize request messages into commands and their flags, addresses and write data
- Down-scaling 64 bit AXI data to 32 bit Æthereal data for read response
- Up-scaling 32 bit Æthereal data to 64 bit AXI data for write request
- Sequentialize read and write responses in response message
- Word-based valid signalling or transaction-based valid signalling
- De-pipeline read requests

1 A normal write request, like a read request, depends upon a request and a response through the network. Posted writes only depend upon a request

*Request*

As explained in the previous section the target shell sequentializes the AXI channel signals according to a given message format (figure 22). The initiator shell mirrors this operation and ensures that the AXI channels are asserted with the correct signals. This includes up-scaling data to 64 bits for write requests.

*Response*

Response requests of the slave party are sequentialized in messages as shown in figure 22. However, in contrary to the target shell, read data is not allowed to be pipelined. The reason for this is explained in the following paragraph.

*QoS Support*

To derive the worst-case time for a request fetched from the source queue until the response is stored in the destination queue the intervening communication path must provide QoS. This means that it has to guarantee that a transaction is served within a bounded amount of time. As the latency from the source queue to the memory, as well as the latency from the memory to the destination queue is fixed (section 5.4) and thus provides a predictable service curve, the only insurance that has to be provided is that both the queues are able to store that complete transaction when the transaction is started.

The source queue has to be able to store one request message, since the write interleave depth is one and hence a command and address are followed either by the corresponding data (maximally 16 data elements) for a write request, or the command and address of a new request otherwise. The minimum depth of the source queue is thus 50 words (command, address and for a maximum burst of 16, 48 data and write strobe words). Hereby it is ensured that if the filling of the source queue is equal to $2 + 3 * \phi$, where $\phi$ is the number of data elements indicated in the command word, the complete transaction is available in the source queue. For a read request only two words need to be available, the command and the address.

Also, it must be ensured that the destination queue is able to store the complete response message. Note that the minimum depth of this queue is equal to the minimum depth of the source queue, which is 50 words (maximum size of a read response).

If read transactions are allowed to be pipelined this endangers compositionality, since memory arbitration is then no longer implied by the predictable arbiter (it is no longer guaranteed that a read transaction is finished before another transaction is started), giving control to the arbiter in the memory controller and hence making the system unpredictable. Thus to ensure that time-slot boundaries are not exceeded and compositionality is maintained, read requests are de-pipelined.

Note that this does not influence the advantage of pipelining read requests in the target shell, since these requests can be stored in the source queue to prevent the the ARMs from stalling.

Furthermore, at the moment a complete read request (command and address) is available in the source queue and there is enough space in

the destination queue for receiving the response data and the request is granted, another request may be pending filling the queue after the counter, indicating the filling depth of the destination queue, is polled. This may result in a case where a request is granted but another pending request has filled the destination queue such that the response data does not fit in the destination queue anymore, thus endangering compositionality. This could however be solved by implementing a separate counter that takes the required storage space in the destination buffer into account.

QoS support is implemented by a separate module (see figure 20) in the initiator shell that polls both the counter signals (both source and destination queue have a counter) provided by the kernel. If the shell is set to provide transaction-based signalling, a valid signal from the source queue indicating that the queue contains a valid data element (command) is not accepted immediately when the TDMA arbiter grants access. First that command word is snooped from the data bus, without accepting it. In this way it is possible to derive the amount of data elements that are going to be transmitted in case of a write or need to be received in case of a read request, since this information is stored in the command word.

The module then waits until the counter of the source queue indicates that the derived number of data elements are available in the queue. If that is the case, the counter of the destination queue is polled to check whether sufficient room is available in the destination queue to accept the response message. If this is also the case, a transaction valid signal is asserted to indicate that the transaction can safely be processed.

The module then sends a valid signal to the TDMA arbiter. If the TDMA arbiter grants access to the memory (the corresponding time-slot is active and there is enough time left in the time-slot to process the transaction) the transaction is processed.

## 6.4 SLOT-TIME DERIVATION

Note that as the maximum size of a transaction can be determined (an AXI burst contains maximally 16 data elements), minimum slot-times are derived accordingly. The minimum queue depth is equal to the maximum size of a transaction and is 50 words (where a word is 32 bits). Every clock cycle one word is processed by the initiator shell. The current configuration of the memory controller is such that for write request the latency from AXI to memory command is 1 cycle and for read requests there is no latency (table 3). The throughput is 1 cycle per element. This results in a minimum slot size of 51 cycles.

## 6.5 CONFIGURATION

In order to execute an application on the hardware platform some configurations need to be performed. Each ip_1036 needs to multiplex its data to correct output ports, which requires an address map. Furthermore, the connections of the network need to be configures to enable the transfer of data.

### 6.5.1 *Address map*

The ip_1036 can provide a specific feature to support multiprocessor systems. Every processor can have its lower address section, starting from address zero, re-mapped to a different memory area. This is required, because multiple ARM processors can be configured to boot from the same address, while this may not be wished. Also most ARM processors have their respective exception vectors located at address zero, this too will require a re-mapping from ROM to RAM of address zero after booting.

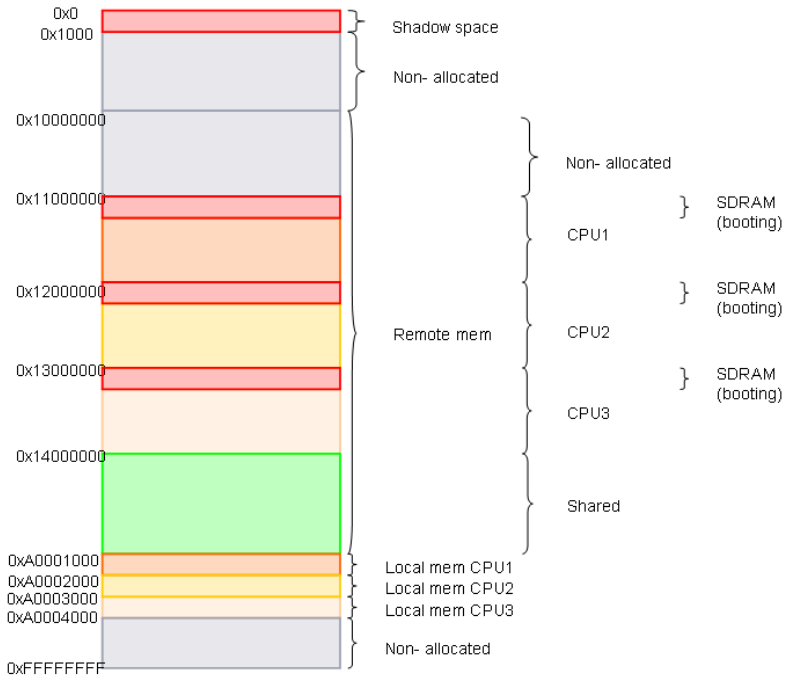


Figure 23. Address map

To avoid breaking the unified memory space, a specific section of the unified memory map is assigned as a *shadow memory section* (see figure 23). This memory section is located from address 0 up to $2^{shadow\_sz} - 1$[2] and is virtual, i.e. no actual memory is present at the shadow address. It can be seen as a copy of a section of unified memory, specific to each master. The size of the shadow memory is the same for all masters, the actual re-mapped locations differ [29].

Based on the address the ip_1036 statically maps data to an output port. In Æthereal the input ports are dynamically mapped to a output port. This is done by reconfiguring connections between ports (see the next section). For this implementation every input port is connected to a fixed output port, and this connection is not changed during execution. Hence, network configuration is only done at startup and the network

2 Where $shadow_{sz}$ represents the highest address bit of the shadow space + 1

is fully connected.

### 6.5.2 *Network configuration*

The Æthereal NoC is used to program itself. This is performed through configuration ports using DTL-MMIO transactions. The NoC can be configured in a distributed fashion (i.e., via multiple configuration ports), or centralized (i.e., via a single port). The designers of the Æthereal opt, when having a small NoC ($\leqslant$ 10 routers) like for this design, for centralized configuration, because it is able to satisfy the needs and has a simpler design and lower cost.

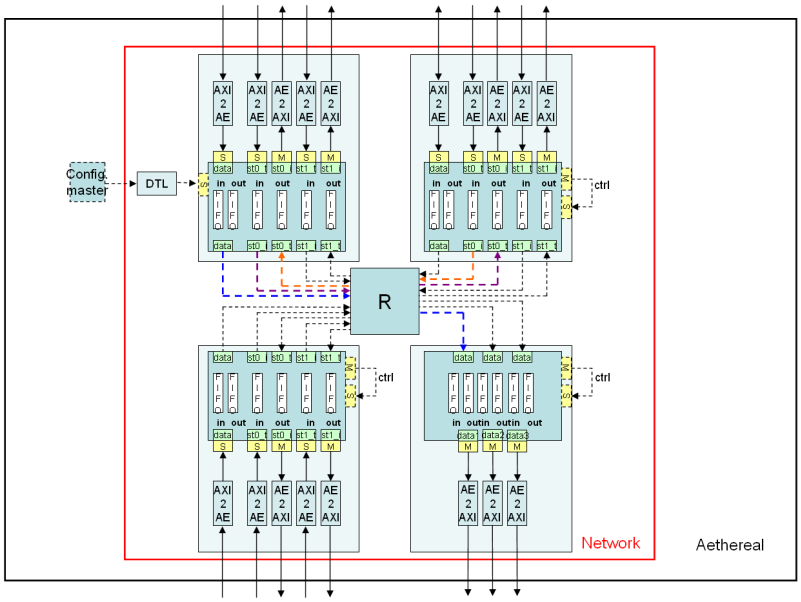


Figure 24. Configuration: NIs are point-to-point connected to the router (not physical wires). Point-to-point connections are set-up by the configuration master.

At the configuration module NI, a configuration shell is introduced, which based on the address configures the local NI or sends configuration messages via the NoC to other NIs. An example of centralized configuration is shown in figure 24, where the left block represents the *configuration master*.

NIs are configured via a configuration port, which offers a memory-mapped view on all control registers in the NIs. This means that the registers in the NI are readable and writable by any master using normal read and write transactions. Configuration is performed using the NoC itself (i.e., there is no separate control interconnect needed for NoC configuration). Consequently, the configuration ports are connected to the NoC like any other slave. For this implementation the configuration master is performed by a centralized exerciser that runs a tcl script. Hence, it is not possible to synthesize the current design.

*Nothing is more practical than a good theory*

*— Ludwig Boltzmann*

# 7 RESULTS AND RECOMMENDATIONS

In order to verify that the discussed methods and techniques provide compositionality to the system, an application is mapped onto the implemented architecture. The application contains three independent jobs each mapped on one of the ARMs. For this implementation all instructions and data of the application are stored in the remote memory. Compositionality of first degree (see table 2) is confirmed by showing that change in the behavior of one of these jobs does not affect the behavior of the other two jobs. This is comparing trace files, which for the two unchanged jobs are verified to be identical for every cycle.

## 7.1 USE-CASE

For the test use-case each ARM is connected to a different interrupt controller. The interrupt controller is standard deployed from an IP database with verification software to confirm correct behavior. The correct behavior of the entire system is thereby implicitly verified. If all verification tests pass, this implies that the master, the interconnect and the slave are functionally correct. The application is thus the verification of the three interrupt controllers.

The software code of each of the three verification jobs is stored at the corresponding address ranges within the address map (see figure 23). This ensures that the three ARMs all have there own address spaces in the shared memory that is not effected by the other ARMs. The shared memory is not used for communication purposes.

## 7.2 RESULTS

In order to prove compositionality two different setups are simulated. For the first setup three different jobs are mapped on the three processors. For the second setup one of the jobs is changed, thereby changing the temporal behavior of that job. The other two jobs are the same as for the first setup. There are no inter-dependencies between jobs, since the goal is to prove first degree compositionality.

Figure 25 is vertically divided into three blocks, one for each CPU. Each block contains four waveform, divided in two sections of two waveforms. The top section of each block represents read requests and the bottom section represents write requests. The top waveform of each section belongs to the first setup, and the bottom to the second setup. For CPU 2 and CPU 3 the application is the same for both setups. CPU 1 has a different job mapped on it for both setups.

The two waveforms are compared to identify whether there is a change in behavior between the two setups. If there is a cycle that

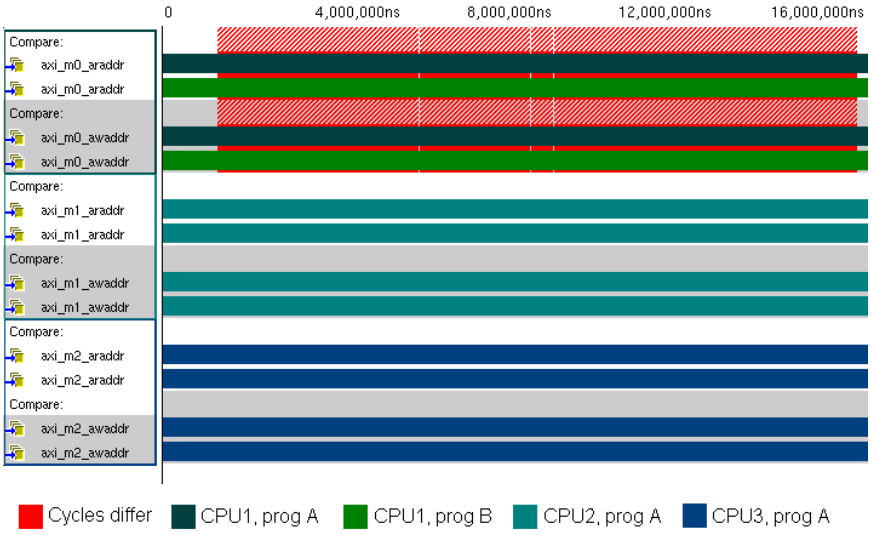


Figure 25. The waveform shows that the change of the application on CPU 1 does not effect the the temporal behavior of CPUs 2 and 3. Red indicates the cycles that differ

differs between the two setups this is indicated by a bar on top of the section. For this use-case this bar shows that the waveforms behave differently for CPU 1 between both setups. This is orthogonal as the trace is observed for two different applications.

However, the waveforms show that the temporal behavior of CPU 2 and CPU 3 are not effected by the change in the job (temporal behavior) of CPU 1. Hence, for this example it is shown that the platform is compositional. Even more, since not a single cycle is different the strictest form of compositionality is confirmed for this use-case.

The use-case shows that virtualization of the hardware platform results in first degree strictest form compositionality (see figure 25). Therefore, it is possible for this system to simulate the behavior of each of the jobs and ensure that changes in one of the jobs does not influence the behavior of the other jobs.

| | ip_1036 (ns) | Æthereal (ns) |
|---|---|---|
| No cache | 5,530,550 | 30,446,430 |
| Instruction cache | 1,848,110 | 14,500,910 |
| Instruction & data cache | 888,760 | 1,946,480 |

Table 4. Influences of cache on the execution-time. First column shows the effects on the reference design, where the second column shows the effects on the compositional design

Next to the compositional system a comparable non-compositional

reference system is implemented and simulated to investigate the increase in run-time due to virtualization. For a fair comparison Æthereal is replaced by a 32 bit AXI interconnect. For this use-case the strictest form compositionality results in an increase of the execution time by five times compared to the reference system (see table 4). The latter uses round-robin arbitration and hence is work-conserving. The efficient use of the memory decreases from 70 procent for the reference system to only 10 procent for the compositional system. Investigation of the trace file shows that the slot utilization of the memory controller for write data is much better than the slot utilization for read data. It appears that latency is not drastically effecting performance if the throughput can be kept high, as for posted write data.

However, if read requests cannot be pipelined (i.e. events) the throughput is effected by the round trip delay of the network. Slot-time at the memory cannot be used by other transactions as arbitration is non-work-conserving and the memory controller is single-threaded, decreasing memory efficiency. This causes a serious performance bottleneck. Introducing instruction and data cache decreases the amount of low-latency request, thereby having a significant impact on execution time (see table 4). To pin-point the precise cause for the overhead in performance cost, analysis is done on the behavior of these low-latency read transactions. Note that for this analysis it is assumed that the clock frequency of all IPs including the network is the same.

## 7.3 COST ANALYSIS

If we look at IP blocks in general they often have a particular communication granularity for which the behavior of the IP is optimal. For example the granularity of SDRAM depends on the access size of the memory (taking banking into account), the granularity if CPU communications may relate to the size of cache lines and accelerators may have a much coarser granularity. Typically for NoCs, the granularity is as fine (fluid) as possible.

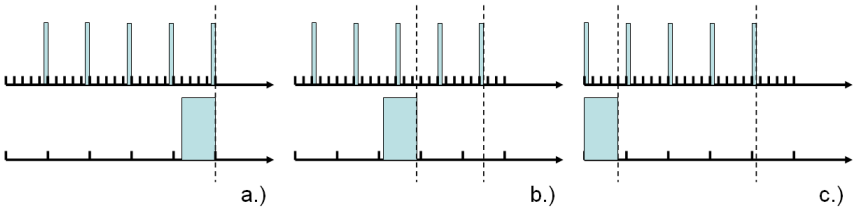


Figure 26. The granularity only effects average and best-case scenarios negatively: a.) Worst-case a small and large granularity perform equally b.) Average-case a large granularity performs better c.) Best-case a large granularity performs even better than average-case

For this use-case the granularity of Æthereal is on flits, whereas the granularity of the ARMs and the memory controller is on AXI transactions. A flit contains three words of 32 bits. A maximum AXI

transaction contains 16 data elements of each 64 bits. Communication granularity only effects average and best-case run-time (see figure 26), worst-case run-time is the same for every granularity. A larger granularity results in better average-case and best-case run-time compared to a small granularity. Therefore, it is advocated to size slots for the largest non-preemptable transaction in the system. However, for NoCs a large granularity is not practical as it requires large buffers and is therefore kept as small as possible. For the memory controller larger granularity improves performance. Due to the difference in granularity between Æthereal and the memory controller latency is effected negatively by the network for average and best-case situations. Latency is crucial for requests where pipelining is not possible, i.e. events.

In order to transfer one AXI request for 16 data elements, the network sequentializes the request in many smaller flits. The latency of the NoC is then effected by the number of revolutions of the slot wheel (depending on the number of connections supported by the NI) to transfer all these flits over the network. The worst-case latency effect is discussed in the upcoming paragraphs. Note that it is assumed that all requests contain 16 data elements (fixed AXI burst size). This is an assumption and certainly not always true, as investigation of the trace file shows (burst lengths vary from one to around six data elements). However, the variability in AXI bursts is another issue and is discussed later on in this report. Also for the first implementation each CPU is connected by one NI, thus requiring five slots in Æthereal (explained later).

In Æthereal every flit that is passed over the network is divided into three words per slot. The first word of the flit is used for the packet header information. The two other words can be used for payload of the message. For packets that can be send in consecutive slots only the first flit contains the header information, the rest of the flits can be completely used for payload. For this first implementation each ARM is connected to one NI. Consequently, the connection to the router is shared by the four ports for streaming data and one port for data to remote memory (see figure 10). This corresponds to the result of the design flow of Æthereal, which has calculated for this use-case that at least five slots are needed to give every connection a time-slot [14]. The granularity of a slot is the size of a flit, hence a complete revolution of the TDMA wheel takes 15 cycles (equation 7.2).

$$5\ \text{slots} * 3\frac{\text{words}}{\text{slot}} * 1\frac{\text{words}}{\text{cycle}} \quad = \quad 15\ \text{cycles} \tag{7.1}$$

Because latency-tolerant request (i.e. posted write requests) do not form the bottleneck in this system, the attention is focussed on requests with low-latency requirements. In this case this are the AXI read requests that cannot be pipelined. This section analyses the worst-case latency for an AXI read request for 16 data elements.

A read request (command and address) has to be available before the beginning of the slot in Æthereal. This means that the request is sequentialized by the NI shell into a command and address. Sequentializing the AXI signals for a read takes two cycles. The request is then send to the FIFO in the NI kernel. It takes one cycle to transfer the first word to

the FIFO. The flit is then scheduled to be send over the network. The flit waits worst-case for all other slots when missing the assigned slot. For the worst-case scenario the request misses the slot with one cycle and has to wait for a complete revolution of the TDMA time wheel, thus adding 14 cycles latency. For this use-case the maximum latency that is added in the NI for one flit (enough to send a packet header, the command and the address that are needed for a read request) is $2 + 1 + 14 = 17$ cycles.

To transport the flit (packet header, command and address) from one NI to another takes two hops (connections between NI and router). The latency added by the router is therefore:

$$3\ words * 1\frac{word}{cycle} * 2\ hops \quad = \quad 6\ cycles \tag{7.2}$$

As a read request takes only one flit, the complete transaction is then available in the source queue of the NI kernel at the memory side. In the shell the command and address of the incoming flit are desequentialized again. This takes two cycles. Thus worst-case a flit arrives in 25 cycles at the output channels of the NI. The AXI request is then scheduled to be send to the memory. The memory is arbitrated with TDMA divided into three slots, each taking 51 cycles. A complete transaction is processed in one such a slot, thus for the worst-case when missing a time-slot the transaction requires 102 cycles to arrive in a corresponding slot again before it can start the transaction. This is when no time is wasted in the missed slot itself. When a request arrives one cycle late, the rest of the slot-time is also unused. Hence, this adds 34 cycles latency for this read request (a read request does not contain the 16 extra write strobe words that are needed by a write request). Furthermore, the memory requires 16 cycles to transfer the 16 data elements. In total it thus takes worst-case $34 + 102 + 16 = 152$ cycles for a read transaction of the memory.

To send the 16 data elements back to the CPU the network requires 33 words (one command word and 32 data words). The network is reserved for one slot per revolution, where each revolution transfers one flit containing two words payload. Hence, in total 17 flits are required. The AXI signals from the memory are sequentialized in the shell. The shell sequentializes the first two words of the read response, namely a command and a data word and transfers these over the network. In worst-case this flit waits for 14 cycles before it is transferred additional to the one cycle that is required by the kernel to store the first word in the FIFO. The rest of the data words are also sequentialized in flits, where between each flit a complete TDMA revolution is performed before a next flit can be send. Before the last flit has arrived at the CPU, the router again adds six cycles latency and desequentializing takes another two cycles.

If we sum up these cycles to total worst-case latency is 457 cycles. Note that 152 latency cycles are caused by the memory and its arbitration. The other 305 cycles are caused by the latency of Æthereal and the arbitration of the different connections in Æthereal. From this is concluded that the large difference in slot granularity of an AXI transaction and a flit forms a big latency issue when pipelining is not possible

(low-latency requests). This effect can be controlled by reducing the number of slots in Æthereal.

### 7.3.1 *Parallel connections*

This design does not contain parallel connections from each of the CPUs to the memory. Each ARM has one NI via which instructions as well as streaming data are transported over the same physical connection. As the NI contains five ports to the router the minimum number of slots needed is five. One of the advantages of NoCs is that they can allow for parallel connections. This is also possible for busses (e.g. multi-layered bus) but in fact this is a combination of n busses and as we have seen this does not scale well for larger designs. If a separate NI connects i.e. the instruction port to the router, this connection is only dedicated for instruction data. Latency is now lower as it the instructions do not have to wait for other slots in Æthereal.

By reducing the number of slots the latency effect of the network decreases towards zero (the latency effect is not completely removed as additional latency is still required for de/ sequentializing data and the transfer of the data between NIs in the NoC). Next to the reduced number of slots, reservation of consecutive slots also decreases the number of packet headers thus improving slot efficiency. The latency caused by the memory thereby becomes the bottleneck in performance as the memory is still shared. In figure 28 a breakdown is shown of the transaction latency for a read request of 16 data elements into network and memory latency.

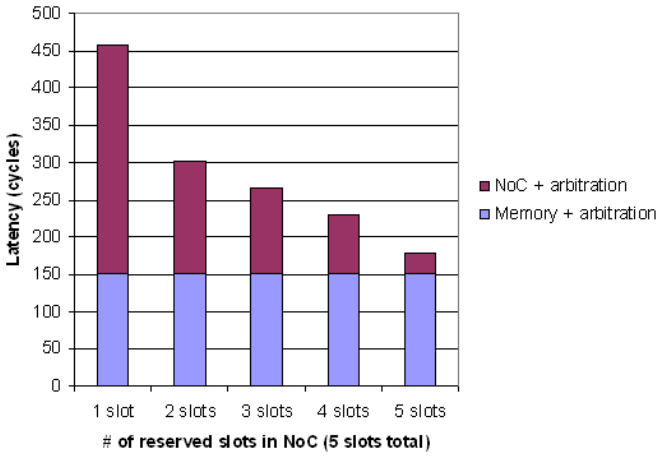


Figure 27. NoC and memory contribution to worst-case read request latency for 16 data elements of 64 bits

To investigate the effect on latency by Æthereal if all CPUs have a dedicated connection to memory, a new test-case is setup where all slots in the network are assigned to only one CPU. Note that the other two CPU's are thereby not able to retrieve data from shared memory

and hence are stalled. Arbitration to shared memory is still divided in three slots, as this is still a contention point. The performance of this test-case increased by 30 procent. Note that this is still almost 4 times slower than the reference system. The reason for this relative low effect is explained in the following section.

### 7.3.2 *Dynamism*

Until now the assumption is made that an AXI burst is always for 16 data elements. However, investigation of the trace file shows that this is far but true for this use-case. Burst lengths vary from one data element to around six data elements. The dynamic behavior of AXI burst cause a new set of problems. A small request of only few data elements may not fill the time-slot for the memory transaction. The rest of the time can then be used for a new request. As for posted writes throughput can be kept high and the CPU is not stalled, a new request at the memory is done almost instantly (the throughput for the network is around one flit every three cycles).

However, read requests that cannot be pipelined stall the CPU. A new request is therefore only done when data of a previous request has arrived at the CPU. The time in between these requests the memory is unused, thereby decreasing efficiency. Hence, the round trip delay of the network effects the efficiency of the memory. Consequently, for the best-case scenario where no latency is added by arbitration (dedicated parallel connections) maximally two non-pipelined read requests can be served in the same slot. This is because if a read request for one data element has arrived at the memory, the memory adds minimally one cycle latency and the two flits of the response require 14 cycles to return. In total this is 15 cycles. A new request then arrives after minimally 26 cycles as a new request takes 11 cycles to arrive at the memory again. Not included are the number of cycles needed by the processor to process the received data. Hence there are maximally 25 cycles left to return the response and send a new request, which does not fit in the time slot of the memory arbiter. Therefore, the next request has to wait for the next assigned time slot.

The 30 procent increase in efficiency between the system with five slots in Æthereal and the system with dedicated connections can be concluded from the fact the for the first setup the best-case situation (two low-latency read requests in one slot) does not always occur, where it does for the system with dedicated connections. However, the efficiency of the memory is still low because of the limited amount of requests in a slot.

The low load of the memory does not advocate the use of non-work-conserving arbitration. For the non-work-conserving arbitration used in the implementation the average-case and the worst-case behavior are not depending on the work load of the system. This is because the bounds on the uncertainties in resource behavior are removed, resulting in the strictest form of compositionality. Therefore, average-case latency can be derived without information about the application, since the percentage of time a job has access to a resource is known at design time.

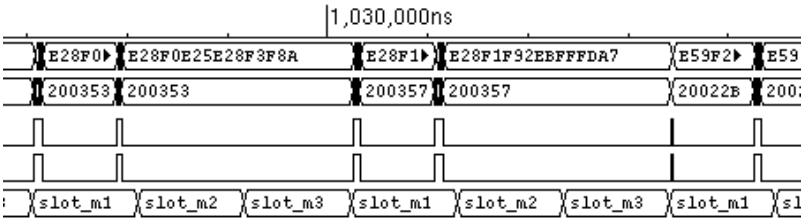


Figure 28. By assigning all slots in Æthereal to one CPU, no scheduling waiting time is added in the network. The figure shows the time-slots for scheduling the memory. Note that for this test case the other two CPUs are stalled, thus do not use there slots. In total six read request are completed in three slots. The unassigned time is mainly due to the round-trip latency of the network. This confirms the low utilization of the memory

Note that this implicitly assumes that the complete time-slot can be used to access the resource, which is not completely true for this implementation. I.e. the difference in length of read and write request causes the slot size of the memory arbitration to be sized for the longer write requests. The variability in AXI bursts may optimize average-case behavior in normal systems, but does not suit well for compositionality. It may not be possible to fill a complete time-slot with consecutive AXI request as these request cannot be distributed over multiple slots. Therefore slot-time is not used efficiency.

## 7.4 RECOMMENDATIONS

The results show that the solution for compositionality provided in this thesis is not optimal for all situations. For instance applications with low-latency requirements do not suit well because of i.e. the latency added due to TDMA arbitration at the memory controller. To improve performance costs there are several recommendations.

As the low-latency requests result in a low memory load, performance results are expected to be better in combination with work-conserving arbitration. For maximum load both arbitration techniques behave similar. For work-conserving arbitration average and worst-case behavior diverge for low memory load. This is because the probability that there is unused slot-time available in a low load system is bigger than for high load systems. The unused slot-time between a low-latency request and a new request can then be used by other jobs. The variance in the bounds on the temporal behavior is not completely removed anymore as it now is also possible to use time-slots unutilized by other jobs. When a job does not present data at the start of its slot, the access to shared memory is backlogged for a bounded period of time. This can maximally be a complete revolution of the TDMA wheel, depending whether other jobs have data that they presented in the time the slot was not used. Hereafter a fixed service rate is provided, as the job has data present at the beginning of the next slot. Thus the strictest form

of compositionality is not achieved, but one can still derive worst-case inter-dependencies.

The average-case system behavior is now dependent on the load of the system. This makes it difficult to derive average-case behavior. However, since the average behavior is better for systems with lower load it is expected to give performance benefits for these systems. The designer has to make the decision whether it is sufficient to have bounded uncertainties (derive worst-case behavior) at the benefit of better performance for low load systems. As discussed earlier a related issue is that is now not longer possible to show compositionality per cycle. More refined techniques are required to show that the property of compositionality is obtained. For this reason we opt for strictest form compositionality.

Slots can also be sized for average-case behavior. For example, if on average burst lengths contain four data elements, time-slots can be dimensioned for this situation. A problem occurs when a transaction exceeds this limit. The transaction then is stalled until the next slot. Conventional memory controllers do not allow for this as they are single-threaded. This then stalls all other CPUs until the transaction is completed. In this case no hard real-time guarantees can be provided anymore.

There are a couple of solutions for this. First transactions may be split into multiple smaller transactions before they are presented to the memory controller. This can be achieved by adding a new command and address for a subset of data elements. This requires extra bandwidth for the command and address and is not as efficient as requests for larger bursts. However, because time-slots itself are then used more efficient this might result in an increased overal performance. This is not confirmed in this thesis. Note that the data elements still have to be presented in-order to the memory. In this way hard real-time guarantees can still be provided. Another solution can be multi-threaded slaves. If slaves are able to accept multiple commands and are able to give service guarantees compositionality can still be achieved and hard real-time guarantees can be provided.

The discussed issues all relate to the use of the AXI protocol. From this can be concluded that AXI is not the optimal solution for achieving compositionality. The variability and the size of the transactions negatively effect performance results.

What is not considered until now is one of the advantages of NoCs, which is the potential to run on higher clock frequencies. An increase in the clock frequency by a factor two results in half the latency of the network. This is reasonable as the ip_1036 operates around 150 Megahertz where Æthereal could operate at 500 megahertz. Unfortunately, due to the current implementation of the NI clock domain crossing is not possible in the network.

### 7.4.1 *Related issues*

In chapter 4 it is advocated that the interconnect provides flexibility with respect to supported protocols. In this report this is addressed by introducing shells to convert from one protocol (i.e. AXI) to another (i.e.

native Æthereal). As all components used in the design are based on the AXI protocol, the network only has to be AXI compliant. However, this is an optimized situation. In practise AXI components may be connected to components based on DTL or other protocols.

For $n$ different protocols, which are allowed to be connected to $m$ other protocols, this requires $n$ x $m$ different shells. This is because every protocol needs a message format that must be understood by the shell on the other side of the network.

Optimally one would need only one target and initiator shell that is compliant to the complete set of protocols. This means that there is one unique message format that includes the superset of all protocols and can be used to sequentialize and de-sequentialize all possible protocol conversions. This issue is addressed in [4]. Note that for every new protocol that is introduced the message format has to be adopted. Also, this includes overhead of bandwidth for the protocols that need less information (command and flags).

*The most important discoveries will provide answers to questions that we do not yet know how to ask and will concern objects we have not yet imagined*

*— John N. Bahcall*

# 8 CONCLUSIONS

In this thesis we focus on the strictest form of compositionality, where jobs do not effect each others behavior at a cycle-true level. Therefore hardware virtualization is applied to a platform implemented in RTL. This thesis shows by simulation that virtualization of the hardware platform results in the strictest form of compositionality. From the analysis of the simulation results and the comparison with a non-compositional reference design several conclusion are formulated.

Current day system level architectures consist of IP blocks (processors, memories, peripherals, etc.) that communicate via an on-chip network. IP blocks operate at different levels of communication granularity. For example, the granularity of SDRAM depends on the access size of the memory (taking banking into account), the granularity of processor communication may relate to the size of cache lines and accelerators (e.g. FFT) may have a much coarser granularity. Typically for NoCs, the concept of a flit is important and the granularity is as fine (fluid) as possible.

A critical parameter is the round-trip latency in case of an event like a cache miss. The latency of a communication path built from a chain of resources depends on the reservations and transfer time of each shared resource along this path. The latency is thus coupled to the scheduling waiting time and transfer time of all the resources in the chain. Two situations are possible depending on the alignment of the scheduling of resources. If there is no alignment the latency is affected by the scheduling waiting time and the transfer time of all preceding resources in the chain. If there is alignment the latency is only affected by the transfer time of all preceding resources in the chain. The impact of the scheduling waiting time is limited to scheduling only the first shared resource of the complete chain.

An important question is: "What is the impact of a NoC on the round-trip latency?" At one hand NoCs give more parallelism, less sharing, less conflicts and consequently reduces the scheduling waiting time. The benefits of a NoC from a latency point of view are most obvious for systems with many conflict [4] and multiple scratchpad memories. Furthermore, by reserving multiple slots for a job on one connection the latency of the NoC can be reduced and the round-trip bottleneck shift towards the scheduling waiting time (arbitration) of the memory. By reserving all slots (dedicated connection) the influence on latency by the NoC is limited to the cycles required for the data transfer and memory arbitration is a true bottleneck.

Not only do we have different communication granularities between resources, protocols may also allow for dynamic burst lengths. The AXI

burst length is not fixed but varies between one and 16. The dynamism of AXI transactions has a large impact on performance. In AXI no component can terminate a burst early to reduce the number of data transfers [22]. Since transactions cannot be interrupted once they are accepted by the memory, time-slots are sized for worst-case situations. For small AXI requests that cannot be pipelined, such as instruction fetches, this results in low slot utilization. This is worsened by the additional round-trip latency of the network that is added before a new request can be done. During this transfer time the time-slot and thus the processor is unused. For this thesis we accept these costs for achieving the strictest form of compositionality. Optimizations can be implemented by relaxing the constraints on compositionality, i.e. assigning unused slot-time to other jobs. Performance is expected to be better, but the strictest form of compositionality is then not obtained. In this case it is no longer possible to look only to cycle counts, but more refined techniques are needed [24].

These issues show that AXI is not an optimal protocol for obtaining compositionality. Fixed burst length and identical behavior for read and write request is advocated to more efficiently use reserved slot-time. The latency is effected by the size of the biggest non-preemptable transaction in the system [34]. The granularity of transactions, from a compositional perspective, has to be sized as small as possible to increase slot utilization and decrease the latency effect caused by memory arbitration. A solution is needed for transactions that exceed slot-times. In this report two solutions are provided. First AXI transactions can be split into multiple smaller transactions before it is accepted by a component. Second, multi-threaded slaves may be implemented that do allow for interrupting transactions. Next to memory arbitration also the arbitration of the NoC is an issue. Parallel and dedicated connections are possible to minimize the latency effect of the network to only transfer latency.

To obtain strictest form compositionality, especially in combination with AXI, the cost for unused slot-time has to be considered. This thesis shows that latency-tolerant data streams (i.e. produced by streaming applications) are well suited for the obtained compositional system. When latency is hidden by buffers and throughput is kept high, e.g. by writing instead of reading data, then reservations are used efficiently. However, for low-latency requests the granularity and dynamism of transactions are an issue. To suit compositionality communication protocols must not be designed for the behavior of an individual IP, but for overal compositional system behavior.

*We are at the very beginning of time for the human race.*
*It is not unreasonable that we grapple with problems.*
*But there are tens of thousands of years in the future.*
*Our responsibility is to do what we can, learn what we*
*can, improve the solutions, and pass them on.*

*— Richard Feynman*

# 9 FUTURE WORK

In this thesis several issues are observed that influence compositionality with respect to i.e. its property and the (performance) cost. Most of these issues are not further addressed and are considered as future work. These issues are listed in the current chapter.

- *Hybrid platform*, the idea is to distinguish low-latency off-chip communication (instructions and load/ store operations) and latency-tolerant on-chip communication (streaming data) by using two separate communication paths. Low-latency communication is thereby transferred over a normal AXI interconnect and latency-tolerant communication via the network, minimizing the round trip delay of low-latency communication. Note that arbitration to remote memory still needs to provide compositionality
- *Clock domain crossing*, Currently the implementation of Æthereal does not allow for clock domain crossing. Either this must be implemented or other techniques have to be applied to allow for this. The increase of the clock frequency of the network decreases the latency of the network
- *Protocol superset*, as discussed the network should provide flexibility with respect to the compliance of protocols. Heterogeneousness of connected components ask for different protocol types to be bridged to allow for inter-communication. The current implementation asks for $n^2$ shells for all possibilities of connecting these $n$ shells together, which is not preferred. When a new protocol has to be supported another $n + 1$ shells have to be implemented. A solution is to introduce a superset of all protocols, as proposed in [4]. This means that only two shells need to be implemented to bridge between all possible protocols as discussed in 7.4.1.

# INDEX

# BIBLIOGRAPHY


[1] S. Stuijk T. Basten A.J.M. Moonen M.J.G. Bekooij B.D.Theelen M.R. Mousavi A.H. Ghamarian, M.C.W. Geilen. Throughput analysis of synchronous data flow graphs. (Cited on page 18.)

[2] Om Prakash Gangwal Ramanathan Sethureman Natalino Busá Kees Goossens Rafael Peset Llopis Paul Lippens André Nieuwland, Jeffrey Kang. *C-HEAP: A Hetrogeneous Multi-Processor Architecture Template and Scalable and Flexible Protocol for the Design of Embedded Signal Processing Systems*. Kluwer Academic Publishers, 2002. (Cited on pages 16 and 27.)

[3] Kees Goossens Edwin Rijpkema Paul Wielage Andrei Râdulescu, John Dielissen. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration. (Cited on pages 23, 30, 32, 33, 34, and 35.)

[4] Arteris. A comparison of network-on-chip and busses. 2005. (Cited on pages 23, 63, 64, and 66.)

[5] L. Benini and G. De Micheli. Networks on chip: A new soc paradigm. *IEEE Computer*, 35(1):p. 70–80, 2002. (Cited on pages 8 and 23.)

[6] André Ivanov Res Saleh Cristian Grecu, Partha Pratim Pande. Structured interconnect architecture: A solution for the non-scalability of bus-based socs. (Cited on pages 8 and 23.)

[7] W.J. Dally and B. Towles. Route packets, not wires: On-chip interconnect networks. *Proceedings of DAC*, 2001. (Cited on page 8.)

[8] A. Râdulescu J. Dielissen J. van Meerbergen P. Wielage E. Waterlander E. Rijpkema, K.G.W. Goossens. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. (Cited on pages 25, 30, 31, and 32.)

[9] D.J. Culler et al. *Parallel Computer Architecture: A Hardware/ Software approach*. Morgan Kaufmann Publishers, 1999. (Cited on page 10.)

[10] The International Roadmap for Semiconductors. Design. 2005. (Cited on pages 1, 2, 4, 13, and 17.)

[11] The International Roadmap for Semiconductors. System drivers. 2005. (Cited on pages 8 and 10.)

[12] The International Roadmap for Semiconductors. Yield enhancement. 2005. (Cited on page 9.)

[13] Björn Nilsson Kees Goossens Frits Steenhof, Harry Duque. Network on chips for high-end consumer-electronics tv system architectures. (Cited on pages 23 and 24.)

[14] Andreas Hansson, Kees Goossens, and Andrei 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, September 2005. ISBN 1-59593-161-9. doi: http://doi.acm.org/10.1145/1084834.1084857. (Cited on page 57.)

[15] Marco Bekooij Jan-Willem van den Brand. A compositional multiprocessor system with caches. (Cited on pages 13, 15, and 37.)

[16] Marco Bekooij Jan Willem van den Brand. A software cache coherent and memory consistant multiprocessor system. (Cited on pages 14 and 27.)

[17] Kees Goossens Edwin Rijpkema John Dielissen, Andrei Râdulescu. Concepts and implementation of the philips network-on-chip. (Cited on page 31.)

[18] Srimat Chakradhar Jörg Henkel, Wayne Wolf. On-chip networks: A scalable, communication-centric embedded system design paradigm. *Proceedings of IEEE, 17th International Conference on VLSI Design*, 2004. (Cited on pages 6, 8, and 23.)

[19] Andrei Râdulescu Kees Goossens, John Dielissen. æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design & Test of Computers*, pages p. 414–421, September-October 2005. (Cited on pages 23, 30, 31, and 32.)

[20] Hermann Kopetz. *Real-Time Systems — Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachuttes 02061, 1997. (Cited on page 19.)

[21] D. Bertozzi L. Benini. Network-on-chip architectures and design methods. *Proceedings of IEEE, Comput. Digit. Tech.*, 152(2), March 2005. (Cited on page 23.)

[22] ARM Limited. *AMBA AXI Protocol, version 1.0*. ARM, http://www.arm.com, 2003-2004. (Cited on pages 36, 41, 43, 45, 46, 48, and 65.)

[23] ARM Limited. *Technical Reference Manual*. ARM, http://www.arm.com, 2004-2006. (Cited on pages 36 and 39.)

[24] Peter Poplavko Bart Mesman Milan Pastrnak Jef van Meerbergen Marco Bekooij, Orlando Moreira. Predictable embedded multiprocessor system design. (Cited on pages 18, 21, and 65.)

[25] A. Kumar S. Ellervee P. Oberg J. Olsson T. Nilsson P. Lindqvist D. Tenhunen H. Meincke, T. Hemani. Globally asynchronous locally synchronous architecture for largehigh-performance asics. *Proceedings of IEEE, Circuits and Systems*, 2, 1999. (Cited on page 8.)

[26] Kevin W. Rudd Michael J. Flynn, Patrick Hung. Deep submicron microprocessor design issues. *IEEE Micro*, 1999. (Cited on page 7.)

[27] Sani R. Nassif. Design for variability in dsm technologies. *First International Symposium on Quality of Electronic Design*, page p. 451, 2000. (Cited on page 9.)

[28] Kees Goossens Santiago González Pestana Edwin Rijpkema Om Prakash Gangwal, Andrei Rădulescu. Building predictable systems on chip: an analysis of guarenteed communication in the æthereal network on chip. (Cited on page 25.)

[29] NXP Semiconductors. *AXI Interconnect (ip_1036)*. NXP, 2005. (Cited on pages 38, 39, 40, and 52.)

[30] NXP Semiconductors. *AXI embedded SRAM/ ROM controller (ip_2114)*. NXP, 2005. (Cited on page 36.)

[31] Juha-Pekka Soininen Martti Forsell Mikael Millberg Johny Öberg Kari Tiensyrjä Ahmed Hemani Shashi Kumar, Axel Jantsch. A network-on-chip architecture and design methodology. *Proceedings of IEEE, Computer Society Annual Symposium on VLSI*, 2002. (Cited on page 23.)

[32] A. Singh. An introduction to virtualization. 2005. (Cited on page 20.)

[33] Jef van Meerbergen. Networks on a Chip: A communication-centric approach to platform-based design. 2006. (Cited on pages 7, 8, 9, 12, 23, and 24.)

[34] Hui Zhang. Service disiplines for guaranteed performance service in packet-switching networks. *Proceedings of IEEE*, 83(10):p. 1374–1396, October 1995. (Cited on pages 19, 42, and 65.)

## COLOPHON

This thesis was typeset with LaTeX $2_\varepsilon$ using Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL* were used). The listings are typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)

The typographic style was inspired by Robert Bringhurst's genius as presented in *The Elements of Typographic Style* (Version 2.5, Hartley & Marks, 2002).

*Final Version* as of January 9, 2007 at 14:48.

# DECLARATION

Put your declaration here.

*Eindhoven, Jan 2007*

Marc van Hintum