Programming of the T-CREST real-time multi-processor platform

Rasmus Bo Sørensen

Kongens Lyngby 2013
IMM-MSc-2013-5

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-MSc-2013-5

Abstract (English)

The goal of this thesis is to integrate the T-CREST components into a tool chain for programming a multi-processor real-time platform. We present our view of the T-CREST tool chain, and we implement an initial tool chain, restricted by the current state of the T-CREST project. The T-CREST project is an ongoing research project supported by the European Union's 7th Framework Programme, aiming to develop a homogeneous time-predictable multi-processor platform. To integrate the components into a common tool chain, we define the interfaces between the components of the tool chain, and extend the components to implement the specified interfaces. With the intuition gain from integrating the components we propose extensions to improve performance or reduce cost.

Resumé (Danish)

Målet for denne afhandling er at integrere T-CREST komponenterne ind i en værktøjskæde til programmering af en multi-processor sandtids platform. Vi præsenterer vores blik på T-CREST værktøjskæden, og vi implementerer en foreløbig værktøjskæde, begrænset af den nuværende tilstand af T-CREST projektet. T-CREST projektet er et igangværende forsknings projekt støttet af den Europæiske Unions 7. rammeprogram, hvis mål er at udvikle en homogen tidsforudsigelig multi-processor platform. For at integrere komponenterne ind i en fælles værktøjskæde, definerer vi grænsefladerne mellem komponenterne i værktøjskæden. Vi udvider komponenterne til at implementere de specificerede grænseflader. Med udgangspunkt i den intuition vi har vundet ved at integrere komponenterne, foreslår vi udvidelser for at forbedre ydeevne eller reducere omkostningerne.

Preface

This thesis was carried out at the Department of Informatics and Mathematical Modelling, at the Technical University of Denmark, in fulfillment of the requirements for acquiring an M.Sc. (Hons.) in Informatics.

During my master studies I was enrolled in the Honors program, at the Technical University of Denmark. The objective of the Honors program is to familiarize the student with research projects on an international level. During the main part of my studies, I have been participating in the early stages of the time-predictable multi-core architecture for embedded systems project (T-CREST), supported by the European Union's 7th Framework Programme. I have been taking part in the weekly project meetings, and most of the projects I have worked on during my master studies have been related to the T-CREST project. This thesis deals with the programming of the T-CREST platform. As a whole the thesis presents a first attempt at developing a coherent tool chain for software programming and hardware configuration of the T-CREST platform. The tool chain draws upon, and extends, several of the projects I have worked on during my master studies.

This thesis should be seen as a stand alone report on the early and first iteration of the tool chain for the T-CREST platform. The T-CREST processor and the T-CREST worst-case execution time (WCET) compiler are under development, and are not yet stable for integration. These components in the tool chain are therefore replaced with a stable processor and compiler. This compiler is not optimizing for WCET, and the thesis will therefore not go into details with WCET-aware compilation. I have concentrated on integrating the tools into the tool chain, to enable the developers of the T-CREST platform to test various ideas.

My work for this thesis has been carried out simultaneously with the first iteration of the T-CREST project. The uncertainty of when and which hardware components will become available, has proved an additional unforeseen challenge. I spent the first month of my thesis working on the S4NoC platform, before the T-CREST NoC platform was available. The work I did on the S4NoC platform is published in [1]. The S4NoC is only mentioned briefly in the thesis.

Lyngby, 07-December-2012

Rasmus Bo Sørensen

Acknowledgements

I would like to thank my supervisor Jens Sparsø for all his great input to my work and my report, and my co-supervisor Martin Schoeberl for his comments and his advice on the JOP processor. Also I would like to thank the rest of the T-CREST members at the Technical University of Denmark for the good discussions we have had about the T-CREST project. Finally I would like to thank Mark Ruvald Pedersen, Lars Bo Sørensen, Tabita Niemann Kristensen and Madava Dilshan Vithanage for their sparring.

Contents

Chapter 1

Introduction

This thesis is concerned with programming the T-CREST[2] real-time multi-processor platform.

The T-CREST research project is supported by the European Union and the goal of the T-CREST project is to develop a real-time multi-processor hardware platform in which all components (processors, interconnection network, and compiler) are designed to facilitate a predictable worst case execution time of the application executing on the platform. The T-CREST project creates a novel hard real-time multi-processor platform.

To program the T-CREST platform four tasks need to be performed in order:

1. Creation of a static schedule for the time-predictable interconnect.
2. Compilation of the source code.
3. Calculation of the worst-case execution time (WCET) of the application,
4. Configuration of the hardware platform.

The work of this thesis is carried out in close interaction with the T-CREST project. As we often need to describe details of the T-CREST work, it can be

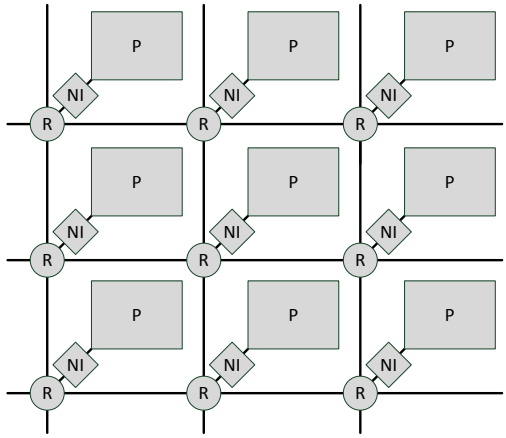


Figure 1.1: Conceptual view of the homogeneous multi-processor T-CREST platform with the three components, a processor (P), a network interface (NI) and a router (R).

confusing to the reader to distinguish our work from the work of T-CREST. We will refer to the work done in this thesis as "our *something*" or "we have *done*". The work of the T-CREST project will be referred to as "the T-CREST *component name*". When referring to this thesis we mean both the report and the work behind it.

In this thesis we integrate the T-CREST tools into a coherent tool chain. Our goal is to provide a tool chain that allows developers to investigate architectural features and flaws in the system. Our tool chain can help the developer to gain insight into the challenges of developing the T-CREST platform, and to improve the platform. Our tool chain will be a modular design, enabling individual testing of the modules and testing of the evolving T-CREST platform. Even though our tool chain is not the final T-CREST tool chain it can help avoid flaws in the final T-CREST platform. To support portability of applications between hardware revisions, a message passing interface (MPI) is needed.

A conceptual view of the homogeneous T-CREST hardware platform is shown in Figure 1.1; A regular grid of identical processors connected by a network-on-chip. The interconnect in the T-CREST platform is statically scheduled to achieve time-predictability. The static schedule is created by a scheduler at compile time. Each processor has a local memory to support message passing between processors. An application calls the MPI when it needs to communicate to another processor.

The T-CREST platform was under development during the work of this thesis, so the requirement to align our work with the availability of the T-CREST components, had an impact on the topics to which we have contributed to. The following points describe the decisions we took to limit the scope of our work in this thesis.

Allocation and mapping The bandwidth allocation and hardware mapping of an application could be found doing static analysis on the source code. In this thesis we assume the bandwidth allocation and hardware mapping for an application is supplied to the tool chain along with the application source code. The bandwidth allocation depends on the hardware mapping.

Code generation The T-CREST processor Patmos[3] and its compiler is at the time of writing still unstable. In this project we use the stable JOP[4] processor with a compiler. As the processor we use is not the final T-CREST processor, we will in this thesis not be concerned with code generation for this processor. The final compiler should optimize worst-case paths and not average case paths as regular compilers do.

Worst-case execution time analysis Analyzing the worst-case execution time (WCET) of an application is very dependent of the processor architecture. In this thesis we do not integrate WCET analysis in the tool chain, but we will make the tool chain ready for it.

Evaluation The tool chain should enable the designers of the T-CREST tools and the hardware platform to evaluate them. It is difficult to evaluate each component in the tool chain without the whole tool chain. We create a modular tool chain where each module can be evaluated and optimized while decoupled from the other components. In this thesis we concentrate on functional evaluation.

Contributions In this thesis we have worked in three main areas of research; A time-predictable multi-processor platform, a time division multiplexing scheduler for routing traffic statically and a message passing interface. The contributions of this thesis are described in the following bullet points, along with an indication of in which chapter the given contribution is described.

- Proposal of interfaces between the T-CREST tools. [**Chapter 3**]
- An implementation and publication of the minimalistic time-predictable S4NoC[1] platform. [**Chapter 4, Appendix A**]
- Integration of the T-CREST NoC platform and the JOP processor. [**Chapter 4**]

- A theoretical comparison of the demand for storage bits in source routing and in distributed routing. [**Chapter 4**]
- Proposals of extensions to the T-CREST NoC platform. [**Chapter 4**]
- Extension of the TDM scheduler to integrate it into our tool chain. [**Chapter 5**]
- Proposal of an extension to the TDM scheduler reducing the worst-case latency in the static schedule. [**Chapter 5**]
- Implementation of an MPI for the T-CREST platform in Java. [**Chapter 6**]
- Proposal of improvements to the MPI to reduce the buffering space. [**Chapter 6**]
- The first working tool chain for programming our homogeneous multiprocessor platform. [**Chapter 7**]

This thesis consists of: An explanation of the main terms and concepts related to the T-CREST platform in Chapter 2. The tool chain and its components are outlined in Chapter 3. In Chapter 4 we present two hardware platforms, the S4NoC platform and the T-CREST Network-on-Chip platform and discuss improvements to the latter. In Chapter 5, a time division multiplexing (TDM) scheduler is presented and the implementations of its interaction with the WCET-aware compiler is described. Chapter 6 contains a discussion of which communication primitives to implement in the MPI library and a description of the implementation. A test of our tool chain and hardware platform is presented in Chapter 7. A conclusion of the project is given in Chapter 8. The related work is presented in the beginning of the chapters it relates to. The source code we have written or changed is shown in the appendices, and in the beginning of each appendix we have written a short explanation of what we have done. The source files are also available online at `http://rbscloud.dk/sourcecode.zip`. We reference the appendices from where it is relevant.

Chapter 2

Background

In this chapter we introduce the main topics of our project: real-time systems, network-on-chip and message passing.

2.1 Real-time systems

There are basically two different kinds of real-time systems; soft real-time and hard real-time. Soft real-time systems may miss a deadline once in a while and is typically applied in non-safety critical systems, such as TV set-top boxes or other streaming applications. Hard real-time requires all timing requirements to be met at all times, and is typically applied in safety-critical systems, such as aviation and train-control systems. The systems addressed in this thesis belong to the hard real-time category. Implementing hardware for real-time systems requires that the hardware is time-predictable and analyzable.

In Figure 2.1 we show how the different run times of a program relate to each other on the time axis. Due to different inputs to the program, the execution time can vary. Also a varying system state before and during the execution can vary the execution time of the program. A varying system state can be the state of the caches and the state of other programs running in parallel. The

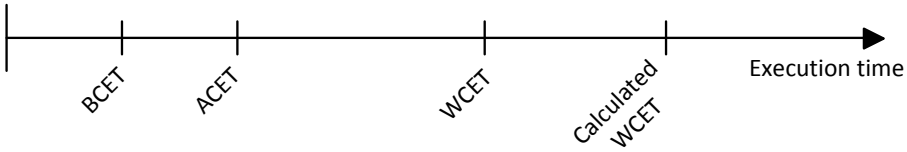


Figure 2.1: Relating the best-case execution time (BCET), the avarage-case execution time (ACET), the worst-case execution time (WCET)and the calculated WCET.

shortest possible execution time is called the best-case execution time (BCET), which is generally not interesting in any kind of computer systems. The average execution time when the program is executed multiple times is called the average-case execution time (ACET), which in general purpose systems is regarded as the performance of a program. The longest possible execution time of the program is called the worst-case execution time (WCET). The WCET is reached when the system receives the worst-case input in the worst-case system state. The performance of a real-time system is equal to the WCET of the given application.

The WCET is found by analyzing the application together with the hardware platform. To find the actual WCET all possible inputs and systems states must be analyzed. An exhaustive analysis is usually not feasible, and in that case a pessimistic estimate of the WCET can be calculated. Depending on the complexity of the systems, the calculated WCET might be far from the actual WCET. The gap between the calculated WCET and the actual WCET can be minimized by using a more accurate model or by making the hardware easier to predict. A more accurate model results in a more complicated WCET analysis. The calculated WCET is regarded as the system performance in real-time systems. More accurate WCET models may result in tighter and lower WCET bounds, but will increase the complexity of the calculation. In parallel real-time systems the WCET analysis is known to be difficult, maybe even impossible. The T-CREST project addresses analysability in parallel real-time systems. The hardware for real-time systems must be deterministic. If the hardware is non-deterministic the analysis must always rely on worst-case latencies. The worst-case in non-deterministic systems might not even be bounded making the analysis impossible.

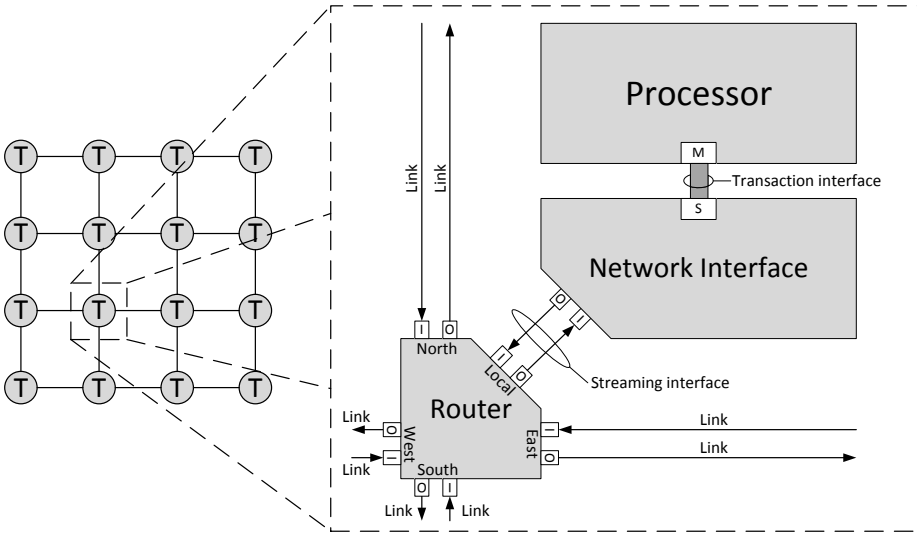


Figure 2.2: The basic Network-on-Chip component a tile (T) contains: A router, a network interface and a processor. The interface between the processor and the network interface is a transaction based master slave interface, and the interface between the network interface and the router is a streaming interface.

2.2 Network-on-Chip

A Network-on-Chip (NoC) is a type of interconnect supporting many communicating nodes. The basic component of a multi-processor platform with a NoC interconnect is illustrated in Figure 2.2. There are two main types of components in a NoC: routers and network interfaces. The processing cores connected to a NoC are each connected through a network interface to the network of routers. In this structured design we group a processor, a network interface and a router into a Tile. The connections between routers are called links. To ease the understanding when describing the ports of a router the ports are usually named after the corners of the world. So the north port of one router is connected to the south port of the router "above".

These routers can be connected in many different topologies. In this thesis we will only concentrate on grid topologies, such as torus, bidirectional-torus (bitorus) and mesh shown in Figure 2.3. In network-on-chip there are no general restrictions on the topology of the network of routers. The network of routers consists of the routers and the links between them. The links are just wires, but as the wires can be very long they can infer a considerable latency in the path.

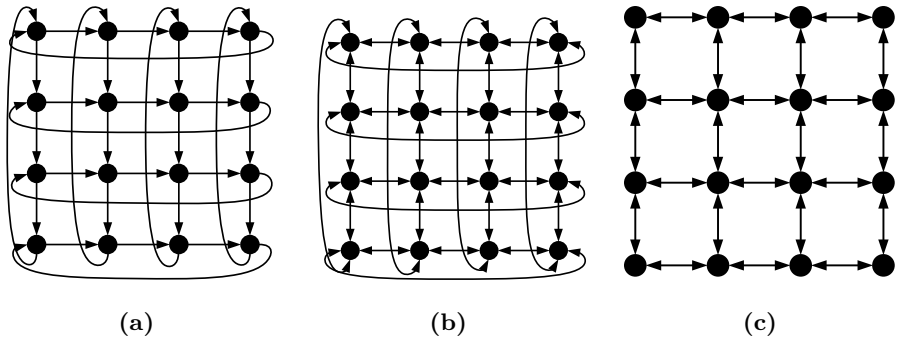


Figure 2.3: The grid topologies relevant to this thesis: **(a)** a torus network, **(b)** a bidirectional-torus network and **(c)** a mesh network.

To increase the clock frequency the links can be pipelined, which increases the amount of buffering in the network. When packets are sent through the network they are chopped into smaller pieces. The smallest amount of data that the network enforces flow control on is called a flit and each flit can be divided into smaller chunks called phits. The phits are the smallest physical data units transmitted over a link, usually equal to the link width.

The basic functionality of the router is very simple. When the router receives a flit on an input port, the router decides to which output port it is sent. Implementing the logic to make routing decisions dynamically can be a complex problem, because the hardware has to ensure that no deadlocks can happen, and that all packets are routed to their destination. Dynamic routing can be implemented in many different ways. In the T-CREST project all routing decisions are made statically. This simplifies the hardware as well as the WCET analysis, because the latency is known in advance. The router implementation scales quite well, as the number of ports stay constant when the size of the whole systems grows.

The basic functionality of the NI, is to convert the transactional requests from a processing core, to the streaming interface of the network. The detailed functionality of the NI depends very much on the routing scheme in the network. The scalability issues of Network-on-Chip are most dominant in the NI. This is where the hardware needs to consider all cores that it is communicating with.

2.2.1 Static routing in real-time Network-on-Chip

Static routing is applied and enforced in such a way that the communication behavior of one processor cannot affect the communication of other processors. Making the scheduler responsible of avoiding colliding flits, allows the hardware to be very simple and efficient. Deciding the routes statically is done by dividing access to the transmission medium in time. This approach is called time division multiplexing (TDM). Only one communication channel must be scheduled on a link in a given time slot. The scheduling of communication channels in the network is done at compile time, before the WCET analysis.

If the communication behavior of one processor can affect the communication behavior of another processor, the WCET will increase dramatically. If for example a real-time system runs on a general purpose platform, the communication between one pair of communicating processors can interfere with the communication of another pair of communication processors. In this case the WCET analysis will have to assume the worst-case of interfering communication, this will increase the WCET estimate by orders of magnitude.

2.2.2 Source routing

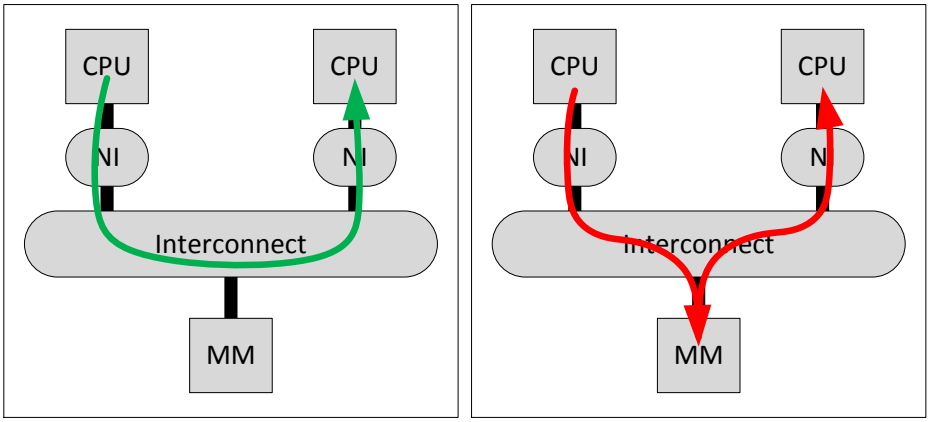
The static route of a flit is stored in the flit header. The route in the flit header is read on its way through the network. The sending NI needs to store a route for each time slot and the destination ID of that route. When a flit reaches a router, the router reads the header to determine whereto the incoming flit should be routed. The receiving NI can see the origin of a flit in its header.

2.2.3 Distributed routing

In distributed routing the static routing information is distributed to the routers where it is needed. This implies that the flits does not need a header, which increases the bandwidth. The sending NI stores an entry table with the destination ID of the flit that is allowed to enter the network in the given time slot. The router stores how its input ports are connected to its output ports in each time slot. The receiving NI stores an exit table with the source ID of the flits which can arrive from the network in the given time slot.

2.3 Message passing

Message passing is a way of communicating between tasks, as opposed to shared memory communication. The conceptual difference of communicating with message passing and shared memory in multi-processor systems is illustrated in Figure 2.4.



(a) Message passing **(b)** Shared memory

Figure 2.4: Conceptual illustration of message passing and shared memory communication. **(a)** With message passing the processors (CPU) communicate directly through the network interface (NI) and the interconnect. **(b)** With shared memory communication the communication goes through the main memory (MM).

Message passing is when a packet of data is sent directly from one processing core to another. When tasks are being executed on different processors, which have no locally shared memory, a message can be sent to the other processor through the interconnect. Data that is transported via message passing must only reside in the local memory of the processor or the internal registers of the processor. Message passing is a benefit when the data is transferred directly from one local memory to another local memory. It is only allowed to write the data to the main memory once, when it should not be used any longer. Thus applications with a high level of interprocessor communication, such as streaming applications, are well suited for message passing architectures.

Shared memory is widely used in many kinds of computer systems today. Communicating between processing cores using shared memory the data will go through the main memory, or a lower level cache, which adds additional complexity due to cache coherency.

Message passing can increase the bandwidth and lower the latency of inter-processor communication compared to shared memory communication. Some systems will have both the possibility of message passing and shared memory communication. When the local on-chip memory is no longer sufficient, the off-chip main memory will come in to play.

Chapter 3

Tool chain

In this chapter we present two tool chains. The first tool chain is the tool chain we have implemented, we refer to this tool chain as our tool chain. The second tool chain is how we imagine the final version of the T-CREST tools will work together, we refer to this tool chain simply as the T-CREST tool chain. Our tool chain is slightly different from the T-CREST tool chain because not all the T-CREST tools are at a stable state.

The component based structure of the tool chains helps the integration and testing of new components. Each component can be replaced, by a component with the same interfaces. The T-CREST project dictates that the tool chain should be compatible with multi-processor platforms using TDM in the interconnect.

In the T-CREST platform, the interprocessor communication and the communication to the main memory are decoupled because there is a dedicated interconnect for each of the two communication types. The NoC is only used for interprocessor communication, in the form of message passing. Managing the main memory and the caches is all done by the compiler, therefore we have not focused on main memory access, as the compiler is out of the scope of this thesis. In the following section, we describe how our tool chain is implemented, and in the next section we describe how the implementation the T-CREST tool chain differs.

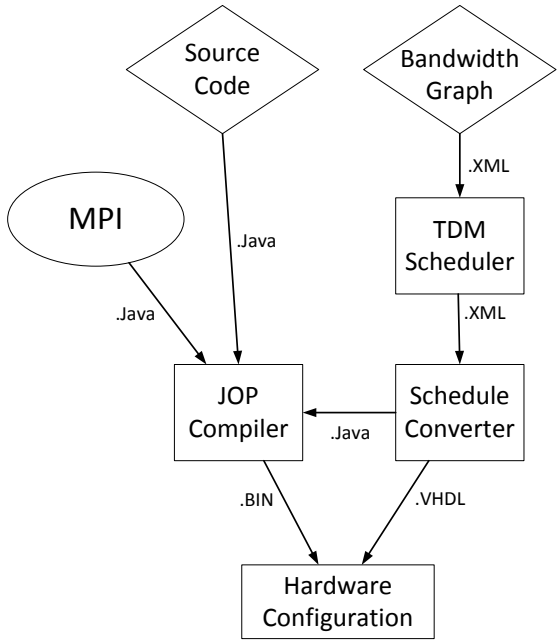


Figure 3.1: An illustration of our tool chain programmed in Java.

3.1 Our tool chain

A block diagram of our tool chain is shown in Figure 3.1. The arrows between blocks are the flow of data, and the labels on the arrows are the file formats of the interfaces. The diamond shapes are inputs to the tool chain, describing the application and its requirements. The elliptic shape is a platform specific library, mapping an abstract application interface to the given hardware platform. The rectangular shapes are the tools in the tool chain. The tool chain takes source code and a bandwidth graph as inputs. The source code describes the desired application, and the bandwidth graph describes the bandwidth requirements between all processing cores and the topology of the NoC.

First, the schedule is generated by the TDM scheduler. Then the schedule converter converts the schedule to either hardware or software tables, depending on the hardware platform. The compiler then compiles the application source code along with the MPI (message passing interface) and possibly the software tables. In the end the application is loaded on to the hardware platform. Loading the application onto the hardware, can be done before or after synthesis depending on the hardware platform. Since we use the Java programmed JOP processor,

Listing 3.1: XML example of the bandwidth specification in a 3 by 3 bitorus.

```
<?xml version="1.0"?>
<topology width="3" height="3">
  <graph type="bitorus">
    <link source="(0,0)" sink="(1,0)"/>
  </graph>
</topology>
<channels type="arbitrary">
  <channel from="(0,0)" to="(1,1)" bandwidth="10" />
  <channel from="(0,1)" to="(2,2)" bandwidth="2" />
  <channel from="(0,2)" to="(2,2)" bandwidth="2" />
  <channel from="(1,1)" to="(2,2)" bandwidth="2" />
  ...
</channels>
```

we also use the JOP compiler.

The mapping between processes and processors are done by the application programmer. The mapping is specified in the source code and in the input to the TDM scheduler. The input to the TDM scheduler is an XML formated file, describing the topology of the network-on-chip and the communication pattern of the application. We use XML because it is human readable, flexible and expendable. An example of the XML input format can be seen in Listing 3.1. In the example a `bitorus` topology is specified, other possible topology types are the `mesh` type and `arbitrary` type. If a `arbitrary` topology is specified, each link in the topology must be specified inside the `graph` tag using a `link` tag. When `bitorus` or `mesh` is specified the `link` tags are ignored. The TDM scheduler schedules the specified traffic in the topology graph. The amount of traffic scheduled between two nodes in the NoC is specified in the `bandwidth` attribute of the `channel` tag in the input XML file.

Besides the schedule, the TDM scheduler also calculates the worst-case latency (WCL) of each communication channel. The WCL of a communication channel is the worst-case time separation in time slots between access to two communication paths. If the scheduled latencies are not sufficient to meet a given real-time deadline, re-scheduling with a different bandwidth specification could decrease the WCL. The output of the scheduler is an XML file. This XML file describes how the router and network interface of each tile should be configured in every time slot and the WCL of each communication channel. An example of the output XML file can be seen in Listing 3.2, specifying a schedule of length 9. The data is separated into tiles, and in a tile each time slot describes the NI and the router. In each tile the WCL for each destination is specified.

Our MPI implements the communication primitives. The MPI hides all the

Listing 3.2: XML example of the scheduled communication channels in a 3 by 3 bitorus.

```
<?xml version="1.0"?>
<schedule length="9">
  <tile id="(2,0)">
    <timeslot value="0">
      <ni rx="(2,1)" tx="(1,1)" />
      <router>
        <output id="N" input="D" />
        <output id="S" input="D" />
        <output id="E" input="L" />
        <output id="W" input="D" />
        <output id="L" input="S" />
      </router>
    </timeslot>
    ...
    <latency>
      <destination id="(0,0)" WCL="8" />
      <destination id="(1,0)" WCL="8" />
      <destination id="(0,2)" WCL="8" />
       ...
    </latency>
  </tile>
</schedule>
```

hardware specific implementation details of the communication primitives from the application programmer. In this case of embedded real-time systems, the MPI could just as well be called the communication driver. If the hardware is changed, the driver also needs to be changed, but the application source code will not have to be changed. The WCET-aware compiler compiles and analyzes the application source code, along with the MPI library, and the timing information from the TDM scheduler.

3.2 The T-CREST tool chain

A block diagram of the T-CREST tool chain is shown in Figure 3.2. The difference from our tool chain is the programming language and the WCET-aware compiler. The T-CREST platform is programmed in C and the WCET-aware compiler optimizes the worst-case execution path in the control flow graph (CFG) instead of the average-case execution path. The WCET-aware compiler uses the information from the TDM scheduler to find the worst-case execution path.

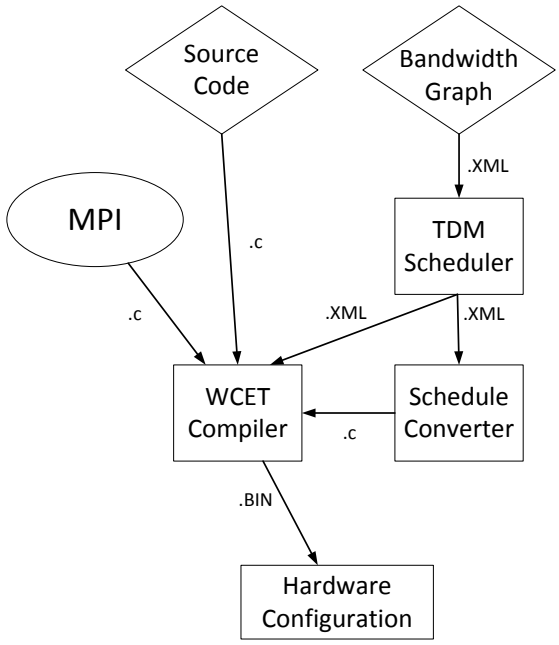


Figure 3.2: An illustration of how we imagine the T-CREST tool chain.

Chapter 4

Hardware platforms

In this chapter we present two hardware platforms and a theoretical comparison of methods to store static routing information. The current state-of-the-art real-time NoC platforms, presented in the related work section, are complex hardware devices. The first platform we present is the S4NoC platform, it is an attempt to design a minimalistic hardware platform. We made this minimalistic NoC to gain intuition on how simple a NoC can be, and as the first hardware platform for our tool chain. The second hardware platform is the first version of the T-CREST platform.

4.1 Related work

Network-on-chip has been an active research area for many years, in this project we need time-predictability to enforce real-time. The following NoC platforms provide time-predictability.

Æthereal lite The Æthereal[5] NoC was developed at Philips. Æthereal provides guaranteed service and best-effort traffic. Guaranteed service is provided using TDM. A lite version of Æthereal called aelite has been developed only

providing guaranteed service. aelite is an application specific NoC, which can be instantiated in a topology that fits to the application. The Æthereal design flow is proprietary and application specific. The hardware generation and mapping is carried out in one step. Several versions of Æthereal has been made, some using source routing and others using distributed routing.

Mango The MANGO[6] NoC was developed at the Technical University of Denmark. MANGO is an asynchronous NoC with delay insensitive links. MANGO provides both best effort traffic and guaranteed service. The guaranteed service is provided using virtual channels and rate control, opposed to TDM.

Nostrum The Nostrum[7] NoC was developed at the Royal Institute of Technology in Sweden. Nostrum implements guaranteed service with their concept of looped container, which are statically scheduled containers looping in the network. A flit can be sent via a looped container to its destination.

4.2 The S4NoC platform

The S4NoC[1] [8, 1] is a light-weight time-predictable NoC using distributed routing. The paper we wrote about the S4NoC is presented in Appendix A. To enable time-predictability the S4NoC implements TDM. The S4NoC consists of a very simple router and network interface(NI). We show a 64 core FPGA implementation of the S4NoC connected to Leros processors in [1]. The Leros processor is presented in [9]. Leros is an accumulator machine programmed in assembler or in Java. A block diagram of an S4NoC tile is shown in Figure 4.1. The router is very simple, containing one slot counter, one slot table and 5 output ports. One output port is a register and a 4-to-1 multiplexer. The NI has one word queues for each communication channel in and out of the node. These are placed in the RX and TX buffers, implemented in block RAM. The processor can read and write single words to the RX and TX buffers. Status registers indicate when words are sent and received, these status registers can also be accessed by the processor. The limited buffering makes this platform hard to program in such that the full bandwidth is utilized for all communication channels.

[1]The S4NoC is open source and is publicly available at `https://github.com/t-crest/s4noc`.

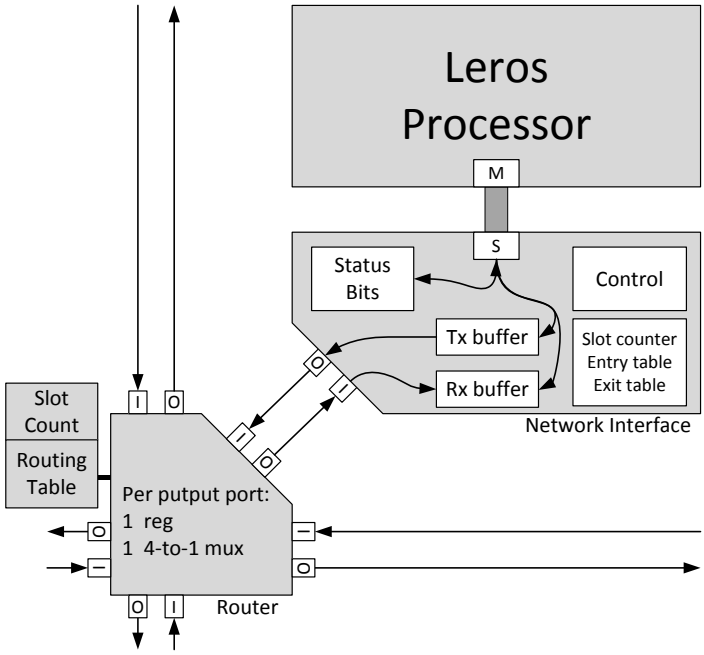


Figure 4.1: Block diagram of the S4NoC. The Leros processor can read the status bits, write the Tx buffer and read the Rx buffer.

4.2.1 Programming the platform

The Leros processors run from a local instruction ROM, no code can be loaded into it at run time. The NI is connected through the 8 bit I/O address space of Leros. Figure 4.2 shows the address space of the NI connected to Leros. The communication channel to and from each other processor in the system is mapped to the address matching its core ID. Writing to that address sends a data word to the given core, reading from the address receives a data word from the given core. The status registers indicate the receive and transmit status of each communication channel, on a word level. Flow control on a higher level than single data words needs to be implemented by the processor. The upper addresses of the address space are for the UART, the CPU ID and the total number of cores in the system.

To synchronize with the TDM slots, the NI has a counter, an entry table and an exit table, dictating when and which packets enter or exit the network. The router has a routing table dictating which input port should be routed to which

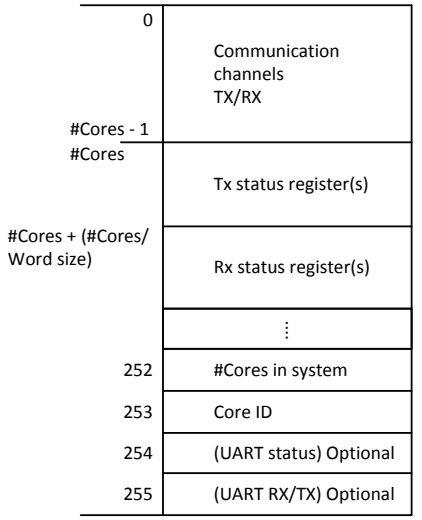


Figure 4.2: The address space of the S4NoC NI connected to the I/O address space of the Leros processor.

output ports. The tables in the NI and the router are ROMs, which in an FPGA implementation can be implemented in look-up-tables (LUTs). These tables are generated at design time, and the whole design has to be re-synthesized to change the tables. The S4NoC tables can be generated by our application specific TDM scheduler.

4.3 The T-CREST NoC platform

The T-CREST NoC, presented in [10], is a time-predictable interconnect using source routing. A block diagram of the T-CREST NI and router connected to the JOP processor can be seen in Figure 4.3. The T-CREST NoC uses direct memory accesses (DMA) to move data form one processor the another. To utilize as much of the bandwidth as possible several DMAs can be interleaved, each waiting for their time slot. This enables utilization of time slots from different communication channels at the same time. Controlling these interleaved DMAs is the core function of the NI. The interleaved DMA controller moves data from its local scratch pad to the local scratch pads of other processors. The local scratch pad memory is a dual ported memory, with one port connected to the processor, and one port connected to the NI. The individual DMA transfers are stored in a DMA table along with the route to the destination. To keep the

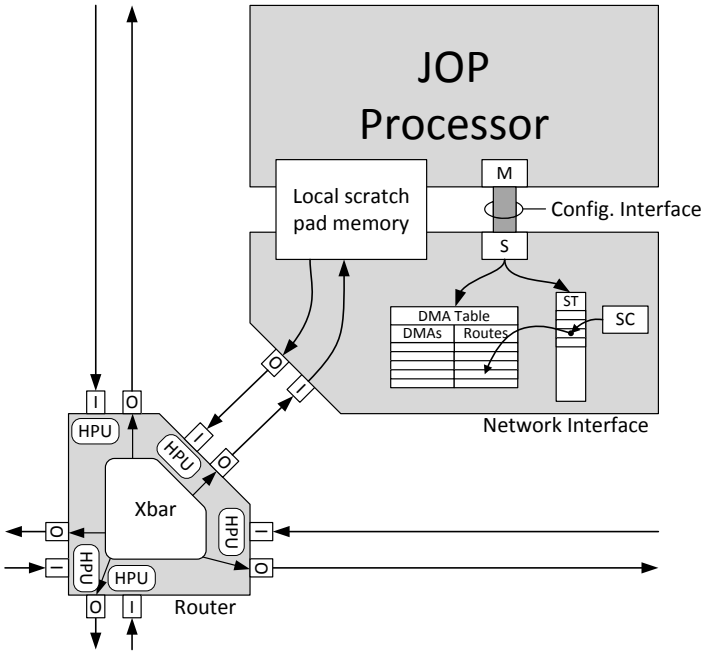


Figure 4.3: A block diagram of the DMA NoC. In the router, each input port is connected to a header parsing unit (HPU), each output port is connected to the crossbar (Xbar). The JOP processor can access the data in the local scratch pad memory and configure the slot tables (ST) and the DMA table in the NI.

synchronization with the TDM slots, a slot table is indexed by the slot counter. The slot table indexes into the DMA table. When a flit is sent from the NI, its route is stored in the flit header, along with the write pointer. The router needs to decode the header of a flit before it routes the flit to a output port. The decoding of the flit header is done in the header parsing unit (HPU) in the router. The flits are sent to the crossbar and multiplexed to the output port.

4.3.1 Configuration interface

The configuration of the DMA table, and the slot table, is carried out by the processor. Before the processor can send any packets through the NoC, the processor has to configure the slot table and the routes in the DMA table. The first operation is to write the routes of each DMA entry to the DMA table.

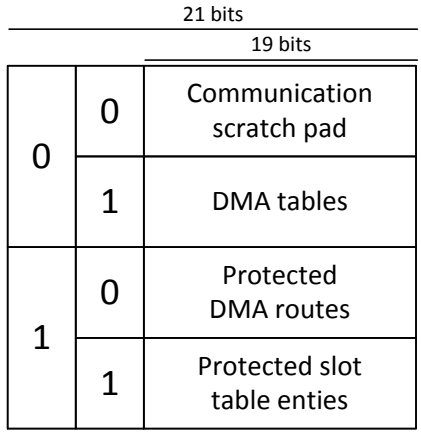


Figure 4.4: The local address space for each processor in the system. The 21 st bit is the protection bit, indicating whether to access the protected part of the local address space or not. The protected part of the address space is only for configuration of the TDM schedule.

Then it is written in each entry of the slot table, which DMA is allowed to send in that given time slot.

There are 4 segments in the local address space of each processor, the scratch pad, the DMA tables, the protected DMA routes, and the protected slot table. The number of accessible addresses in the 4 segments of the address space are not constant, nor is the ratio between them, this is because they vary depending on the current system configuration. We decide to make the simple division of the address space as seen in Figure 4.4. This is a flexible solution, but it wasts address space. The most significant bit of the 21 bits of the address space is the protection bit. The part of the address space addressed with the protection bit, should only be changed during the configuration phase of the NI.

The task of configuring the NI is done in software. Each processor needs access to the configuration data for the TDM schedule. To give access, we write the configuration data in static arrays, and compile it along with the application source code.

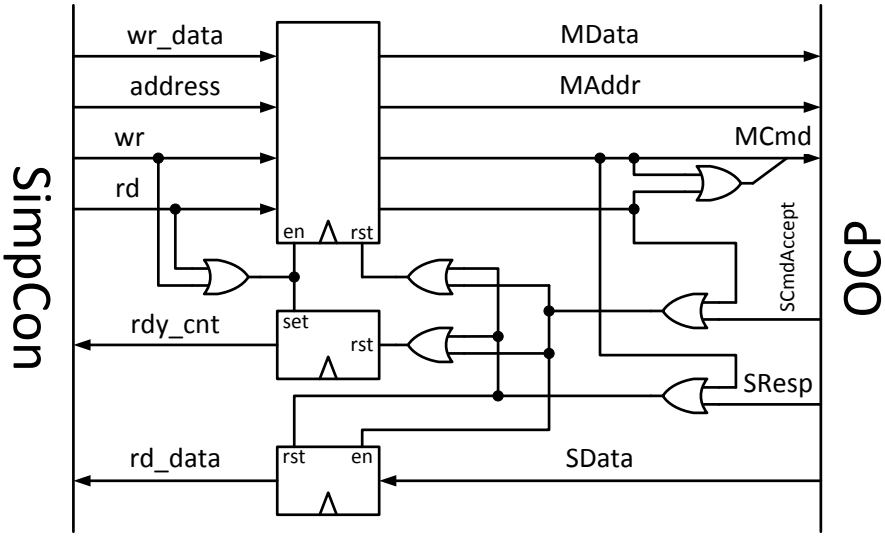


Figure 4.5: Conversion from SimpCon to OCP. The reset (rst), set and enable (en) are all synchronous signals. The circuit adds 2 clock cycles of latency to a request. The latency can be removed by adding multiplexers to bypass the registers, delivering stable signals one clock cycle earlier in each direction.

4.3.2 Integration of the hardware platform

The T-CREST processor Patmos[3] was not in a stable state at the time we were ready for integration into our tool chain. To ease our work of integrating all these alpha state components, we decided to use the well tested JOP[4] processor, with good support. The JOP processor has a SimpCon[11] interface, whereas the T-CREST NoC has a simple subset of the open core protocol (OCP)[12] interface, thus some conversion is needed. To integrate the JOP[2] processors and the T-CREST NoC[3] platform we have wrapped the NoC in an array of SimpCon interfaces. The wrapped NoC is then instantiated in the JOP top level and connected to the JOP processors. The source files we have written and changed for wrapping the T-CREST NoC platform in SimpCon interfaces can be seen in Appendix B. These files also include a testbench for the NoC wrapped in SimpCon interfaces. The source files we have changed to connect the JOP processors and the T-CREST NoC platform can be seen in Appendix C.

[2]The JOP processor is open source and is publicly available at `https://github.com/jop-devel/jop`.

[3]The T-CREST NoC platform is open source and is publicly available at `https://github.com/t-crest/t-crest-noc`.

A diagram of the conversion between the SimpCon and the OCP interfaces can be seen in Figure 4.5. The SimpCon interface supports pipelined accesses, it holds the master signals stable for one clock cycle and waits for the rdy_cnt to be 0. The reply data of the SimpCon interface is expected to be stable until the next request is started. The OCP interface needs the master signals to be stable until the slave acknowledges the request. The reply data of the OCP interface is high in one clock cycle. The incompatibilities of the SimpCon interface and the OCP interface adds 2 clock cycles of extra latency to a request. The latency can be removed by adding multiplexers to bypass the registers. These bypass multiplexers should be controlled by the enable signals, and would deliver stable signal one clock cycle earlier. Since the final T-CREST processor has an OCP interface we have focused on getting this preliminary platform to work, and not to optimize it.

4.4 Storage of static routing information

Through our work with the hardware platforms described in this chapter and the TDM scheduler, we have made two observations:

- Storage of the static routing information is a considerable part of the total resource consumption, this can be seen in [1] for the S4NoC and in [10] for the T-CREST NoC platform.
- The port configurations of the router in distributed routing stores redundant information.

The first observation has made us interested in optimizing the second observation. The following comparison, has not been verified in an implementation, but is a theoretical comparison, which could be the target for future T-CREST experiments. In the literature there are several methods to compress routing tables, such as [13] and [14]. These methods does not work in TDM NoCs. In [14] it is described how static routing tables are compressed, this kind of static routing means that the path of a flit is static, but not the arrival time as in our TDM NoCs. We have not found any compression methods in the literature that works for TDM NoCs. We compare the required storage bits for source routing, distributed routing and compressed distributed routing. A summary of the storage bits for each routing method is shown in Table 4.1

Storage bits in source routing In source routing one node in the network needs to store a route and a destination ID in each time slot. The number

of storage bits for storing a route depends on the size of the network, in the T-CREST platform 2 bits are used for each hop in the network. The route can be stored in $\lceil \log_2(2 \times H) \rceil$ bits, where H is the maximum number of hops between two processors in the network.

Storage bits in distributed routing In distributed routing one node in the network needs to store a destination ID, a source ID and the router configuration in each time slot. An ID of a processor can be stored in $\lceil \log_2(N) \rceil$ bits, where N is the number of CPUs in the system. The common way of storing the router configuration in a time slot is to store 2 bits for each output port per time slot. The 2 bits describe which one of the 4 possible input ports is connected to the given output port. With 5 ports this is 10 bits, which is 1024 possible combinations, not all these combinations are valid.

Compressing the distributed routing tables The distributed routing tables can be compressed because there are not 1024 possible router configurations. A router configuration can be perceived as a permutation of the 5 input ports, connected to the 5 output ports. An example of a port permutation is shown in Figure 4.6a. The sequence of the output ports is static and the sequence of the input ports change. In the following we calculate the number of valid port permutations.

In the actual system there are 3 restrictions on the router configurations:

1. No output port must be connected to multiple input ports.
2. No input port must be connected to multiple output ports.
3. No incoming flit must be routed out the same direction that it arrived, e.g., a flit arriving at the south input port must not depart from the south output port.

The first and second restriction implies that a valid router configuration must be a permutation of the 5 distinct input ports, such a valid router configuration can be seen in Figure 4.6a. We call a router configuration a port permutation.

The third restriction implies that if an input port is connected to the output port in the same direction it is considered as unconnected.

A port permutation where two or more ports are unconnected is is redundant because it can be represented by a permutation where the ports are swapped, i.e.,

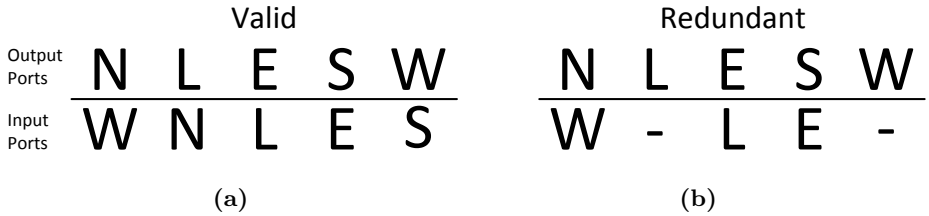


Figure 4.6: Router configurations, perceived as permutations of the 5 input ports. **(a)** is a valid port permutations and **(b)** is a redundant port permutation because it can be represented by **(b)**

the port permutation in Figure 4.6b can be represented by the port permutation in Figure 4.6a. The static schedule guarantees that no flit is routed through the ports that are swapped.

In combinatorics a port permutation with no unconnected ports is called a derangement, and a permutation with one unconnected port is called a partial derangement with one fixed point. We call the number of all valid port permutations V. V can be stored in $\lceil \log_2(\mathrm{V}) \rceil$ bits. V is found using equation 4.1, the general formula is shown in [15].

$$\mathrm{V} = D_{5,0} + D_{5,1} = \left\lceil \frac{5!}{e} \right\rceil + \binom{5}{1} \cdot \left\lceil \frac{4!}{e} \right\rceil = 89 \tag{4.1}$$

Where $D_{5,0}$ is the number of derangements of 5 elements, and $D_{5,1}$ is the number of derangements of 5 elements with one fixed point. The number of bits to store the router configuration in one time slot is:

$$\lceil \log_2(\mathrm{V}) \rceil = \lceil \log_2(89) \rceil = 7 \tag{4.2}$$

Comparing the storage of the routing methods A summary of the storage bits for distributed routing and source routing is shown in Table 4.1.

Table 4.1: Storage bits per time slot per core for source routing (Src), distributed routing (Dist) and distributed routing with compression (Dist w/comp).

	NI ID (Bit)	NI Route (Bit)	Router (Bit)
Src	$\lceil \log_2(\mathrm{N}) \rceil$	$\lceil \log_2(2 \times \mathrm{H}) \rceil$	–
Dist	$2 \times \lceil \log_2(\mathrm{N}) \rceil$	–	10
Dist w/comp	$2 \times \lceil \log_2(\mathrm{N}) \rceil$	–	7

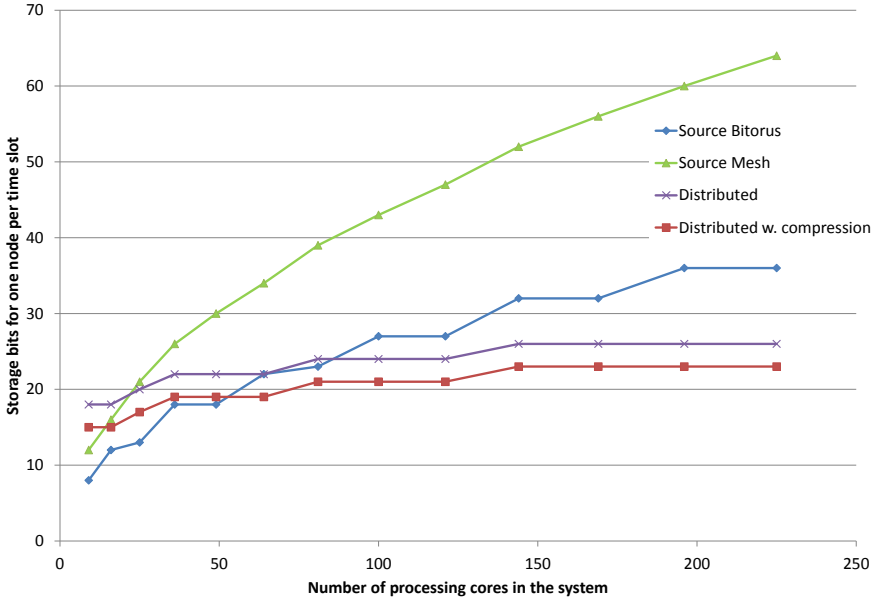


Figure 4.7: The number of storage bits for one node per time slot for source routing in a bitorus and in a mesh, and distributed routing with and without compression, as a function of network size. The storage requirements for distributed routing is the same for bitorus and mesh.

In Figure 4.7 we show the storage requirements for distributed routing and source routing as a function of the network size. The number of storage bits with source routing in a mesh network scales very poorly compared to the other routing methods. Using source routing in a bitorus scales better, due to a smaller maximum distance of two nodes in a bitorus network. The storage requirements in distributed routing is invariant for the bitorus topology and the mesh topology. The difference is in the number of time slots needed. The number of time slots when routing in a bitorus is smaller than routing the same communication pattern in a mesh topology. As can be seen in Figure 4.7, distributed routing with compression is the most efficient way in terms of storage bits for networks as small as 64 nodes. In the case of 36 and 49 nodes it can be argued that the increase in bandwidth due to distributed routing makes distributed routing with compression the most efficient routing method. This only comes at the expense of one more storage bit per time slot for each network node.

One could argue that it is not a fair comparison, because the source routing information could also be compressed. The problem with compressing the source routing information is that it adds latency through a router. The router can not start decompressing the route before the flit arrives, whereas the decompression of the router configuration with distributed routing can be pipelined.

Compressed distributed routing requires less storage bits, the decompression adds no latency to the routing, and the bandwidth is higher because no header is sent through the network.

4.5 Discussion

In this section we discuss the current limitations of the T-CREST platform, and we propose possibilities to avoid these limitations.

4.5.1 Scheduling limitations

We have found two limitations of the current T-CREST NoC platform that limit the schedules:

- The platform does not support TDM schedules where one communication channel can send in multiple time slots on different routes.
- The platform only allows for reconfigurable schedule period lengths, at synthesis.

The first limitation adds another restriction to the scheduler, this additional restriction can increase the TDM period. At this point the T-CREST NoC platform, together with the TDM scheduler we are using, only supports schedules where each communication channel has a bandwidth of one time slot per schedule period. The second limitation requires to re-synthesis to change the schedule period. For larger systems synthesis can be very time consuming, and in an ASIC, which is the target for the T-CREST project even impossible.

A block diagram of a redesigned version of the T-CREST hardware platform can be seen in Figure 4.8. To resolve the mentioned restrictions we propose to extend the architecture by moving the routes out of the DMA table, and into the slot table. The packets of a communication channel can be routed on different

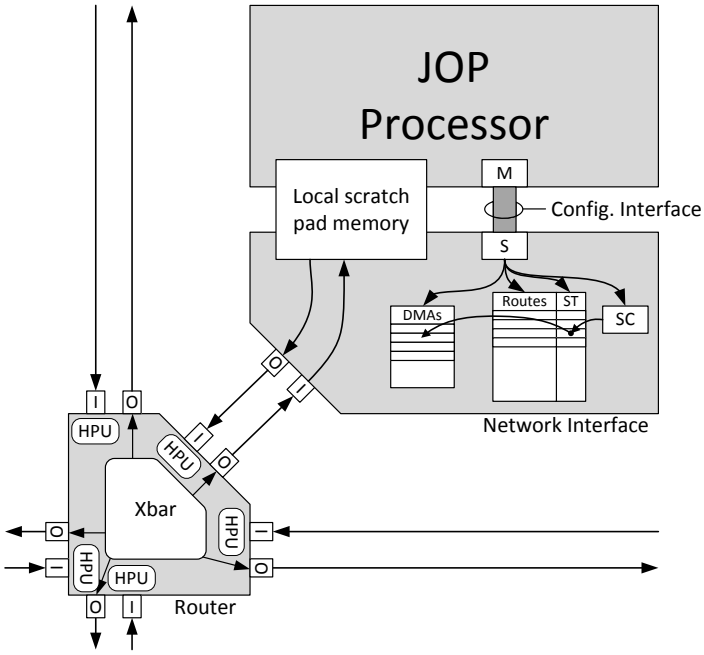


Figure 4.8: Extended block diagram of the DMA noc NI. The JOP processor can access data in the scratch pad memory, and configure the slot counter (SC), the slot table (ST) and the routes.

paths in each time slot of a schedule period. This extension requires more configuration storage, but is more flexible and can decrease the TDM period. One slot entry and one route can be written in the same configuration write, reducing the configuration time. Also the size of the static array is reduced, because the two values can be saved in the same 32 bit integer. We propose to make the TDM period configurable in run time to support variable length TDM schedules at run time. This can be supported by extending the counter to have a variable reset, configured along with the slot table.

4.5.2 Backwards flow control

In the current T-CREST NoC platform there is no backward flow control. In real-time systems where performance is analyzed, we can guarantee that tokens can be consumed at a certain rate. As long as this rate is higher than the rate at which tokens are produced there are no problems. The problem arises during

application development. The developer might not want to analyze the prototype because it takes time, or might need to lower the speed with debugging info. In these cases backwards flow control can ease development. The backwards flow control can be implemented in hardware or software. In hardware the backwards flow control can be implemented by sending empty tokens back when a place in the receiving buffer is freed by software. These empty tokens can be sent back as a specially formatted package that is processed by the NI. In software the backwards flow control can be implemented by sending a normal message back to a special address that the software in the other end is polling when it tries to send. Analyzing the systems with backwards flow control might be difficult, so backwards flow control should not be used when the application is analyzed.

Chapter 5

TDM scheduler

In this chapter we describe the scheduling problem and two types of schedulers. We also describe how we integrate an application specific scheduler into our tool chain.

5.1 Related work

The scheduling problem that the TDM scheduler needs to solve, is known as a integer multi-commodity flow problem. This problem has been proven to be NP-hard in [16]. A scheduler for scheduling all-to-all communication in these kinds of networks is shown in [17]. The advantage of this scheduler is that the schedules are symmetric, meaning that the routing tables for each router are the same, allowing for resource sharing. A scheduler for the Æthereal is shown in [18]. This scheduler schedules in two phases, the first phase is path allocation and the second is time slot allocation to TDM slots. The Æthereal scheduler instantiates extra hardware to increase the capacity on links if needed. Our tool chain needs a scheduler for scheduling application specific communication requirements on to the homogeneous T-CREST NoC platform.

5.2 Static routing

In the TDM interconnect we route packets statically, to guarantee that no packets collide. This guarantee enables us to make very simple hardware, with no arbitration mechanism or buffering. We need a TDM scheduler to create virtual end-to-end circuits. The interconnect in the T-CREST platform is a time-predictable TDM NoC. Time-predictability in the TDM NoC is enforced by a static routing. In the following we define the routing terms.

A static routing is a mapping of communication channels to the TDM links fulfilling the specification. This mapping is performed by a TDM scheduler. The communication channels are specified by the application.

Definition 5.1 The communication channel from a to b is a collection of communication paths which can route data from a to b.

The TDM scheduler finds a collection of communication paths that satisfy the specification of the communication channels. The bandwidth of a communication channel is the number of communication paths in the communication channel, divided by the schedule period.

Definition 5.2 A communication path from a to b, is a sequence of neighboring links mapped to consecutive time slots. This sequence starts in a and ends in b.

A valid communication path is mapped to one of the shortest paths from a to b. In regular topologies the length of a valid communication path is equal to the Manhattan distance from a to b.

Definition 5.3 The schedule period is equal to the length of the complete schedule in time slots. The complete schedule is a schedule that satisfies all communication channels given in the application requirements.

The requirements to a schedule is specified in the XML input of the TDM scheduler. It is specified which communication channels the application needs and the bandwidth for each communication channel. When the application developer needs a schedule for an application, the developer specifies the platform topology and the communication channels in the XML input. There are two types of schedules, there is the all-to-all schedule and the application specific schedule.

5.3 All-to-all scheduling

An all-to-all schedule is a schedule where all processors in the network can communicate directly to all other processors in the network with equal bandwidth. All-to-all schedules have advantages and disadvantages. It is an advantage that the schedule is application independent, and there is only need to configure the schedule once, and it can be implemented in hardware. For small networks the latency of an all-to-all schedule is quite small, and it is more likely that all processors need to communicate to all other processors. The all-to-all schedule is ideal in systems where the communication pattern is very uniform between all processors. It could also be an advantage to use the all-to-all schedule when prototyping a system, as long as the developer is testing functionality and not runtime requirements. An all-to-all scheduler is shown in [8], the advantage of this approach is that the schedule for each router is the same, allowing for resource sharing, and the generated all-to-all schedules are close to optimal in terms of a short schedule period.

In large networks the number of processors each processor talks to is very dependent on the application, therefore an all-to-all schedule might wast a considerable amount of bandwidth. In systems with low latency requirements or high bandwidth requirements, an application specific schedule should be calculated.

5.4 Application specific scheduling

An application specific schedule is a schedule where only processors that are specified to communicate can communicate. Creating an application specific schedule can lower the schedule period compared to an all-to-all schedule. The lowered schedule period decreases the latency and increases the bandwidth.

In our tool chain we used the Static NoC TDM scheduler[1] (SNTs)[19] to schedule the communication channels described in the XML input. The SNTs is a metaheuristic scheduler using adaptive large neighborhood search (ALNS)[20] and greedy randomized adaptive search procedures (GRASP)[21] to optimize the schedule period. The SNTs schedules the static routes in a time expanded graph of the NoC topology. A scheduled communication path in a time expanded graph of a 3 by 3 mesh is shown in Figure 5.1. The communication path marked with yellow is routed from processor 0 to processor 5. Each consecutive link of the routed communication path is routed in a consecutive time slot. The metaheuristic optimization algorithms break down part of the initial solution

[1] The SNTs is open source and is publicly available at `https://github.com/t-crest/SNTs`.

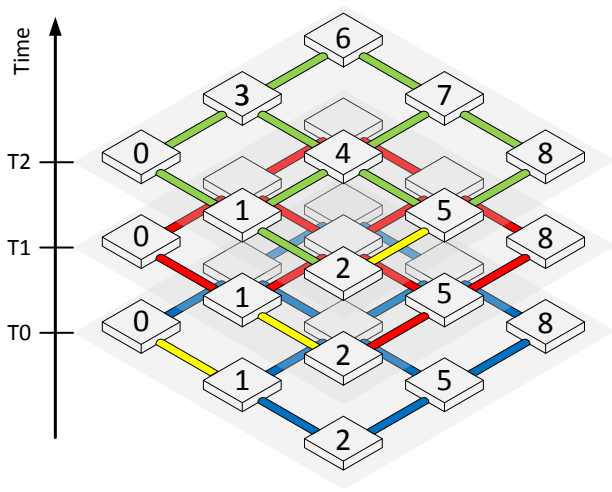


Figure 5.1: Time expanded graph of a 3 by 3 mesh topology, with a communication path routed from processor 0 to processor 5.

and rebuild it trying to make the new solution shorter than the initial. The SNTs is designed to run for days, or as long as the application designer wants, continuously trying to optimize the solution.

To integrate the scheduler into the tool chain, we have made the following extension to the scheduler:

- Support of arbitrary bandwidths for any communication channel.
- Calculation of the WCL for all communication channels.
- XML formatted output of the calculated schedule and WCL.

The files we have added to the scheduler are the `.cpp` and `.h` files listed and described in Appendix D. Support for arbitrary bandwidth is given, by allowing multiple communication paths to be routed from source to destination of a communication channel. The arbitrary bandwidth is specified in the input XML file. The WCL time for a communication path of a given communication channel is calculated when the scheduler is done. When the schedule has been created, the scheduler goes through the schedule and counts the maximum space between any two consecutive communication paths belonging the the same communication channel. The scheduler writes the schedule and WCL together into an XML formated output file. The output from the scheduler is written to an XML file

with the open source pugixml[22] library. The representation of the schedule in the TDM scheduler is router centric. The schedule describes the configuration of each router in each time slot, this description aligns with distributed routing. A description that aligns with a source routing is a schedule describing the NI and the flit routes from the NI, called an NI centric schedule. To avoid multiple conversion back and forth between router centric and NI centric schedules the schedule in the XML file is router centric.

5.5 Schedule converter

To keep the design modular, we have made a schedule converter that writes the platform specific details of the schedule. The source files of the schedule converter are the `.java` files in Appendix D. It converts the XML file into the format that is supported by the JOP processor and the T-CREST NoC platform. This conversion involves a conversion from a router centric schedule to a NI centric schedule. The conversion is performed by following the outgoing routes from each NI in each time slot. The static routing can be configured in the NoC in two ways: compile time configuration and run time configuration.

Compile time configuration is done by configuring the schedule in hardware tables at compile time. Compile time configuration is normally used in FPGA implementations. For compile time configuration we convert the schedule to VHDL tables for each node in the network, connecting it directly to the resource it is controlling. Conversion to VHDL tables is integrated into the SNTs scheduler. It prints out one VHDL entity containing a table for each router, the tables are indexed with the node ID.

Run time configuration is done by configuring the schedule by programming it from the processor at systems startup. For run time configuration we convert the schedule to a static array of integers such that each processor indexes the array with their processor ID and loads the contents of the array into the hardware configuration tables. The conversion of the XML file to a static Java array is done by a small Java program, which wraps the static array in a Table class that also defines methods for loading and verifying the schedule.

5.6 WCET-aware compiler

To get good real-time performance the compiler needs to optimize the WCET path in the control flow graph (CFG). Optimizing the WCET path, the compiler

knows the WCET path, thus the analysis and compilation could benefit from being performed by one tool. To find the WCET path, the compiler needs to make a pessimistic estimate of the run time of the given path. With more precise models the estimate can be less pessimistic. The estimate of the WCET path is found by assigning worst-case latencies to each instruction. The worst-case latencies might vary with the state of the system. Routing the interprocessor communication statically decouples this communication from the state of the system, reducing it to the WCL and the bandwidth between the two communication processes. For many core systems such a reduction in the state space is a great benefit and makes it possible to analyze the system. As an example, the latencies of a memory access to the communication scratch pad vary depending on the latency and bandwidth given by the scheduler. The latency does not have to be constant even for the same instruction, it can vary with the system's state. The latency of transferring a message ML in the system can be calculated as follows:

$$\mathrm{ML} = \mathrm{WCL} + \frac{\mathrm{MSG}_{\mathrm{Size}}}{\mathrm{Channel}_{\mathrm{Bandwidth}}} \tag{5.1}$$

Where WCL is the worst-case latency of waiting the a channel time slot, $\mathrm{MSG}_{\mathrm{Size}}$ is the size of the message and the $\mathrm{Channel}_{\mathrm{Bandwidth}}$ is the bandwidth of the channel to the message destination. In the case where the WCET is higher than allowed the ML can be lowered by scheduling more communication paths for the given communication channel or spread out the communication paths in the schedule.

5.7 Discussion

In this section we suggest two improvements to the scheduler that could decrease the latency of a communication channel in a schedule. If the bandwidth of a communication channel needs to be increased, more communication channels are routed. If the latency of a communication channel needs to be decreased, more communication paths can be added, but adding more communication paths does not guarantee this. If all the communication paths of a communication channel are routed closely together, the latency of the communication channel is worse than if the communication paths would be evenly distributed throughout the schedule. The first improvement is to make the scheduler aware of the proximity of other communication paths, when routing. A low latency channel could be specified by a low latency tag in the XML file.

The second improvement will decrease the latency of a complete transaction through the network, the scheduler could be made to support reply messages. A reply message is a message sent from a to b followed by a reply from b to a. If we know the time separation of the first message arrival and the reply message departure, called the response delay, we can schedule two communication channels, such that only the departure of the first message needs to wait for its time slot. When the reply message is ready for departure it gets its time slot right away. This could be useful when slave components are accessed with known response delays, especially for a single word reads where the WCL is the largest contributer to the latency.

Chapter 6

Message passing interface

In this chapter we will create a message passing interface (MPI) for use with our tool chain. We will discuss the communication primitives in communicating sequential processes (CSP)[23] and Kahn process network (KPN)[24] and choose which communication type to implement. We will describe the software for transferring data from one processing core to another. This software takes care of the low level, hardware specific details of data transfers.

6.1 Related work

For message passing in large computer systems, the MPI [25] standard has been made. The MPI standard specifies an interface for sending and receiving messages in a large computer system without shared memory. The MPI standard defines a set of operations for communication through message passing and run time management of processes on massively parallel systems. An open source implementation of the MPI standard is the Open MPI [26]. The MPI standard is made for large computer systems made up of many computers connected together in a cluster. What we need for our tool chain at this point is a very simple MPI with only the most basic communication primitives.

6.2 Communication primitives

To design a correct and efficient parallel application the parallelism should be considered from the early design phase. A specification of the application could be written in a formal language that supports message passing natively, such as CSP or KPN. Our hardware platform is designed to run one process on one processor. This design feature comes from the fact that running multiple processes on one processor will make the processes interfere, and the uncertainty of this interference will increase the WCET. When mapping an application onto the platform of this thesis, the application should be divided into different processes, to utilize multiple processing cores. The number of processors to map one application to, is determined by the timing requirements of the application and the resources available to the application.

Processes in both KPNs and CSP communicate by passing messages between each other. The CSP semantics implement synchronous message passing and the KPN semantics implement asynchronous message passing. Synchronous message passing is when the two processes synchronize when they exchange a message. The two processes are connected directly. This means that the sender and receiver returns from the execution of the send and receive function calls at the same time. In asynchronous message passing the two processes are connected by an infinite FIFO, meaning that the sender can send multiple messages without the receiver attempting to receive anything. Infinite FIFOs can of course not be implemented and in practice the FIFOs are bounded in size. Asynchronous message passing makes it possible to interleave calculation and communication. Both synchronous and asynchronous message passing can be implemented on top of each other. We chose the style of message passing with the lowest implementation overhead. The hardware implements asynchronous message passing with bounded FIFOs, so this is our choice. If needed, synchronous message passing can be implemented on top of our MPI, but this results in poor performance. The communication primitives we have chosen to implement are:

`Send()` The `Send()` primitive sends the specified data to the specified recipient. If the bounded FIFO towards the recipient is full the `Send()` primitive blocks until there is room in the FIFO.

`Receive()` The `Receive()` primitive receives data from a specified sender. If the bounded FIFO from the sender is empty the `Receive()` primitive blocks until there is data in the FIFO.

`RdySend()` The `RdySend()` primitive is a way of avoiding blocking `Send()` calls. The `RdySend()` primitive checks if there is room in the bounded FIFO

towards the specified recipient. `RdySend()` returns `true` if there is room in the FIFO and `false` if the FIFO is full.

`RdyReceive()` The `RdyReceive()` primitive can be used to avoid blocking `Receive()` calls. The `RdyReceive()` primitive checks if there is data in the bounded FIFO from the specified sender. `RdyReceive()` returns `true` if there is data in the FIFO and `false` if it is empty.

6.3 MPI in the T-CREST platform

As the programming language in our tool chain is Java, and the programming language in the T-CREST tool chain is C, we will only use basic Java for our MPI, which can easily be ported to C. Many of the observations we make will also be applicable in C. The source code for our MPI can be seen in Appendix E. The `Tables.java` file is the static array written by the schedule converter. In this embedded Java ported to JOP it is difficult to manage the location of variables and objects, this is a problem because the performance of message passing depends on placing the data for communication locally. In this JOP multiprocessor system, Garbage collect was not available, which limits the memory footprint and the run-time of the applications running on the system.

In the T-CREST platform, processors setup DMAs to transfer data from its local scratch pad to other processors' local scratch pad. The hardware platform we use in our tool chain is limited because it has to copy data in and out of the local scratch pad memory. Setting up a DMA requires a read pointer and a write pointer. The sending and receiving processors of a DMA transfer has to agree on the write pointer. One way of agreeing on the write pointer is to let the receiver send the next write pointer to the sender, each time it is ready to receive. Another way to agree is to layout the address space of which buffers are placed where. Allocating buffers statically is easy to analyze for the WCET-aware compiler, and it avoids the extra latency of sending new write pointers back. The downside of allocating buffers statically, is that it might waste space in the already limited local scratch pad if not all buffers are used.

6.3.1 Address space

The size of the local scratch pad of a single processor varies with the configuration of the system. Therefore the systems should be designed not to depend on a specific address space. Local scratch pad memory is very limited and accesses to main memory is very time consuming because many cores need to share the

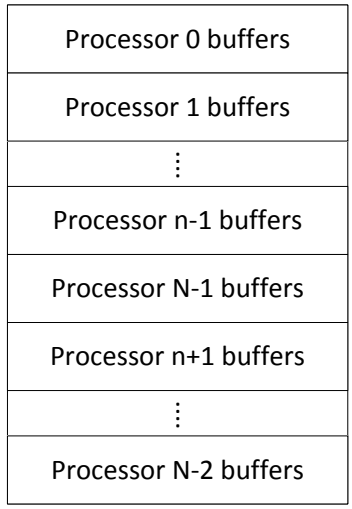


Figure 6.1: The static DMA NI address space of the n^{th} processor in a systems with N processors.

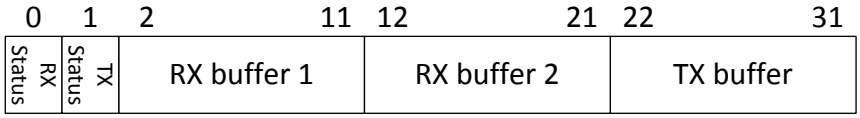


Figure 6.2: The address space in the local DMA NI of the buffers for one processor.

same off-chip memory. This means that address space of the local scratch pad should be compact. In this first version we support all-to-all communication by having buffers for all cores in the network in each tile. The static address space of the n^{th} processor is shown in Figure 6.1

In a network with N nodes each NI has buffers for the N-1 other nodes. The nodes are zero indexed. We need to know the addresses statically, and we need to compact the address space. The buffers for the $N - 1^{th}$ node is positioned in place of the local tile buffers. In this way all nodes can calculate their buffer address in all other cores. The address space of the buffers for one processor is shown in Figure 6.2.

The hardware does not support any way of signaling that a DMA transfer is finished. To signal that a DMA transfer is done we wrap the data in a header and a footer phit. The header phit carries the length of the complete DMA transfer and the footer carries 0xFFFFFFFF. The size of the maximum data

Listing 6.1: Pseudo code for the `Send()` primitive.

```
Send()
  while not RdySend() do
    do nothing
  od;
  copy message to mem
  swap receive buffer
  setup DMA
```

Listing 6.2: Pseudo code for the `Receive()` primitive.

```
Receive()
  while not RdySend() do
    do nothing
  od;
  copy message from mem
  swap receive buffer
```

message is 8 words (32 bytes).

6.3.2 Communication primitives

In this section we describe how the communication primitives are implemented.

`Send()` The `Send()` primitive, setup a DMA transfer to transmit the data to the recipient. To send a packet we need to check that there is not a DMA transfer in progress. If no DMA is in progress the message is copied into the transmit buffer, and the buffer in the receiving end is swapped. To complete the send operation we need to set up the DMA transfer. The pseudo code for the `Send()` primitive is shown in Listing 6.1.

`Receive()` The `Receive()` primitive, waits until a DMA transfer has completed. When the transfer has completed, the message is copied out and the receive buffer is swapped. The pseudo code for the `Receive()` primitive is shown in Listing 6.2.

`RdySend()` The `RdySend()` primitive checks if the DMA is ready to setup. To check the status of the DMA, the DMA done bit is read from the NI. The pseudo

Listing 6.3: Pseudo code for the `RdySend()` primitive.

```
RdySend()
  read DMA done bit
  if done bit equals 1
    return true
  fi;
  return false
```

Listing 6.4: Pseudo code for the `RdyReceive()` primitive.

```
RdyReceive()
  read header
  if footer equals -1
    return true
  fi;
  return false
```

code for the `RdySend()` primitive is shown in Listing 6.3.

`RdyReceive()` The `RdyReceive()` primitive checks if a DMA transfer has completed. To check if a DMA transfer has completed we read the header for the length of the transfer. We wait for the footer of the transfer to be 0xFFFFFFFF. The pseudo code for the `RdyReceive()` primitive is shown in Listing 6.4.

6.4 Discussion

In this section we discuss possible improvements to our MPI.

6.4.1 Dynamic allocation of buffering space

To make better use of the scratch pad, dynamic allocation of the buffering space can be applied. Then the first step in sending a message would be to allocate a buffer of the size of the message. After the message was sent the buffering space would then be freed. Allocating the message buffers to the local scratch pad could be done using the first fit algorithm, starting from the lowest address finding the first possible place to allocated the buffer. In real-time systems dynamic behavior can make analysis more difficult, because the system's state

is more complicated. One way of modelling the state is to fix the maximum message size and then only allocate buffers of this size. Then the compiler can keep a worst-case count of the number of outstanding packets. The first problem with dynamic allocation is to determine who will free the allocated buffers. The hardware is the last to use the transmit buffers, and the software is the last to use the receive buffers. If the software is freeing the buffers, then it needs to poll the transmit buffers to check if they are done. If the hardware is freeing the buffers, the bookkeeping needs to be in the communication buffer, which is already crowded. This will infer an unwanted overhead into the communication primitives. The dynamic allocation still suffers from having to send information about its receive buffers to the processors trying to transmit to it. A compromise to avoid sending addresses of receive buffers back is to allocate the receive buffers statically and the transmit buffers dynamically. This would also simplify the analysis because the dynamic behavior is local and independent of other processors.

6.4.2 Compiler optimizations

If the WCET-aware compiler can find the message sizes when it analyzes an application, it can allocate both the static and dynamic buffers in the local scratch pad, avoiding unused buffering space. The tool chain and programming model should help the programmer to parallelize the applications. The WCET-aware compiler should help the programmer by giving feedback. Such feedback could be information on the load of the different processors helping the programmer to load balance the application. In this first iteration, where we do not have a compiler with these abilities we choose to implement the dumb all-to-all address space and communication primitives.

Chapter 7

Test

In this chapter we show how a Hello World program is implemented with our tool chain, and show how the system could be benchmarked.

7.1 Hello World!

We will show how a Hello World program is taken through our tool chain and finally implemented on the T-CREST NoC platform. Our Hello World program sends a message through all processors in a ring. In Listing 7.1 we show a piece of the source code for processor zero. The full source code can be seen in Appendix F. Processor zero initializes the DMA NI by writing the static tables. Then it initializes the other processors by setting a runnable. When the other processors are started, processor zero starts by sending the message to the processor with the highest ID. When the message reaches processor zero again "Hello World!" is written to the console. The stringbuffer is a message queue for the other processors for writing out their start messages.

Listing 7.2 shows the XML input for the TDM scheduler. In the XML file we specify that the topology is a 3 by 3 bitorus and the communication pattern of the application is all-to-all. For the Hello World application the actual communication pattern is a ring. We will show application specific schedules in the

Listing 7.1: Source code the the Hello World application.

```
Tables.load(0);    // Initialization of DMA NI
System.out.println("Core 0 started");
for (int i=0; i<sys.nrCpu-1; ++i) {
  Runnable r = new HelloDMA(i+1);
  Startup.setRunnable(r, i);
}
// start the other CPUs
sys.signal = 1;
int[] message = {0,1,2,3,4,5,6,7};
int[] rmessage = {0,0,0,0,0,0,0,0};
NoC.send(message,sys.nrCpu-1,0);
for (;;) {
  int size = msg.size();
  if (size!=0) {
    StringBuffer sb = (StringBuffer) msg.remove(0);
    System.out.println(sb);
  }
  if(NoC.recvRdy(1,0)){
    NoC.recv(rmessage,1,0);
    for(int i = 0; i < message.length; i++){
      if(message[i] != rmessage[i]){ System.exit(1);}
    }
    System.out.println("Hello World!");
  }
}
```

Listing 7.2: XML input for the Hello World application.

```
<?xml version="1.0"?>
<topology width="3" height="3">
    <graph type="bitorus"></graph>
</topology>
<channels type="all2all">
</channels>
```

Listing 7.3: Console output of the Hello World application running on the hardware.

```
JOP start V 20110107
60 MHz, 2048 KB RAM, 9 CPUs
Core 0 started
Core 1 started
Core 2 started
Core 3 started
Core 4 started
Core 5 started
Core 6 started
Core 7 started
Core 8 started
Hello World!
```

following section. The application output to the console is shown in Listing 7.3. First we see the JOP processor starting, then each processor in the network starts. At last the message is passed around the network and "Hello World!" is written to the console.

7.2 Microbenchmarks

To measure the performance of a system real applications should be used as benchmarks. For our current tool chain we do not have any real applications, so microbenchmarks are the only way of benchmarking the system. With the current state of the T-CREST platform, microbenchmarks is a good way of characterizing specific design features. Microbenchmarks are good for evaluating different design alternatives, which is what the T-CREST project needs in its current state. The NoC Benchmark by OCP-IP states that microbenchmarks[27, sec. 2.3] for a NoC should benchmark:

- Packets and transactions

- Unloaded and loaded cases
- Temporal and spatial distribution
- Best effort and guaranteed services
- Network size

Not all these benchmarks apply for our statically scheduled NoC. The latency and bandwidth of communication channels are invariant of the communication load in the network. Therefore it is also irrelevant to benchmark temporal and spatial distributions of the traffic in the network. Also the system does not provide best effort services. For our network it is only relevant to do microbenchmarks for packet transfers and complete transactions as a function of the network size. Our FPGA only fit 9 JOP processors and the T-CREST NoC, so varying the network size is not interesting.

In general purpose systems benchmarks are used to measure the performance of the system. In real-time systems, where the performance is equal to the calculated WCET, the benchmarks should be analyzed by the WCET-aware compiler. In the current state of the T-CREST project, where we do not have a WCET-aware compiler, we do the measurements in hardware. In our benchmark we will measure the following operations:

Send() The time it takes to perform a send operation when there is room in the transmit buffer.

Recv() The time it takes to perform a receive operation when there is a message in the receive buffer.

Echo() The time from sending a message to another processor till a message is received from the given processor.

Roundtrip() The time it takes to send a message through all processors in the network.

The source code of the microbenchmark can be seen in Appendix F. In Figure 7.1 we show the execution time of the microbenchmarks as a function of the message size. For echo and roundtrip we also show two interleaved operations. The execution time of the interleaved roundtrip operation is around 6 percent larger than the normal operation, sending twice the data around in the network. The execution time of the interleaved echo operation is around 60 percent larger than the normal operation, sending twice the data.

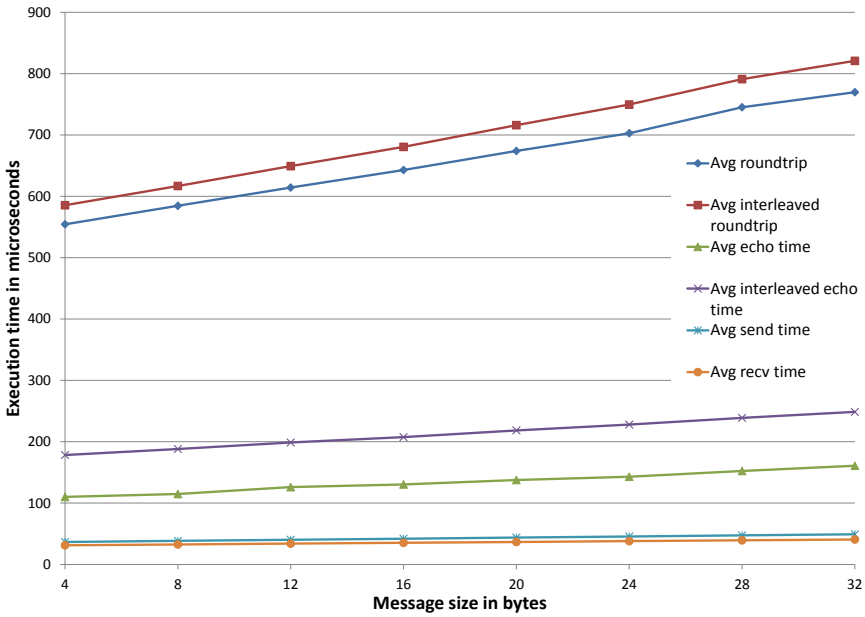


Figure 7.1: Measured execution time of the microbenchmarks as a function of the message size with an all-to-all schedule.

To optimize the execution time of the microbenchmarks we make an application specific schedule, and the XML file specifying this application specific communication pattern can be seen in Listing 7.4. In the XML file only the communication channels that are needed are specified. Using an application specific schedule decreases the execution time slightly. A plot of the execution times of the roundtrip benchmark and the echo benchmark, with and without an application specific schedule, is shown in Figure 7.2. One reason why the execution time with the application specific schedule is not a greater improvement could be the size of the network. For small networks such as the 3 by 3 bi directional torus, the all-to-all schedules have quite low latencies, and an application specific schedule can not improve the run time by an order of magnitude. The application specific schedule will result in larger improvements for larger networks. Since the design is optimized for WCET we expect to see a larger improvement when the benchmarks are analyzed.

Another reason could be that the execution time of the benchmarks is dominated by processor I/O. We have measures the execution time of a single read from the local scratch pad memory to be 28 clock cycles. With a schedule period of 10 or less these I/O capabilities suppress the significance of the NoC delay.

Listing 7.4: XML bandwidth specification for the microbenchmark.

```
<?xmlversion="1.0" encoding="UTF-8"?>
  <topology width="3" height="3">
    <graph type="bitorus"></graph>
  </topology>
  <channels type="arbitrary">
  <channel from="(0,0)" to="(2,2)" bandwidth="1" />
  <channel from="(0,0)" to="(1,1)" bandwidth="1" />
  <channel from="(0,0)" to="(0,1)" bandwidth="1" />
  <channel from="(0,0)" to="(2,0)" bandwidth="1" />
  <channel from="(0,0)" to="(1,0)" bandwidth="1" />
  <channel from="(1,0)" to="(0,0)" bandwidth="1" />
  <channel from="(2,0)" to="(1,0)" bandwidth="1" />
  <channel from="(2,0)" to="(0,0)" bandwidth="1" />
  <channel from="(0,1)" to="(2,0)" bandwidth="1" />
  <channel from="(0,1)" to="(0,0)" bandwidth="1" />
  <channel from="(1,1)" to="(0,1)" bandwidth="1" />
  <channel from="(1,1)" to="(0,0)" bandwidth="1" />
  <channel from="(2,1)" to="(1,1)" bandwidth="1" />
  <channel from="(0,2)" to="(2,1)" bandwidth="1" />
  <channel from="(1,2)" to="(0,2)" bandwidth="1" />
  <channel from="(2,2)" to="(1,2)" bandwidth="1" />
  </channels>
```

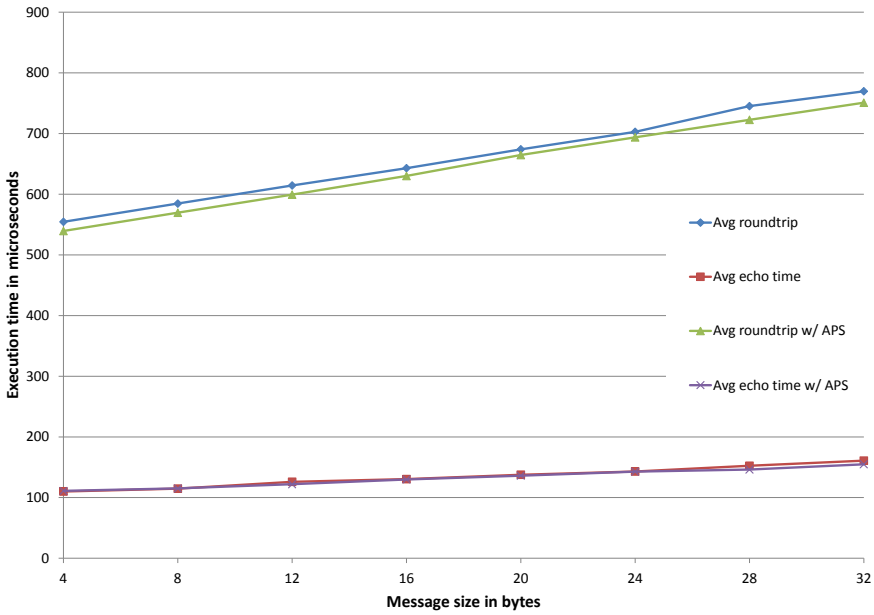


Figure 7.2: Measured execution time of a message round trip, with all-to-all and application specific schedules (APS).

Chapter 8

Conclusion

In this chapter we conclude the work carried out in this thesis. First we summarize our findings and describe the contributions of the thesis. Finally we point to future areas of work.

8.1 Summary of findings

In this thesis we have presented our tool chain for programming a real-time multi-processor platform. This tool chain is very similar to how we imagine the final T-CREST tool chain. Our tool chain is ready for integration in the full T-CREST platform.

We used our tool chain to implement the first application sending messages around between processors with the T-CREST NoC platform in an FPGA. During the work of integrating the hardware platform into the tool chain, we have identified limitations and suggested how they can be removed. The current platform is limited by low run-time configurability, no optimization of WCL and static buffer allocation in the MPI. We have changed the interface of the TDM scheduler to be compatible with our tool chain. We have implemented a Kahn process network style message passing interface in Java to communicate asynchronously between processors.

We have also tested our tool chain by implementing a "Hello World" program using 9 processors to send one message around. Another test was to implement a few microbenchmarks and measuring their execution time. The benchmarks shows that the tool chain is ready to help the developers of the T-CREST project to evaluate their components.

8.2 Project contribution

The contribution bullets from the introduction are elaborated in the following bullets in a one to one correspondence.

- We have defined the file format of the interfaces in the block diagram for both the tool chains. We have presented the structure and tags of the XML files.
- We have implemented a minimalistic time-predictable NoC platform that is published in [1].
- We have integrated the the T-CREST NoC platform and the JOP processor to one hardware platform that can be programmed by our tool chain.
- We have shown that distributed routing is more efficient than source routing in terms of storage bits for larger networks. In networks of 36 nodes or larger, we have shown that distributed routing is a better trade-off, because bandwidth is higher, hardware is simpler, and the storage bits are about the same or less.
- Working with the T-CREST NoC platform we have suggested to extend the hardware by making it more configurable in run-time. We proposed to move the routes in the DMA table to the slot table, and to make a programmable reset of the slot counter. These extensions will make the hardware more flexible and enable better utilization of the hardware.
- We have changed the scheduler to support arbitrary bandwidths for communication channels, calculate the WCL and output schedule and latency information to the WCET-aware compiler in XML format. To configure the network interfaces we convert the schedule into a static array, which is loaded into the network adapters at run-time. This approach can be extended to enable loading of a new schedule at run-time, if the mode of operation changes.
- To reduce the WCL we propose to make the scheduler aware of the location of other paths in the same communication channel when scheduling a path.

When the scheduler knows the path locations it can spread out the paths minimizing the the WCL for the given bandwidth.

- We have implemented a message passing interface using statically allocated buffers. These statically allocated buffers can be made more efficient with the suggested compiler support. The MPI implements asynchronous message passing with bounded buffers.
- We have proposed to change the MPI to use dynamic allocation of transmit buffers and static allocation of receive buffers.
- We have implemented a Hello World program on the T-CREST NoC platform using our tool chain. We have made microbenchmarks to enable evaluation of design features for the developers of the T-CREST platform.

8.3 Future work

The T-CREST NoC platform should be updated with the suggested improvements, to increase the flexibility of the platform. A version of the T-CREST platform using compressed distributed routing should be investigated. Our results indicate that the resource consumption should decrease and the bandwidth should increase. Also the hardware complexity of the router should decrease to something similar to the router of the S4NoC platform. When the T-CREST WCET-aware compiler and the Patmos processor are stable, they should be integrated into the T-CREST tool chain.

Appendix A

S4NoC paper

Our published paper about the S4NoC is attached in the following pages of this appendix.

A Light-Weight Statically Scheduled Network-on-Chip

Rasmus Bo Sørensen, Martin Schoeberl, Jens Sparsø
Department of Informatics and Mathematical Modeling
Technical University of Denmark
Email: rasmus@rbscloud.dk, masca@imm.dtu.dk, jsp@imm.dtu.dk

***Abstract*—This paper investigates how a light-weight, statically scheduled network-on-chip (NoC) for real-time systems can be designed and implemented. The NoC provides communication channels between all cores with equal bandwidth and latency. The design is FPGA-friendly and consumes a minimum of resources. We implemented a 64 core 16-bit multiprocessor connected with the proposed NoC in a low-cost FPGA.**

I. Introduction

For chip-multiprocessor (CMP) systems used in real-time systems we need time-predictable processors, memories, and communication channels. For on-chip core-to-core communication, a network-on-chip (NoC) is a scalable solution. In order to build a time-predictable CMP, the NoC is time-division-multiplexed (TDM). The NoC uses a static schedule; tables implementing this schedule are stored in each router and each network adapter. We use a schedule that provides all-to-all communication between all nodes, as depicted conceptually in Figure I.

In [11] we have shown that a router for a statically scheduled NoC is very small. In this paper we explore the full design, containing a processor, the network adapter, and the router. We explore how small this system can be and still represent a usable architecture. In other words we aim at a many-core architecture in a medium size FPGA. With our size-optimized processor Leros [10], which can be implemented in about 190 logic cells (LC), we set a very low bar for a NoC. One expects that the communication infrastructure is smaller than the processing node.

One TDM based router and one minimalistic network adapter consumes 665 LCs and 2 on-chip memory blocks for an 8x8 bi-torus configuration. Therefore, we were able to synthesize and run a 8x8 CMP system, containing 64 processors, network adapters, and routers, in the low-cost Cyclone II FPGA EP2C70 on the DE2-70 board. The contributions of the paper are:

- The design of a minimal network adapter for a TDM based NoC
- A 64 core CMP, running a simple test application
- Providing the NoC in open-source form

The paper is organized as follows: The following section presents related work in the area of real-time NoCs. Section III presents the design of the TDM scheduled network-on-chip. A minimal network adapter is described in Section IV. The simple implementation of the system is presented in Section V.

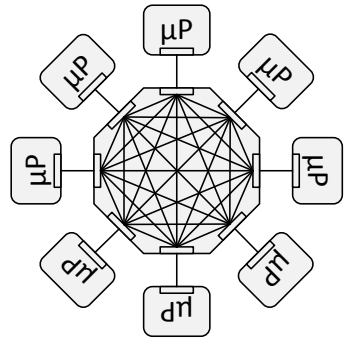


Fig. 1. A conceptual interconnect providing all-to-all connection between micro processors (μP).

An alternative implementation of the system is described in Section VI. We present our results in Section VII. In Section VIII we discuss the strengths and weaknesses of the design. The paper is concluded in Section IX.

II. Related Work

Æthereal [4] uses TDM, i.e., reserves resources for certain points in time. In each time slot a block of data is forwarded through a router without waiting or blocking traffic, hence, contention cannot occur. Slot tables with routing information are contained in the routers and no arbitration or link-to-link flow control is required. Instead, a credit-based flow control is applied for end-to-end control, saving buffer space between links. Guaranteed services are combined with best effort routing in order to utilize unreserved resources. aelite, a light version of Æthereal, only offers guaranteed services resulting in a simpler router design [5]. Slot tables are placed in the network adapter and routing is done through message headers. In the latest version of aelite, called dAElite [12], the static routing tables are back in the routers to support multicast routing.

SoCBUS [13] and the NoC presented in [14] use a circuit-switching NoC, i.e., no resources, such as wires and router buffers, are shared between connections. This lowers utilization and increases costs. However, once a connection has been established, real-time guarantees are trivially achieved. It is, however, possible that a requested connection cannot be set

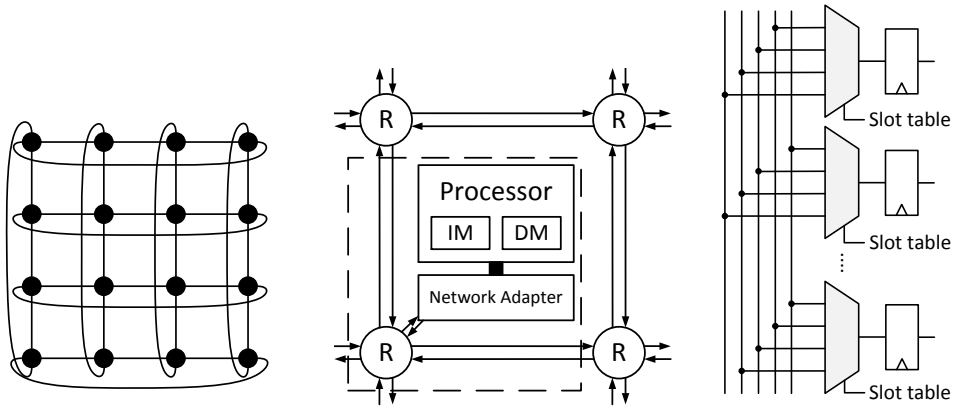


Fig. 2. The network architecture: (a) the bidirectional torus topology, (b) a node/tile, and (c) the router.

up due to lack of resources (links) – this may compromise the real-time properties.

MANGO [1] is an asynchronous NoC, which supports both guaranteed service (GS) and best effort (BE) traffic, by using non-blocking routers and rate control. A non-blocking router requires a separate physical buffer for each virtual circuit, an elaborate arbitration mechanism for each router output port, and a credit-based flow control mechanism among output buffers in neighboring routers. This indicates a considerable hardware cost of the rate-controlled routers.

A time-triggered NoC (TT-NoC) applies the concepts of the time-triggered architecture (TTA) [6] to NoCs [9]. The TT-NoC consists of a ring structure and is therefore only intended for a small number of IP-cores. As the ring is built out of simple multiplexers and registers, it is clocked at double the frequency of the computation nodes. Similar to our presented design, the communication schedule is static and predetermined.

Paukovits and Kopetz use a time-triggered NoC for the time-triggered system-on-chip (TTSoC) architecture [7]. The messages use wormhole routing and the TDM slotting is based on complete message transmissions. The TTSoC is topology agnostic. The prototype uses an uncommon version of a mesh topology: a 3x2 mesh supporting 10 computation nodes. Therefore, the corner routers are connected to two computation nodes. Our design shares the idea of static scheduling based on TDM. However, we base our schedule on the finer granular network clock and take pipeline effects into account in the network.

III. A Statically Scheduled NoC

In [11] we presented the idea of a statically scheduled TDM-based NoC, called S4NoC, that provides all-to-all communication in regular topologies (e.g., mesh, torus, bi-torus, tree). We presented results on the minimum period of a schedule that provides all-to-all communication and derived first resource estimates for the routers. All-to-all communication schedules, which are only 15% to 20% longer than theoretical lower bounds, can be calculated with a heuristics [2].

In this paper we design a simple network adapter to go along with the simple router and we implement the whole system. The network adapter has to do some bookkeeping and buffering of data and thus the design will be more complex than that of the router. We still aim to keep the design as simple as possible.

A router for the NoC is very simple, which is one of the motivations for using a statically scheduled TDM-based NoC. For a mesh or a bi-torus a router has 5 bi-directional ports (north, east, south, west, local) and each output port is a pipeline stage consisting of a register with a 4-to-1 multiplexer on its input (in and out of the same port is not allowed). The multiplexers are controlled by schedule tables indexed by a slot counter. This avoids the need to transmit address information with the packet. Without the pressure to amortize for the header overhead we can use arbitrary short packets. Therefore, we transmit and schedule single words as packets, which helps to keep the schedule period short and the latency moderate.

For the evaluation described in the following sections we assume a bi-torus topology, as shown in Figure 2(a). Each node consists of a processor with local instruction and data memories, a network adapter, and a router, as shown in Figure 2(b). The processors execute from their local memories and communicate by sending messages across the network. The NoC provides (virtual) channels, all with the same bandwidth, allowing a processor to send messages to and receive messages from all other processors. For simplicity we restrict to single word messages, and by using the same width of the links and routers in the NoC we get a simple design, as illustrated in Figure 2(c), where a message traverses a router in one clock cycle.

The router is obviously very simple (i.e. small and fast) and

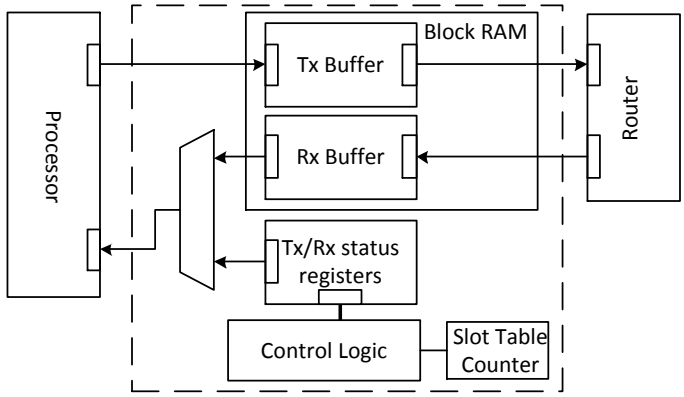


Fig. 3. A tile including the Leros processor, the network adapter with receive, transmit, and status registers, the interface to the router, and the router.

the sizes of different processors targeting FPGA implementations are also quite well known. The third and last component in a tile is the network adapter. Despite our aim for simplicity its function is non-trivial, and its size and speed is difficult to assess. This is one of the main reasons for the design experiment reported in this paper – to get reliable speed and area figures and to gain insight in the design of this critical component.

The network adapter's interface towards the processor is similar to a memory mapped IO-device, and it offers input and output registers corresponding to all the incoming and outgoing (virtual) channels connecting it to all the other processors. The design is described in more detail in the following.

IV. The Network Adapter

The basic functionality of a minimalistic network adapter (NA) is to present an interface to the processing core, which enables the processor to access communication channels to other cores efficiently. The processing core should not be concerned with managing time slots. To fully utilize the network, there are the following requirements to the minimalistic NA:

- Provide an interface to view the status of all channels to the processor
- Send and receive single data words to and from the network in line with the TDM mindset to all other cores in the system
- The NA must be able to transmit and receive data in all consecutive time slots

To synchronize the sending and receiving of flits (transmitted logical words) to the router, the NA uses a time slot table. The time slot table is generated from the static schedule of the size and topology of the desired system. The time slot table in a NA, maps a given time slot to a source and a destination address. These addresses are the ID of the receiving or transmitting processing core, thus the time slot tables are different for all NAs. The time slot table is driven by a counter in the NA.

The block diagram of the NA is shown in Figure 3. The processor can write to the transmit (Tx) buffer, or read from either the receive (Rx) buffer or the status registers. The status registers shows the status of each register in the Tx or Rx channels, i.e., if the Tx register is ready to receive or if the Rx register is ready to be read out.

The interface to the processor is an address space, where each communication channel is mapped to one address and status registers are mapped to several registers depending on the number of cores in the network. In each of the two status registers, each bit represents a communication channel to one other core in the system. Maximizing the utilization of the given hardware, the static schedule is made such that the NA can both send and receive flits in each time slot.

In this simple NA not much control is needed. The task of the control logic is to set and reset bits in the status registers when flits are received and transmitted. The task of controlling each bit of the status registers individually is not complicated, but an increasing number of bits lead to an increasing amount of control logic. To set and reset each bit of the status registers efficiently the status registers should be implemented in flip-flops.

V. Implementation

In this first simple implementation the whole system resides in one global clock domain. Our design is technology agnostic, but in this section the implementation decisions are related to the Cyclone II FPGA we have used for testing.

Processor Interface: On the processor side of the network adapter, the processor needs the ability to read out the status of the communication channels and to read or write data to the individual communication channels. The data to the

communication channels are written or read directly to/from the block RAM. Because the address on the block RAM is registered there is a one cycle delay on a read form the communication buffer. The simple way of solving this problem is to require that when the processor wants to read data, the same read instruction should be executed twice. When the status registers are read there is a multiplexer for selecting which part of the status register to select. In a design where a read from the NA would be limiting the clock frequency of the processor, the NA could implement indirect addressing. For indirect addressing the processor writes the address of a request into an address register and in the following clock cycle the data on that address can be accessed. Indirect addressing will cut the processor I/O bandwidth in half, but the clock frequency of the system could increase.

Communication Buffer: In this simple implementation we use one dual ported block RAM for each communication buffer. A block RAM in a Cyclone II FPGA is 4 KBit. Using the block RAMs as buffers, one port is only used for writing and the other port only used for reading. Our system supports all-to-all communication and each communication channel requires two 16-bit words of storage in the NA. The two RAM blocks will support systems of up to 16x16 nodes. In the Cyclone II FPGA the RAM blocks will not be fully utilized. In systems where block RAMs can be instantiated with a finer granularity the resource consumption can be decreased. In an ASIC design the utilization can be made to 100 %.

Control Logic: Our implementation of the described design is not tuned for any specific number of processing cores. The circuitry for selecting and updating the status registers are (Number of cores)-to-1 multiplexers, and 1-to-(Number of cores) decoders. Updating the status registers can be limiting the clock frequency for a sufficient number of cores in the system. When instantiating a system of a specific size, the control logic can be pipelined, if the desired frequency is not obtained. This pipelining results in a longer latency for status register updates, both for setting and resetting, the software should be aware of this longer latency.

VI. Alternative Double-Clock Implementation

The routers are simple: just registers connected with 4:1 multiplexers. Therefore, those can be run at a higher frequency than a processor. If we use a second clock, synchronous to the main clock and double the frequency, we can time share the router. Thus reducing the size of the router by 50%. The block RAM usually can also run at a higher frequency (e.g., at up to 250 MHz in the Cyclone II device). Therefore, it can also use the double-frequency clock. Then we need only one block RAM for both communication buffers. The block RAM consumption for an FPGA implementation can be reduced for all NoC sizes up to 11x11.

Running the network at a higher frequency requires the NA to split a single flit into two phits. An extra pipeline stage is needed to align phits (physically transmitted words) to flits in a TDM time slot. The processors in the dual clock design all run in the primary clock domain. The network i.e., the routers and part of the NAs run in the double-clock domain. As the clocks are synchronous there is no real clock domain crossing needed. Only the back-end of the NA needs to handle the splitting and merging of phits between the two clock domains. The block RAM is using the double clock to double the number of ports.

If both sides of the block RAM are clocked with the network clock, the NA can return the value of a read to the processor in the same clock cycle as the read is made, thus the need to execute the read instruction twice is eliminated. Furthermore the NA can be made to support simultaneously read/write from the processor, which is supported in the processor interface but not in the Leros Processor. Implementing the Tx and Rx buffers in one block RAM requires three ports to the block RAM. One read/write port for the processor interface, one read port for the Tx channel on the network side of the block RAM and one write port for the Rx channel on the network side of the block RAM. The NA buffers the phits of a flit until the entire flit is sent or received. A flit can be read from the Tx buffer in every even clock cycle and a flit can be written to the Rx buffer in every odd clock cycle.

Additional complexity is added to the NA when the data width of the router is cut in half. The reduction in data width calls for serialization in the NA, taking more area. Also the multipumped block RAM increases complexity, multiplexing the Rx and Tx data through the same port on the block RAM. A not so obvious source of added complexity is the control logic. If the large multiplexers for selecting the status bit to update are clocked on the fast clock, they may need pipelining.

TABLE I
Resource consumption and maximum frequency of one network adapter (NA) and one router (R) implemented in a Cyclone II (EP2C70) FPGA. The numbers include the TDM time slot tables.

Cores	16	25	36	49	64
LUT	278	383	484	517	665
Reg	145	171	186	197	217
RAM (KBits)	1	1	2	2	4
Freq. (MHz)	106.6	106.7	104.3	106.0	104.8

VII. Results

To obtain results for resource consumption and maximum frequency we have used Quartus II 12.0 to compile and synthesize the design. We have also tested the implementation in our Cyclone II FPGA with a small program sending messages between all cores and when all messages are received a message is written to the UART connected to core zero. The test program is written in assembler for a 16-core system, but can easily be extended to 64 cores.

The resources shown on Table I are for one tile except the processor itself for the different network sizes that fits in our Cyclone II FPGA. The resource consumption is shown in lookup tables (LUT), registers (Reg), and memory bits (RAM). Along the resource consumption we also show the maximum frequency that the components can operate at. The numbers include the TDM time slot tables in the router and the network adapter.

TABLE II
RESOURCE CONSUMPTION AND SCHEDULE PERIOD OF THE TDM TIME SLOT TABLES FOR THE NETWORK ADAPTER (NA) AND THE ROUTER (R).

Cores	9	16	25	36	49	64	81
Period (clocks)	10	19	27	42	58	87	113
NA (LUT)	6	12	23	39	46	71	96
R (LUT)	12	28	38	63	78	127	173

TABLE III
A RELATIVE COMPARISON BETWEEN THE SINGLE CLOCKED AND THE DUAL CLOCKED IMPLEMENTATIONS.

Cores	LUT	Reg	RAM	Freq.
16	1.18	1.48	0.50	0.97
64	1.44	1.91	0.50	0.74

The resource consumption of the different entities of the system differs from core to core. The numbers in Table I are from the entities of core zero (upper left corner) for the given network size. Core zero is usually the largest entity, but it can differ from the different network sizes. The resource consumption of tiles is not uniform throughout the implemented systems.

The major reasons for the increase in the resource consumption on one network adapter and a router as the number of cores in the system grows are:

1) Bookkeeping of the status bits in the NA, increases both Reg and LUT count
2) The size of the routing tables grows, increases the LUT count
3) Buffering more data channels, increase the RAM size

The frequency appears to be close to constant for the network sizes we have synthesized, with small fluctuations from run to run of the synthesis. We expect the frequency to decrease when the systems size grows larger than what we have experimented with, because of the increase in bookkeeping. To avoid the frequency slowdown for larger systems the bookkeeping mechanism can be pipelined.

In Table II we present numbers for the resource consumption of the slot tables located in the routers and network adapters along with the period of the TDM schedule. The number of lookup tables increase proportional to the period of the TDM schedules. The numbers for these slot tables are not specific to our implementation of the network adapter, but more general for these types of TDM schedules.

In Table III we show the relative size of the double-clock design compared to the single clock design. The dual-clocked design was intended to be smaller as the router multiplexers are only half the size. However, only the RAM consumption is lower due to double clocking. The logic consumes more resources. The additional circuit in the NA for the packing and unpacking offset the reduction in the routers.

Furthermore there is also a relative decrease in frequency when using the double-clocked implementation. The disadvantages of the double-clocked implementation increase as the system size grows. On top the complexity of the dual clocked network is higher, thus making it more complicated to debug and harder to maintain.

Therefore, the double clocking of the network structure proves not to be beneficial. However, the double clocking of the communication memory reduced the number of block RAMs to a single one. Therefore, one design point can be a single clock per packet NoC and NA, but double-clock the block RAM.

VIII. DISCUSSION

In Section VI we have described an alternative implementation with double clocked routers. However, the results presented in Section VII show a higher resource consumption for this alternative. This is another indication that simplicity often wins, as the simple NA implementation was the smallest and fastest.

With higher number of nodes, the resource consumption of the routing tables increases per node. However, it has to be noted that the router tables start very small and therefore the increase is moderate. A complete NA and router with the routing tables for a 64-core system is still just 665 LCs.

If one would even like to reduce this size further, an application specific schedule can be used, i.e., a schedule where not all cores can communicate to all other cores. An application specific schedule can reduce the period length of the slot table schedule and thereby the resource consumption. It will also reduce the latency due to the shorter period.

An application specific schedule requires reconfigurable hardware. However, this extra hardware complexity could reduce the benefit of application specific schedule. In the natively reconfigurable hardware of an FPGA the application specific schedule can be part of the FPGA configuration and therefore be quite efficient. No programming of the schedule during runtime is needed.

The scheduler presented in [8] is capable of making such application specific schedules. These schedules have been tested on the implementation of our NoC.

As our target is to explore many-core architectures in medium sized FPGAs, we decided to use a small microprocessor, Leros [10], as the processing node. Leros is a 16-bit processor intended for small applications and utility functions similar to Xilinx's PicoBlaze [15]. Leros is an accumulator machine and uses on-chip memory for instructions and data. The data memory also contains a register file, i.e., the first 256 data locations can be directly addressed. Leros implements a two-stage pipeline and can be clocked faster than 100 MHz in Cyclone and Spartan devices.

Tiny microprocessors, like Leros, are usually programmed in assembler. Leros also comes with an assembler. However, to provide a higher level programming language, the muvium Java system has been adaped for Leros [3]. Muvium compiles Java class files into Leros assembler. The Java supported by Muvium/Leros is a *very* restricted subset. However, it is enough to write test and example programs for the presented NoC configuration.

IX. Conclusion

This paper presents a network-on-chip for real-time systems. The communication is scheduled statically in a time-division-multiplexed manner. This static schedule provides all-to-all communication for the chip-multiprocessor system. The resulting router is quite small and calls for an efficient implementation of the network adapter. The presented network adapter provides one word of buffer for each transmit and receive channel. By time-multiplexing a single on-chip memory it can be used to buffer input and output channels, even with one receive and one transmit word per clock cycle.

The presented network adapter is small and therefore is a good fit for the small and simple router. With a tiny processor we where able to build a 64-core system connected via a bidirectional torus network-on-chip in a medium sized FPGA from the low-cost series Altera Cyclone-II.

Acknowledgment

We would like to thank James Caska for his support on the Java bytecode compiler muvium for Leros. Furthermore, we thank Florian Brandner, who has helped us with the schedule generation for the router and NA tables. This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

Source Access

We provide the VHDL code of the NoC and Leros in open source. The design is vendor agnostic; only the Makefile has this board as default target. The default target of our design is the Altera DE2-70 board. The source can be found at

https://github.com/t-crest/s4noc

The source can be downloaded via a zip file or with git

```
git clone git://github.com/t-crest/s4noc.git
```

With a DE2-70 board attached, the whole design can be built and downloaded with a simple:

```
cd s4noc
make
```

See the Makefile for different build options. The build process on a Windows PC needs Altera Quartus, a Java compiler for the Leros application compilation, and a Cygwin environment for the make and git command.

References

[1] T. Bjerregaard and J. Sparsø. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *date*, pages 1226–1231. IEEE Computer Society Press, 2005.

[2] Florian Brandner and Martin Schoeberl. Static routing in symmetric real-time network-on-chips. In *Proceedings of the 20th International Conference on Real-Time and Network Systems (RTNS 2012)*, Pont a Mousson, France, November 2012.

[3] James Caska and Martin Schoeberl. Java dust: How small can embedded Java be? In *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2011)*, York, UK, Spetember 2011. ACM.

[4] Kees Goossens and Andreas Hansson. The AEthereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC 2010)*, pages 306 –311, 2010.

[5] Andreas Hansson, Mahesh Subburaman, and Kees Goossens. aelite: a flit-synchronous network on chip with composable and predictable services. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2009)*, pages 250–255, Leuven, Belgium, 2009.

[6] Hermann Kopetz and Günther Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[7] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2008)*, pages 120 –129, August 2008.

[8] Mark Ruvald Pedersen, Jaspur Højgaard, and Rasmus Bo Sørensen. Scheduling in a real-time network-on-chip. Technical report, https://github.com/t-crest/SNTs, 2012.

[9] Martin Schoeberl. A time-triggered network-on-chip. In *International Conference on Field-Programmable Logic and its Applications (FPL 2007)*, pages 377–382, Amsterdam, Netherlands, August 2007. IEEE.

[10] Martin Schoeberl. Leros: A tiny microcontroller for FPGAs. In *Proceedings of the 21st International Conference on Field Programmable Logic and Applications (FPL 2011)*, Chania, Crete, Greece, September 2011. IEEE Computer Society.

[11] Martin Schoeberl, Florian Brandner, Jens Sparsø, and Evangelia Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, Lyngby, Denmark, May 2012. IEEE.

[12] Radu Stefan, Anca Molnos, Angelo Ambrose, and Kees Goossens. A TDM NoC supporting QoS, multicast, and fast connection set-up. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE 2012)*, 2012.

[13] Daniel Wiklund and Dake Liu. SoCBUS: Switched network on chip for hard real time embedded systems. In *International Parallel and Distributed Processing Symposium (IPDPS'03)*, page 78a, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[14] Pascal T. Wolkotte, Gerard J.M. Smit, Gerard K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit switched network-on-chip. In *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

[15] Xilinx. PicoBlaze 8-bit embedded microcontroller user guide, 2010.

Appendix B

T-CREST NoC source code

This appendix contains the following files:

sc2ocp_noc.vhd is the a wrapper for the whole T-CREST NoC platform. The wrapper converts from an OCP interface on the NoC to an SimpCon interface on the JOP, the file starts on page 70

tb_sc2ocp.vhd is a testbench for the the T-CREST NoC wrapper. The testbench test the different parts of the T-CREST NoC address space, the file starts on page 72

test.vhd is a package with procedures for the testbench, the file starts on page 75

noc_node.vhd this file comes from the T-CREST NoC platform, the file describes a scratch pad and a network interface. The modifications we have made to this file were to merge the port to the scratch pad and the port to the network interface and add address decoding. The file starts on page 76

nAdapter.vhd this file comes from the T-CREST NoC platform, the file describes the network interface. The modifications we have made to this file were changes to the address decoding and the command types for the OCP interface. The file starts on page 81

Listing B.1: sc2ocp_noc.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.defs.all;
use work.sc_pack.all;


entity sc2ocp_noc is

port (
  clk    : in std_logic;
  reset    : in std_logic;

  sc_noc_out   : in sc_out_array_type(0 to (N*N)-1);
  sc_noc_in : out sc_in_array_type(0 to (N*N)-1)

);

end sc2ocp_noc;

architecture struct of sc2ocp_noc is
  -- NoC signals
  signal procM : procMasters;
  signal procS : procSlaves;
  -- SimpCon signals
  signal sc_noc_out_reg, sc_noc_out_next     : sc_out_array_type(0
      to (N*N)-1);
  signal sc_noc_in_reg, sc_noc_in_next     : sc_in_array_type(0 to (
      N*N)-1);

begin

  noc : entity work.noc
    port map(
      p_clk => clk,
      n_clk => clk,
      reset => reset,
      p_ports_in => procM,
      p_ports_out => procS
    );
    sc_noc_in <= sc_noc_in_reg;
--  ---------------------------------------------------------------------

-- Connecting the Noc the the processors
--  ---------------------------------------------------------------------


  process(sc_noc_out_reg, sc_noc_in_reg, sc_noc_out, procS)
  begin
    NoC2Proc : for i in 0 to N-1 loop
      innerNoC2Proc : for j in 0 to N-1 loop
```

```
48
        procM(i)(j).MCmd <= (sc_noc_out_reg(i*N+j).wr or
            sc_noc_out_reg(i*N+j).rd) & sc_noc_out_reg(i*N+j).wr;
50      procM(i)(j).MAddr <= std_logic_vector(to_unsigned(0,
            OCP_ADDR_WIDTH-SC_ADDR_SIZE)) & sc_noc_out_reg(i*N+j).
            address;
        procM(i)(j).MData <= sc_noc_out_reg(i*N+j).wr_data;
52
        sc_noc_out_next(i*N+j) <= sc_noc_out_reg(i*N+j);
54      sc_noc_in_next(i*N+j) <= sc_noc_in_reg(i*N+j);

56      if procS(i)(j).SCmdAccept = '1' and sc_noc_out_reg(i*N+j).
            wr = '1' then  -- The ackowledge of a write
          sc_noc_out_next(i*N+j).rd <= '0';
58        sc_noc_out_next(i*N+j).wr <= '0';
          sc_noc_out_next(i*N+j).address <= (others => '0');
60        sc_noc_out_next(i*N+j).wr_data <= (others => '0');
          sc_noc_in_next(i*N+j).rd_data <= (others => '0');
62        sc_noc_in_next(i*N+j).rdy_cnt <= (others => '0');
        end if;
64
        if procS(i)(j).SResp = '1' and sc_noc_out_reg(i*N+j).rd =
            '1' then -- The ackowledge of a read
66        sc_noc_out_next(i*N+j).rd <= '0';
          sc_noc_out_next(i*N+j).wr <= '0';
68        sc_noc_out_next(i*N+j).address <= (others => '0');
          sc_noc_out_next(i*N+j).wr_data <= (others => '0');
70        sc_noc_in_next(i*N+j).rd_data <= procS(i)(j).SData;
          sc_noc_in_next(i*N+j).rdy_cnt <= (others => '0');
72      end if;

74      if sc_noc_out(i*N+j).wr = '1' or sc_noc_out(i*N+j).rd = '1'
            then
          sc_noc_out_next(i*N+j) <= sc_noc_out(i*N+j);
76        sc_noc_in_next(i*N+j).rdy_cnt <= (others => '1');
        end if;
78

80    end loop ;
    end loop ; -- NoC2Proc
82  end process;

84  noc_reg:process (clk, reset) is
    begin
86    if rising_edge(clk) then
        for i in 0 to N-1 loop
88        for j in 0 to N-1 loop
            if reset = '1' then
90            sc_noc_out_reg(i*N+j).rd <= '0';
              sc_noc_out_reg(i*N+j).wr <= '0';
92            sc_noc_out_reg(i*N+j).address <= (others => '0');
              sc_noc_out_reg(i*N+j).wr_data <= (others => '0');
94            sc_noc_in_reg(i*N+j).rd_data <= (others => '0');
              sc_noc_in_reg(i*N+j).rdy_cnt <= (others => '1');
96          else
```

```
            sc_noc_in_reg(i*N+j) <= sc_noc_in_next(i*N+j);
98          sc_noc_out_reg(i*N+j) <= sc_noc_out_next(i*N+j);
          end if;
100     end loop;
      end loop;
102   end if;
  end process noc_reg;
104
end struct;
```

Listing B.2: tb_sc2ocp.vhd

```
1  library ieee;
   use ieee.std_logic_1164.all;
3  use ieee.NUMERIC_STD.all;
   use work.defs.all;
5  use work.sc_pack.all;
   use work.test.all;
7  use work.txt_util.all;

9  entity tb_sc2ocp is
   end tb_sc2ocp;
11
   architecture RTL of tb_sc2ocp is
13   constant CLOCK_PERIOD : time := 10 ns;
     constant RESET_TIME : time :=21 ns;
15   constant temp1 : std_logic_vector(31 downto 0) := DMA_P_MASK &
         std_logic_vector(to_unsigned(0,OCP_ADDR_WIDTH-ADDR_MASK_W));
     constant DMA_P_ADDR : natural := to_integer(unsigned(temp1));
17   constant temp2 : std_logic_vector(31 downto 0) := SPM_MASK &
         std_logic_vector(to_unsigned(0,OCP_ADDR_WIDTH-ADDR_MASK_W));
     constant SPM_ADDR : natural := to_integer(unsigned(temp2));
19   constant temp3 : std_logic_vector(31 downto 0) := DMA_MASK &
         std_logic_vector(to_unsigned(0,OCP_ADDR_WIDTH-ADDR_MASK_W));
     constant DMA_ADDR : natural := to_integer(unsigned(temp3));
21   constant temp4 : std_logic_vector(31 downto 0) := ST_MASK &
         std_logic_vector(to_unsigned(0,OCP_ADDR_WIDTH-ADDR_MASK_W));
     constant ST_ADDR : natural := to_integer(unsigned(temp4));
23   signal clk : std_logic;
     signal reset : std_logic;
25   signal sc_noc_out : sc_out_array_type(0 to (N*N)-1);
     signal sc_noc_in : sc_in_array_type(0 to (N*N)-1);
27
     alias sc_in is sc_noc_in(0);
29   alias sc_out is sc_noc_out(0);

31   alias sc_in_2 is sc_noc_in(2);
     alias sc_out_2 is sc_noc_out(2);
33


35   alias sc_in_4 is sc_noc_in(4);
     alias sc_out_4 is sc_noc_out(4);
37


39 begin
```

```
  -- Clock and reset
41  clock_generator : clockGen(clk,CLOCK_PERIOD);
    reset_generator : resetGen(reset,RESET_TIME);
43
    dut: entity work.sc2ocp_noc
45    port map(clk         => clk,
               reset       => reset,
47             sc_noc_out => sc_noc_out,
               sc_noc_in  => sc_noc_in);
49
    stimuli_process : process
51  variable result : natural;
    begin
53    for i in 0 to N-1 loop
        for j in 0 to N-1 loop
55        if reset = '1' then
            sc_noc_out(i*N+j).rd <= '0';
57          sc_noc_out(i*N+j).wr <= '0';
            sc_noc_out(i*N+j).address <= (others => '0');
59          sc_noc_out(i*N+j).wr_data <= (others => '0');
          end if;
61      end loop;
      end loop;
63    wait until reset = '0';
      wait for 11 ns;
65--    report "--------- Testing DMA P ---------";
--    for i in 0 to 10 loop
67--      wait until rising_edge(clk);
--      sc_write(clk,DMA_P_ADDR+i,i,sc_out,sc_in,5);
69--      wait for CLOCK_PERIOD;
--    end loop;
71--
--    for i in 0 to 10 loop
73--      wait until rising_edge(clk);
--      sc_read(clk,DMA_P_ADDR+i,result,sc_out,sc_in,6);
75--      assert result = i report "Wrong result read out!" severity
    failure;
--      wait for CLOCK_PERIOD;
77--    end loop;
--    report "--------- DMA test passed ---------";
79
      wait until rising_edge(clk);
81
      sc_write(clk,DMA_P_ADDR,54,sc_out_4,sc_in_4,5);
83    wait until rising_edge(clk);
      sc_write(clk,ST_ADDR+2,16,sc_out_4,sc_in_4,5);
85    -- Write to spm
      wait until rising_edge(clk);
87    sc_write(clk,SPM_ADDR+22,3,sc_out_4,sc_in_4,5);
      wait until rising_edge(clk);
89    sc_write(clk,SPM_ADDR+22+1,6,sc_out_4,sc_in_4,5);
      wait until rising_edge(clk);
91    sc_write(clk,SPM_ADDR+22+2,9,sc_out_4,sc_in_4,5);
      wait until rising_edge(clk);
93    sc_write(clk,SPM_ADDR+22+3,12,sc_out_4,sc_in_4,5);
```

```
    wait until rising_edge(clk);
    sc_write(clk,SPM_ADDR+22+4,15,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);
    sc_write(clk,SPM_ADDR+22+5,18,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);
    sc_write(clk,SPM_ADDR+22+6,21,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);
    sc_write(clk,SPM_ADDR+22+7,24,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);
    sc_write(clk,SPM_ADDR+22+8,27,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);
    sc_write(clk,SPM_ADDR+22+9,30,sc_out_4,sc_in_4,5);
    -- Setup dma
    wait until rising_edge(clk);
    sc_write(clk,DMA_ADDR+1,1441922,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);
    sc_write(clk,DMA_ADDR,32778,sc_out_4,sc_in_4,5);
    wait until rising_edge(clk);



    sc_write(clk,SPM_ADDR+224,4,sc_out,sc_in,5);
    wait until rising_edge(clk);
    sc_read(clk,SPM_ADDR+224,result,sc_out,sc_in,5);
    assert result = 4 report "Something is very wrong!" severity
        failure;
    wait until rising_edge(clk);

    wait for 300 ns;


--    test_function(clk,sc_out,sc_in,DMA_P_ADDR,CLOCK_PERIOD);
--    test_function(clk,sc_out,sc_in,SPM_ADDR,CLOCK_PERIOD);
--    test_function(clk,sc_out,sc_in,DMA_ADDR,CLOCK_PERIOD);
--    test_function(clk,sc_out,sc_in,ST_ADDR,CLOCK_PERIOD);

--    report "-------- Testing SPM --------";
--    for i in 0 to 10 loop
--      wait until rising_edge(clk);
--      sc_write(clk,SPM_ADDR+i,i,sc_out,sc_in,5);
--      wait for CLOCK_PERIOD;
--    end loop;
--
--    for i in 0 to 10 loop
--      wait until rising_edge(clk);
--      sc_read(clk,SPM_ADDR+i,result,sc_out,sc_in,6);
--      assert result = i report "Wrong result read out!" severity
    failure;
--      wait for CLOCK_PERIOD;
--    end loop;
--    report "-------- SPM test passed --------";

    wait;
  end process;
```

```
147
end architecture RTL;
```

Listing B.3: test.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.sc_pack.all;

package test is
  procedure clockGen (signal clk : out std_logic; constant period :
      in time);
  procedure resetGen (signal reset : out std_logic ; constant
      reset_time : in time);
  procedure test_function (signal clk : in std_logic; signal sc_out
      : out sc_out_type; signal sc_in : in sc_in_type; constant
      addr : in natural; constant period : in time);

end package test;

package body test is
  procedure clockGen (signal clk : out std_logic; constant period :
      in time) is
  variable clk_int : std_logic := '0';
  begin -- Careful this process runs forever:
    loop
      clk_int := not clk_int;
      clk <= clk_int;
      wait for period/2;
    end loop;
  end;

  procedure resetGen (signal reset : out std_logic ; constant
      reset_time : in time) is
  begin -- Careful this process runs forever:
    reset <= '1';
    wait for reset_time;
    reset <= '0';
    wait;
  end;


  procedure test_function (signal clk : in std_logic;
              signal sc_out : out sc_out_type;
              signal sc_in : in sc_in_type;
              constant addr : in natural;
              constant period : in time) is
  variable result : natural;
  begin
    report "-------- Testing " & addr'simple_name & " --------";
    for i in 0 to 10 loop
      wait until rising_edge(clk);
      sc_write(clk,addr+i,i,sc_out,sc_in,5);
      wait for period;
    end loop;
```

```
46      for i in 0 to 10 loop
          wait until rising_edge(clk);
48        sc_read(clk,addr+i,result,sc_out,sc_in,6);
          assert result = i report "Wrong result read out!" severity
              failure;
50        wait for period;
        end loop;
52      report "--------- " & addr'simple_name & " test passed ---------"
          ;
    end;
54
end package body test;
```

Listing B.4: noc_node.vhd

```
1  library ieee;
   use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
   use work.defs.all;
5
7  entity noc_node is
   port (
9    p_clk    : std_logic;
     n_clk    : std_logic;
11   reset    : std_logic;

13   proc_in    : in ocp_master;
     proc_out   : out ocp_slave;
15
     inNorth    : in network_link;
17   inSouth    : in network_link;
     inEast     : in network_link;
19   inWest     : in network_link;

21   outNorth   : out network_link;
     outSouth   : out network_link;
23   outEast    : out network_link;
     outWest    : out network_link
25
   );
27
   end noc_node;
29
   architecture struct of noc_node is
31
   ---------------------------------component declarations
       -------------------------------
33 --2 spms
   component bram_tdp is
35
   generic (
37     DATA     : integer := 32;
       ADDR     : integer := 14
```

```
39 );

41 port (
   -- Port A
43      a_clk    : in  std_logic;
        a_wr     : in  std_logic;
45      a_addr   : in  std_logic_vector(ADDR-1 downto 0);
        a_din    : in  std_logic_vector(DATA-1 downto 0);
47      a_dout   : out std_logic_vector(DATA-1 downto 0);

49 -- Port B
        b_clk    : in  std_logic;
51      b_wr     : in  std_logic;
        b_addr   : in  std_logic_vector(ADDR-1 downto 0);
53      b_din    : in  std_logic_vector(DATA-1 downto 0);
        b_dout   : out std_logic_vector(DATA-1 downto 0)
55 );
   end component;
57
   --1 na
59 component nAdapter is


61
   port (
63 -- General
     na_clk     : in  std_logic;
65   na_reset   : in  std_logic;

67 -- Processor Ports
   -- DMA Configuration Port - OCP
69   proc_in    : in ocp_master;
     proc_out   : out ocp_slave;
71
   -- SPM Data Port - OCP?
73   spm_in     : in ocp_slave_spm;
     spm_out    : out ocp_master_spm;
75
   -- Network Ports
77 -- Incoming Port
     pkt_in     : in  network_link;
79
   -- Outgoing Port
81   pkt_out    : out network_link

83 );
   end component;
85
   --1 router
87 component router is
     port (
89     clk : in std_logic;
       reset : in std_logic;
91     inPort  : in routerPort;
       outPort : out routerPort
93   );
```

```
end component;

-------------------------------------signal declarations
    ---------------------------------

signal ip_to_net  : network_link;
signal net_to_ip  : network_link;

signal spm_to_net : ocp_slave_spm;
signal net_to_spm : ocp_master_spm;

signal proc_spm_out_h : ocp_slave;
signal proc_spm_out_l : ocp_slave;
signal proc_noc_in  : ocp_master;
signal proc_noc_out : ocp_slave;

signal spm_h_access : std_logic;
signal spm_l_access : std_logic;
signal dma_access : std_logic;

signal spm_h_wr : std_logic;
signal spm_l_wr : std_logic;

signal rd_rdy, next_rd_rdy : std_logic;

type proc_sel is (spm_h_sel, spm_l_sel, dma_sel, none);
signal proc_out_sel : proc_sel;
signal cmd_acc : std_logic;

begin


-- High SPM instance
spm_h : bram_tdp
generic map (DATA=>DATA_WIDTH, ADDR => SPM_ADDR_WIDTH-1)
port map (a_clk => p_clk,
  a_wr => spm_h_wr,
  a_addr => proc_in.MAddr(SPM_ADDR_WIDTH-1 downto 1),
  a_din => proc_in.MData,
  a_dout => proc_spm_out_h.SData,
  b_clk => n_clk,
  b_wr => net_to_spm.MCmd(0),
  b_addr => net_to_spm.MAddr(SPM_ADDR_WIDTH-2 downto 0),
  b_din => net_to_spm.MData(63 downto 32),
  b_dout => spm_to_net.SData(63 downto 32));

spm_h_wr <= proc_in.MCmd(0) and spm_h_access;

-- Low SPM instance
spm_l : bram_tdp
generic map (DATA => DATA_WIDTH, ADDR => SPM_ADDR_WIDTH-1)
port map (a_clk => p_clk,
  a_wr => spm_l_wr,
  a_addr => proc_in.MAddr(SPM_ADDR_WIDTH-1 downto 1),
  a_din => proc_in.MData,
```

```
    a_dout => proc_spm_out_l.SData,
    b_clk => n_clk,
    b_wr => net_to_spm.MCmd(0),
    b_addr => net_to_spm.MAddr(SPM_ADDR_WIDTH-2 downto 0),
    b_din => net_to_spm.MData(31 downto 0),
    b_dout => spm_to_net.SData(31 downto 0));

spm_l_wr <= proc_in.MCmd(0) and spm_l_access;

-- NA instance
na : nAdapter
port map(
  -- General
  na_clk=>n_clk,
  na_reset=>reset,

  -- Processor Ports
  -- DMA Configuration Port - OCP
  proc_in=>proc_noc_in,
  proc_out=>proc_noc_out,

  -- SPM Data Port - OCP?
  spm_in=>spm_to_net,
  spm_out=>net_to_spm,

  -- Network Ports
  -- Incoming Port
  pkt_in=>net_to_ip,

  -- Outgoing Port
  pkt_out=>ip_to_net
);

  proc_noc_in.MData <= proc_in.MData;
  proc_noc_in.MAddr <= proc_in.MAddr;
  proc_noc_in.MCmd(1) <= proc_in.MCmd(1) and dma_access;
  proc_noc_in.MCmd(0) <= proc_in.MCmd(0) and dma_access;

-- router instance
r : router
port map (
  clk => n_clk,
  reset => reset,
  inPort(0) => inSouth,
  inPort(1) => inWest,
  inPort(2) => inNorth,
  inPort(3) => inEast,
  inPort(4) => ip_to_net,
  outPort(0) => outSouth,
  outPort(1) => outWest,
  outPort(2) => outNorth,
  outPort(3) => outEast,
  outPort(4) => net_to_ip
);
```

```
203 proc_logic: process(proc_in, proc_spm_out_h, proc_spm_out_l, rd_rdy
    , proc_noc_out, dma_access)
    begin
205   spm_h_access <= '0';
      spm_l_access <= '0';
207   dma_access <= '0';
      next_rd_rdy <= '0';
209   cmd_acc <= '0';
      proc_out_sel <= none;
211
      if proc_in.MAddr(OCP_ADDR_WIDTH-1 downto SPM_ADDR_WIDTH) =
          SPM_MASK & std_logic_vector(to_unsigned(0,OCP_ADDR_WIDTH-
          ADDR_MASK_W-(SPM_ADDR_WIDTH))) then -- Access to the spm port
213     if proc_in.MAddr(0) = '0' then -- Access high spm
          spm_h_access <= '1';
215       proc_out_sel <= spm_h_sel;
        else -- Access low spm
217       spm_l_access <= '1';
          proc_out_sel <= spm_l_sel;
219     end if;
        -- Write operation
221     if proc_in.MCmd(0) = '1' then
          cmd_acc <= '1';
223     end if;
        -- Read operation
225     if proc_in.MCmd(1) = '1' and proc_in.MCmd(0) = '0' then
          next_rd_rdy <= '1';
227     end if;
      else -- Access to the dma configuration port
229     dma_access <= '1';
        proc_out_sel <= dma_sel;
231   end if;
    end process;
233
    process(proc_out_sel, proc_spm_out_h, proc_spm_out_l, proc_noc_out,
         rd_rdy, cmd_acc)
235 --process(all)
      begin
237   proc_out.SData <= (others => '0');
      proc_out.SResp <= '0';
239   proc_out.SCmdAccept <= '0';
      -- Proc_out mux
241
      case proc_out_sel is
243     when spm_h_sel =>
          proc_out.SData <= proc_spm_out_h.SData;
245       proc_out.SCmdAccept <= cmd_acc;
          proc_out.SResp <= rd_rdy;
247     when spm_l_sel =>
          proc_out.SData <= proc_spm_out_l.SData;
249       proc_out.SCmdAccept <= cmd_acc;
          proc_out.SResp <= rd_rdy;
251     when dma_sel =>
          proc_out <= proc_noc_out;
253     when none =>
```

```
      proc_out.SData <= (others => '0');
      proc_out.SResp <= '0';
      proc_out.SCmdAccept <= '0';
  end case;

end process;

process(p_clk)
begin
  if rising_edge(p_clk) then
    if reset = '1' then
      rd_rdy <= '0';
    else
      rd_rdy <= next_rd_rdy;
    end if;
  end if;
end process;

end struct;
```

Listing B.5: nAdapter.vhd

```
--
-- Copyright Technical University of Denmark. All rights reserved.
-- This file is part of the T-CREST project.
--
-- Redistribution and use in source and binary forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
--    1. Redistributions of source code must retain the above copyright notice,
--       this list of conditions and the following disclaimer.
--
--    2. Redistributions in binary form must reproduce the above copyright
--       notice, this list of conditions and the following disclaimer in the
--       documentation and/or other materials provided with the distribution.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS
-- OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
-- OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN
-- NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY
-- DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
-- (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
```

```
-- LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
-- ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
-- (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
-- THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
--
-- The views and conclusions contained in the software and documentation are
-- those of the authors and should not be interpreted as representing official
-- policies, either expressed or implied, of the copyright holder.
--


--
-----------------------------------------------------------------------------

-- Network Adaptor (NI) for the TDM NoC with DMAs.
--
-- Author: Evangelia Kasapaki
--
-----------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.defs.all;


entity nAdapter is

port (
-- General
  na_clk     : in  std_logic;
  na_reset   : in  std_logic;

-- Processor Ports
-- DMA Configuration Port - OCP
  proc_in    : in ocp_master;
  proc_out   : out ocp_slave;

-- SPM Data Port - OCP?
  spm_in     : in ocp_slave_spm;
  spm_out    : out ocp_master_spm;

-- Network Ports
-- Incoming Port
  pkt_in     : in  network_link;

-- Outgoing Port
  pkt_out    : out network_link
```

```
66
);
68 end nAdapter;

70
architecture rtl of nAdapter is
72

74 ---------------------------------- signal declarations
    ----------------------------------
signal slt_index   : std_logic_vector(SLT_WIDTH-1 downto 0);
76 signal sc_en     : std_logic;
signal slt_en    : std_logic;
78
signal slt_entry   : std_logic_vector(DMA_IND_WIDTH downto 0);
80 signal vld_slt     : std_logic;

82 signal config    : std_logic_vector(3 downto 0);
signal config_reg : std_logic_vector(4 downto 0);
84
signal dma_index   : std_logic_vector(DMA_IND_WIDTH-1 downto 0);
86 signal dma_entry   : std_logic_vector(DMA_WIDTH-1 downto 0);
signal dma_entry_updated: std_logic_vector(DMA_WIDTH-1 downto 0);
88
signal dma_ren     : std_logic_vector(2 downto 0);
90 signal dma_wen     : std_logic_vector(2 downto 0);
signal dma_waddr   : std_logic_vector(DMA_IND_WIDTH-1 downto 0);
92 signal dma_wdata   : std_logic_vector(DMA_WIDTH-1 downto 0);
signal dma_raddr   : std_logic_vector(DMA_IND_WIDTH-1 downto 0);
94 signal dma_rdata   : std_logic_vector(DMA_WIDTH-1 downto 0);

96 signal dma_cnt     : unsigned(BLK_CNT-1 downto 0);
signal dma_cnt_new   : unsigned(BLK_CNT-1 downto 0);
98 signal dma_rp_new  : unsigned(SPM_ADDR_WIDTH-1 downto 0);
signal dma_wp_new  : unsigned(SPM_ADDR_WIDTH-1 downto 0);
100
signal dma_ctrl    : std_logic;
102 signal dma_ctrl_new : std_logic_vector(1 downto 0);
signal done    : std_logic;
104 signal done_new    : std_logic;

106 signal state_cnt   : unsigned(1 downto 0);
signal val     : unsigned(1 downto 0);
108
signal dIn_h     : std_logic_vector(DATA_WIDTH-1 downto 0);
110 signal dOut_l    : std_logic_vector(DATA_WIDTH-1 downto 0);

112 signal address     : std_logic_vector(SPM_ADDR_WIDTH-1 downto 0);
signal m_cmd     : std_logic;
114
signal dOutreg_ld : std_logic;
116 signal dInreg_ld  : std_logic;
signal adreg_ld    : std_logic;
118
signal mux_out     : std_logic_vector(DATA_WIDTH-1 downto 0);
```

```
120 signal hdr_phit    : std_logic_vector(DATA_WIDTH-1 downto 0);

122 signal phitOut     : std_logic_vector(PHIT_WIDTH-1 downto 0);
    signal phitIn      : std_logic_vector(PHIT_WIDTH-1 downto 0);
124
    signal pkt_ctrl    : std_logic;
126 signal dma_ctrl_reg : std_logic;
    signal ctrlOutreg_ld  : std_logic;
128

130 ------------------------------------ Components declarations
        ------------------------------------
    component counter
132   generic (
        WIDTH : integer
134   );
      port (
136     clk    : in std_logic ;
        reset    : in std_logic ;
138     enable   : in std_logic;
        cnt    : out std_logic_vector(WIDTH-1 downto 0)
140   );
    end component;
142
    component dma_sdp
144   generic (
        DATA     : integer := 64;
146     ADDR     : integer := 2
      );
148   port (
        clk    : in std_logic;
150     reset    : in std_logic;

152     ren    : in std_logic_vector(2 downto 0);
        wen    : in std_logic_vector(2 downto 0);
154     waddr      : in std_logic_vector(ADDR-1 downto 0);
        wdata    : in std_logic_vector(DATA-1 downto 0);
156     raddr      : in std_logic_vector(ADDR-1 downto 0);
        rdata    : out std_logic_vector(DATA-1 downto 0)
158   );
    end component;
160
    component bram
162   generic (
        DATA     : integer := 32;
164     ADDR     : integer := 14
      );
166
      port (
168     clk    : in std_logic ;
        rd_addr : in std_logic_vector(ADDR-1 downto 0);
170     wr_addr : in std_logic_vector(ADDR-1 downto 0);
        wr_data : in std_logic_vector(DATA-1 downto 0);
172     wr_ena  : in std_logic ;
        rd_data : out std_logic_vector(DATA-1 downto 0)
```

```
174   );
end component;
176
178
begin
180
-- component instantiations
-----------------------------------------------------------------
182 -- Slot Counter
  slt_cnt : counter
184   generic map ( WIDTH=>SLT_WIDTH )
    port map ( clk=>na_clk, reset=>na_reset, enable=>sc_en, cnt=>
        slt_index );
186
-- DMA Table - simple block ram
188 dma_table : dma_sdp
    generic map ( DATA=>DMA_WIDTH, ADDR=>DMA_IND_WIDTH )
190   port map (clk=>na_clk, reset=>na_reset,
      ren => dma_ren,
192   wen => dma_wen,
      waddr => dma_waddr,
194   wdata => dma_wdata,
      raddr => dma_raddr,
196   rdata => dma_rdata
    );
198

200 slt_en <= '1' when config=ST_ACCESS and proc_in.MCmd(0)='1'
      else '0';
202 -- Slot Table
  slt_table : bram
204   generic map ( DATA=>DMA_IND_WIDTH+1, ADDR=>SLT_WIDTH )
    port map (clk => na_clk,
206   rd_addr => slt_index,
      wr_addr => proc_in.MAddr(SLT_WIDTH-1 downto 0),
208   wr_data => proc_in.MData(DMA_IND_WIDTH downto 0),
      wr_ena => slt_en,
210   rd_data => slt_entry
    );
212
  dma_index <= slt_entry(DMA_IND_WIDTH-1 downto 0);
214 vld_slt <= slt_entry(DMA_IND_WIDTH);

216 -- configuration interface
    ------------------------------------------------------------------------

  -- decode inputs ---------------------------------------
218 -- address map decoding
  ocp_decode : process (proc_in.MAddr) begin
220   config <= CNULL;
    -- ST configuration
222   if proc_in.MAddr(OCP_ADDR_WIDTH-1 downto OCP_ADDR_WIDTH-
        ADDR_MASK_W)=ST_MASK then
      config <= ST_ACCESS;
```

```
224     -- DMA 3/route configuration
        elsif proc_in.MAddr(OCP_ADDR_WIDTH-1 downto OCP_ADDR_WIDTH-
            ADDR_MASK_W)=DMA_P_MASK then
226       config <= DMA_R_ACCESS;
        -- DMA 1,2 configuration
228     elsif proc_in.MAddr(OCP_ADDR_WIDTH-1 downto OCP_ADDR_WIDTH-
            ADDR_MASK_W)=DMA_MASK
            and proc_in.MAddr(0)='0' then
230       config <= DMA_H_ACCESS;
        elsif proc_in.MAddr(OCP_ADDR_WIDTH-1 downto OCP_ADDR_WIDTH-
            ADDR_MASK_W)=DMA_MASK
232         and proc_in.MAddr(0)='1' then
          config <= DMA_L_ACCESS;
234     -- not configuration
        else
236       config <= CNULL;
        end if;
238   end process;

240
  -- build outputs ------------------------------------------
242   -- ocp data response
    ocp_response : process ( state_cnt, config_reg, dma_rdata) begin
244     proc_out.SData <= (others=>'0');
        proc_out.SResp <= '0';
246
        case state_cnt is
248     when "00" =>
          if config_reg=('1' & DMA_R_ACCESS) or config_reg=('1' &
              DMA_H_ACCESS) or config_reg=('1' & DMA_L_ACCESS) then
250         proc_out.SData <= dma_rdata(OCP_DATA_WIDTH-1 downto 0);
            proc_out.SResp <= '1';
252       end if;
        when "01" =>
254       if config_reg=('1' & DMA_R_ACCESS) or config_reg=('1' &
              DMA_H_ACCESS) or config_reg=('1' & DMA_L_ACCESS) then
            proc_out.SData <= dma_rdata(OCP_DATA_WIDTH-1 downto 0);
256         proc_out.SResp <= '1';
          end if;
258     when others =>
          proc_out.SData <= (others=>'0');
260       proc_out.SResp <= '0';
        end case;
262   end process;

264

266 -- SPM interface
    ------------------------------------------------------------

    --- construct SPM interface signals -->ocp???
268   spm_interface : process (state_cnt, pkt_ctrl, dma_entry, address)
          begin
      if state_cnt = "00" and pkt_ctrl = '1' then
270     spm_out.MCmd <= "11";
```

```
      spm_out.MAddr <= std_logic_vector(to_unsigned(0,
          OCP_ADDR_WIDTH-(SPM_ADDR_WIDTH-1))) & address(
          SPM_ADDR_WIDTH-1 downto 1);
    else
      spm_out.MCmd <= "00";
      spm_out.MAddr <= x"0000" & '0' & dma_entry(47 downto 33);
    end if;
  end process;
  spm_out.MData(SPM_DATA_WIDTH-1 downto DATA_WIDTH) <= dIn_h;
  spm_out.MData(DATA_WIDTH-1 downto 0) <= phitIn(DATA_WIDTH-1
      downto 0);

-- network interface
  ------------------------------------------------------------------------


-- input pkt control-----------------------------------
-- decode incoming packet
  pkt_ctrl <= phitIn(PHIT_WIDTH-1) or phitIn(PHIT_WIDTH-2) or
      phitIn(PHIT_WIDTH-3);


-- output pkt construction-----------------------------
-- build hdr phit
  hdr_phit <= dma_entry(DATA_WIDTH-1 downto 0);

-- mux to choose outgoing data
  nout_select : process(state_cnt, dma_ctrl, dma_ctrl_reg, hdr_phit
      , spm_in.SData(63 downto 32), dOut_l) begin
    case state_cnt is
    when "00" =>
      if dma_ctrl_reg='1'then
        --mux on 1 (data1)
        mux_out <= spm_in.SData(63 downto 32) after PDELAY;
      else
        mux_out <= (others=>'0') after PDELAY;
      end if;
    when "01" =>
      if dma_ctrl_reg='1'then
        --mux on 2 (data2)
        mux_out <= dOut_l after PDELAY;
      else
        mux_out <= (others=>'0') after PDELAY;
      end if;
    when "10" =>
      if dma_ctrl='1' then
        --mux on 0 (hdr)
        mux_out <= hdr_phit after PDELAY;
      else
        mux_out <= (others=>'0') after PDELAY;
      end if;
    when others =>
      mux_out <= (others=>'0');
    end case;
  end process;

```

```
-- build outgoing packet
  --control bits
  phitOut(PHIT_WIDTH-1) <= state_cnt(1) and dma_ctrl; --hdr
  phitOut(PHIT_WIDTH-2) <= not (state_cnt(0) or state_cnt(1)) and
      dma_ctrl_reg; --md
  phitOut(PHIT_WIDTH-3) <= state_cnt(0) and dma_ctrl_reg; --eop
  --hdr or payload
  phitOut(PHIT_WIDTH-4 downto 0) <= mux_out;


-- DMA signals
  ------------------------------------------------------------------------------

  dma_state_control : process (state_cnt, config, proc_in, dma_ctrl
      , dma_index, dma_entry_updated, dma_rdata) begin
    dma_waddr <= (others => '0');
    dma_wdata <= (others => '0');
    dma_wen <= (others => '0');
    dma_raddr <= (others => '0');
    dma_ren <= (others => '0');
    proc_out.SCmdAccept <= '0';
    dma_entry <= (others => '0');

    case state_cnt is
    when "00" =>
      -- configuration write
      if proc_in.MCmd(0)='1' then
        if config=DMA_R_ACCESS then
          dma_waddr <= proc_in.MAddr(DMA_IND_WIDTH-1 downto 0);
          dma_wdata <= x"00000000" & proc_in.MData;
          dma_wen <= config(2 downto 0);
          proc_out.SCmdAccept <= '1';
        elsif config=DMA_H_ACCESS then
          dma_waddr <= proc_in.MAddr(DMA_IND_WIDTH downto 1);
          dma_wdata <= proc_in.MData(BANK0_W-1 downto 0) & x"
              000000000000";
          dma_wen <= config(2 downto 0);
          proc_out.SCmdAccept <= '1';
        elsif config=DMA_L_ACCESS then
          dma_waddr <= proc_in.MAddr(DMA_IND_WIDTH downto 1);
          dma_wdata <= x"0000" & proc_in.MData & x"0000";
          dma_wen <= config(2 downto 0);
          proc_out.SCmdAccept <= '1';
        elsif config=ST_ACCESS then
          dma_waddr <= (others => '0');
          dma_wdata <= (others => '0');
          dma_wen <= (others => '0');
          proc_out.SCmdAccept <= '1';
        else
          dma_waddr <= (others => '0');
          dma_wdata <= (others => '0');
          dma_wen <= (others => '0');
          proc_out.SCmdAccept <= '0';
        end if;
      --configuration read or no read
```

```
      else
370     if config=DMA_R_ACCESS then
          dma_raddr <= proc_in.MAddr(DMA_IND_WIDTH-1 downto 0);
372       dma_ren <= config(2 downto 0);
          --build ocp slave signals
374       proc_out.SCmdAccept <= '1';
        elsif config=DMA_H_ACCESS or config=DMA_L_ACCESS then
376       dma_raddr <= proc_in.MAddr(DMA_IND_WIDTH downto 1);
          dma_ren <= config(2 downto 0);
378       --build ocp read data
          proc_out.SCmdAccept <= '1';
380     else
          dma_waddr <= (others => '0');
382       dma_wdata <= (others => '0');
          dma_wen <= (others => '0');
384       --build ocp read data
          proc_out.SCmdAccept <= '0';
386     end if;
      end if;
388
    when "01" =>
390   if proc_in.MCmd(0)='1' then
        if config=DMA_R_ACCESS then
392       dma_waddr <= proc_in.MAddr(DMA_IND_WIDTH-1 downto 0);
          dma_wdata <= x"00000000" & proc_in.MData;
394       dma_wen <= config(2 downto 0);
          proc_out.SCmdAccept <= '1';
396     elsif config=DMA_H_ACCESS then
          dma_waddr <= proc_in.MAddr(DMA_IND_WIDTH downto 1);
398       dma_wdata <= proc_in.MData(BANK0_W-1 downto 0) & x"
              000000000000";
          dma_wen <= config(2 downto 0);
400       proc_out.SCmdAccept <= '1';
        elsif config=DMA_L_ACCESS then
402       dma_waddr <= proc_in.MAddr(DMA_IND_WIDTH downto 1);
          dma_wdata <= x"0000" & proc_in.MData & x"0000";
404       dma_wen <= config(2 downto 0);
          proc_out.SCmdAccept <= '1';
406     elsif config=ST_ACCESS then
          dma_waddr <= (others => '0');
408       dma_wdata <= (others => '0');
          dma_wen <= (others => '0');
410       proc_out.SCmdAccept <= '1';
        else
412       dma_waddr <= (others => '0');
          dma_wdata <= (others => '0');
414       dma_wen <= (others => '0');
          proc_out.SCmdAccept <= '0';
416     end if;
      end if;
418   dma_raddr <= dma_index;
      dma_ren <= "111";
420
    when "10" =>
422   dma_waddr <= dma_index;
```

```
        dma_wdata <= dma_entry_updated;
424     if dma_ctrl='1' then
          dma_wen <= "110";
426     else
          dma_wen <= "000";
428     end if;

430     if proc_in.MCmd(0)='0' then
          if config=DMA_R_ACCESS then
432         dma_raddr <= proc_in.MAddr(DMA_IND_WIDTH-1 downto 0);
            dma_ren <= config(2 downto 0);
434         proc_out.SCmdAccept <= '1';
          elsif config=DMA_H_ACCESS or config=DMA_L_ACCESS then
436         dma_raddr <= proc_in.MAddr(DMA_IND_WIDTH downto 1);
            dma_ren <= config(2 downto 0);
438         proc_out.SCmdAccept <= '1';
          else
440         dma_raddr <= (others => '0');
            dma_ren <= (others => '0');
442         proc_out.SCmdAccept <= '0';
          end if;
444     end if;

446     if vld_slt='1' then
          dma_entry <= dma_rdata;
448     else
          dma_entry <= (others=>'0');
450     end if;

452   when others =>
        dma_waddr <= (others => '0');
454     dma_wdata <= (others => '0');
        dma_wen <= (others => '0');
456     dma_raddr <= (others => '0');
        dma_ren <= (others => '0');
458     proc_out.SCmdAccept <= '0';
      end case;
460 end process;

462
-- DMA control 0 decode dma entry
464 --valid dma entry and transfer not done yet
  dma_ctrl <= dma_entry(DMA_WIDTH-1) and (not dma_entry(DMA_WIDTH
      -2));
466 dma_cnt <= unsigned(dma_entry(61 downto 48));

468 -- update dma entry fields
  dma_cnt_new <= dma_cnt - 2;
470 dma_rp_new <= unsigned(dma_entry(SPM_ADDR_WIDTH-1+32 downto 32))
      + 2;
  dma_wp_new <= unsigned(dma_entry(SPM_ADDR_WIDTH-1+16 downto 16))
      + 2;
472
  done <= '1' when dma_cnt_new=0
474   else '0';
```

```
  done_new <= dma_entry(DMA_WIDTH-1) and done;
476  dma_ctrl_new <= dma_entry(DMA_WIDTH-1) & done_new;

478 -- updated dma entry
  dma_entry_updated <= (dma_ctrl_new &
480         std_logic_vector(dma_cnt_new) &
          "0000000" & std_logic_vector(dma_rp_new) &
482         "0000000" & std_logic_vector(dma_wp_new) &
          dma_entry(15 downto 0)) when dma_ctrl='1' else
484         dma_entry;


486

488 -- control FSM - just counter
    ------------------------------------------------------------------------


  val <= state_cnt + 1;
490  process (na_reset, na_clk)
  begin
492    if na_reset='1' then
        state_cnt <= (others=>'0') after PDELAY;
494    elsif rising_edge(na_clk) then
        if state_cnt="10" then
496       state_cnt <= (others=>'0') after PDELAY;
        else
498       state_cnt <= val after PDELAY;
        end if;
500    end if;
  end process;
502

  reg_control : process(state_cnt)
504  begin
  dOutreg_ld <= '0';
506  adreg_ld <= '0';
  dInreg_ld <= '0';
508  ctrlOutreg_ld <= '0';
  sc_en <= '0';
510

  if state_cnt="00" then
512    --ld dataOut_reg
    dOutreg_ld <= '1';
514  elsif state_cnt="01" then
    --ld addr_reg
516    adreg_ld <= '1';
  elsif state_cnt="10" then
518    --load dataIn_reg
    dInreg_ld <= '1';
520    ctrlOutreg_ld <='1';
    --update slt_cnt
522    sc_en <= '1';
  else
524    dOutreg_ld <= '0';
    adreg_ld <= '0';
526    dInreg_ld <= '0';
    ctrlOutreg_ld <= '0';
```

```
528     sc_en <= '0';
530   end if;
532   end process;
534
--- registers
536   registers : process(na_clk, na_reset) begin
        if na_reset='1' then
538       dma_ctrl_reg <= '0' after PDELAY;
          address <= (others=>'0') after PDELAY;
540       dIn_h <= (others=>'0') after PDELAY;
          dOut_l <= (others=>'0') after PDELAY;
542       phitIn <= (others=>'0') after PDELAY;
          pkt_out <= (others=>'0') after PDELAY;
544       config_reg <= (others=>'0');
546     elsif rising_edge(na_clk) then
          if ctrlOutreg_ld='1' then
548         dma_ctrl_reg <= dma_ctrl after PDELAY;
          end if;
550       if adreg_ld='1' then
            address <= phitIn(SPM_ADDR_WIDTH-1+16 downto 16) after
                PDELAY;
552       end if;
          if dInreg_ld='1' then
554         dIn_h <= phitIn(DATA_WIDTH-1 downto 0) after PDELAY;
          end if;
556       if dOutreg_ld='1' then
            dOut_l <= spm_in.SData(DATA_WIDTH-1 downto 0) after PDELAY;
558       end if;
          phitIn <= pkt_in after PDELAY;
560       pkt_out <= phitOut after PDELAY;
          config_reg <= proc_in.MCmd(1) & config;
562
        end if;
564   end process;
566
568
570 end rtl;
```

Appendix C

JOP infrastructure

This appendix contains the following files:

jopcpu.vhd this file comes from the JOP project, the file describes the core of the JOP processor. The modification we have made to this file was to pull out the SimpCon interface from the SPM, to connect it to the T-CREST NoC platform. The file starts on page 93.

jopmul_512x32.vhd this file comes from the JOP project, the file describes the top level component of a multicore JOP platform. The modifications we have made to this file was to instantiate the T-CREST NoC platform and connect it to the JOP cores. The file starts on page 101.

Listing C.1: jopcpu.vhd

```
1 --
2 --
3 --  This file is a part of JOP, the Java Optimized Processor
4 --
5 --  Copyright (C) 2001-2008, Martin Schoeberl (martin@jopdesign.com
      )
6 --
7 --  This program is free software: you can redistribute it and/or
      modify
8 --  it under the terms of the GNU General Public License as
      published by
```

```
9  --  the Free Software Foundation, either version 3 of the License, or
   --  (at your option) any later version.
11 --
   --  This program is distributed in the hope that it will be useful,
13 --  but WITHOUT ANY WARRANTY; without even the implied warranty of
   --  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
15 --  GNU General Public License for more details.
   --
17 --  You should have received a copy of the GNU General Public License
   --  along with this program.  If not, see <http://www.gnu.org/licenses/>.
19 --


21
   --
23 --  jopcpu.vhd
   --
25 --  The JOP CPU
   --
27 --  2007-03-16  creation
   --  2007-04-13  Changed memory connection to records
29 --  2008-02-20  memory - I/O muxing after the memory controller (mem_sc)
   --  2008-03-03  added scratchpad RAM
31 --  2008-03-04  correct MUX selection
   --  2009-11-15  include extension code
33 --
   --  todo: clean up: substitute all signals by records
35
   -- comments from former extension.vhd
37 --
   --  2004-09-11  first version
39 --  2005-04-05  Reserve negative addresses for wishbone interface
   --  2005-04-07  generate bsy from delayed wr or'ed with mem_out.bsy
41 --  2005-05-30  added wishbone interface
   --  2005-11-28  Substitute WB interface by the SimpCon IO interface ;-)
43 --              All IO devices are now memory mapped
   --  2007-04-13  Changed memory connection to records
45 --              New array instructions
   --  2007-12-22  Correction of data MUX bug for array read access
47 --  2008-02-20  Removed memory - I/O muxing
   --  2009-11-22  move MMU decode from jopcpu/extension to decode
49 --

51 library ieee;
   use ieee.std_logic_1164.all;
53 use ieee.numeric_std.all;

55 use work.jop_types.all;
   use work.sc_pack.all;
57
```

```
59 entity jopcpu is

61 generic (
     jpc_width : integer;      -- address bits of java bytecode pc =
         cache size
63   block_bits  : integer;      -- 2*block_bits is number of cache
         blocks
     spm_width : integer := 0    -- size of scratchpad RAM (in number
         of address bits)
65 );

67 port (
     clk    : in std_logic;
69   reset : in std_logic;

71 --
   --  SimpCon memory interface
73 --
     sc_mem_out    : out sc_out_type;
75   sc_mem_in    : in sc_in_type;

77 --
   --  SimpCon IO interface
79 --
     sc_io_out    : out sc_out_type;
81   sc_io_in     : in sc_in_type;


83
   --
85 --  SimpCon DMA interface
   --
87   sc_noc_out    : out sc_out_type;
     sc_noc_in    : in sc_in_type;
89
   --
91 --  Interrupts from sc_sys
   --
93   irq_in       : in irq_bcf_type;
     irq_out      : out irq_ack_type;
95   exc_req      : out exception_type;

97 --
   --  TM exception
99 --

101   exc_tm_rollback : in std_logic := '0'
   );
103 end jopcpu;

105 architecture rtl of jopcpu is

107 --
   --  Signals
109 --
```

```
111   signal stack_tos     : std_logic_vector(31 downto 0);
      signal stack_nos     : std_logic_vector(31 downto 0);
113   signal rd, wr        : std_logic;
      signal mmu_instr     : std_logic_vector(MMU_WIDTH-1 downto 0);
115   signal stack_din     : std_logic_vector(31 downto 0);

117 -- extension/mem interface

119   signal mem_in        : mem_in_type;
      signal mem_out       : mem_out_type;
121
      signal sc_ctrl_mem_out  : sc_out_type;
123   signal sc_ctrl_mem_in : sc_in_type;

125   signal sc_scratch_out : sc_out_type;
      signal sc_scratch_in  : sc_in_type;
127
      signal next_mux_mem   : std_logic_vector(1 downto 0);
129   signal dly_mux_mem    : std_logic_vector(1 downto 0);
      signal mux_mem        : std_logic_vector(1 downto 0);
131   signal is_pipelined   : std_logic;

133   signal mem_access     : std_logic;
      signal noc_access     : std_logic;
135   signal io_access      : std_logic;

137   signal bsy            : std_logic;

139   signal bc_wr_addr     : std_logic_vector(jpc_width-3 downto 0); --
          address for jbc (in words!)
      signal bc_wr_data     : std_logic_vector(31 downto 0);  -- write
          data for jbc
141   signal bc_wr_ena      : std_logic;

143 -- SimpCon io interface

145   signal sp_ov          : std_logic;

147 -- *************** signals from extension ***************

149 --
    --  signals for mulitiplier
151 --
      signal mul_dout         : std_logic_vector(31 downto 0);
153   signal mul_wr         : std_logic;


155
      signal wr_dly         : std_logic;  -- generate a bsy with delayed
          wr
157
      signal exr            : std_logic_vector(31 downto 0);  --
          extension data register
159

161 begin
```

```
163   exc_req.rollback <= exc_tm_rollback;

165 --
    --  components of jop
167 --

169   core: entity work.core
        generic map(jpc_width)
171     port map (
          clk => clk,
173       reset => reset,
          bsy => bsy,
175       din => stack_din,
          mem_in => mem_in,
177       mmu_instr => mmu_instr,
          mul_wr => mul_wr,
179       wr_dly => wr_dly,
          bc_wr_addr => bc_wr_addr,
181       bc_wr_data => bc_wr_data,
          bc_wr_ena => bc_wr_ena,
183       irq_in => irq_in,
          irq_out => irq_out,
185       sp_ov => sp_ov,
          aout => stack_tos,
187       bout => stack_nos
        );
189
      exc_req.spov <= sp_ov;
191
      mem: entity work.mem_sc
193     generic map (
          jpc_width => jpc_width,
195       block_bits => block_bits
        )
197     port map (
          clk => clk,
199       reset => reset,
          ain => stack_tos,
201       bin => stack_nos,

203       np_exc => exc_req.np,
          ab_exc => exc_req.ab,
205
          mem_in => mem_in,
207       mem_out => mem_out,

209       bc_wr_addr => bc_wr_addr,
          bc_wr_data => bc_wr_data,
211       bc_wr_ena => bc_wr_ena,

213       sc_mem_out => sc_ctrl_mem_out,
          sc_mem_in => sc_ctrl_mem_in
215     );
```

```
217
  --
219 -- Generate scratchpad memory when size is != 0.
  -- Results in warnings when the size is 0.
221 --
  sc1: if spm_width /= 0 generate
223   scm: entity work.sdpram
        generic map (
225       width => 32,
          addr_width => spm_width
227     )
        port map (
229       wrclk => clk,
          data => sc_scratch_out.wr_data,
231       wraddress => sc_scratch_out.address(spm_width-1 downto 0),
          wren => sc_scratch_out.wr,
233       rdclk => clk,
          rdaddress => sc_scratch_out.address(spm_width-1 downto 0),
235       rden => sc_scratch_out.rd,
          dout => sc_scratch_in.rd_data
237   );
  end generate;
239
  sc_scratch_in.rdy_cnt <= (others => '0');
241
  --
243 --  Select for the read mux
  --
245 --  TODO: this mux selection works ONLY for two cycle pipelining!
  --  25.3.2011: should now be ok - at least the bug with
247 --  SPM, NoC IO, and TDMA arbiter disappeared
  --  TODO: should check more configurations
249 --

251 process(clk, reset)
  begin
253   if (reset='1') then
      dly_mux_mem <= (others => '0');
255   next_mux_mem <= (others => '0');
      is_pipelined <= '0';
257   elsif rising_edge(clk) then

259     if sc_ctrl_mem_out.rd='1' or sc_ctrl_mem_out.wr='1' then
--      if sc_ctrl_mem_out.rd='1' then
261       -- highest address bits decides between IO, memory, and on-
            chip memory
          -- save the mux selection on read or write
263       next_mux_mem <= sc_ctrl_mem_out.address(SC_ADDR_SIZE-1 downto
              SC_ADDR_SIZE-2);
          -- a read or write with rdy_cnt of 1 means pipelining
265       if sc_ctrl_mem_in.rdy_cnt = "01" then
            is_pipelined <= '1';
267       end if;
          -- remeber for the next mux selection in case of pipelining
269       dly_mux_mem <= next_mux_mem;
```

```
    end if;
--    -- delayed mux selection for pipelined access
--    if sc_ctrl_mem_in.rdy_cnt(1) = '0' then
--      dly_mux_mem <= next_mux_mem;
--    end if;
    -- pipelining is over
    if sc_ctrl_mem_in.rdy_cnt = "00" then
      is_pipelined <= '0';
    end if;

  end if;
end process;

process(next_mux_mem, dly_mux_mem, sc_ctrl_mem_out, sc_ctrl_mem_in,
     sc_mem_in, sc_io_in, sc_noc_in, is_pipelined, mux_mem)
begin

  mem_access <= '0';
  noc_access <= '0';
  io_access <= '0';

  -- for one cycle peripherals we need to set the mux from
       next_mux_mem
  mux_mem <= next_mux_mem;
  -- for pipelining we need to delay the mux selection
  if is_pipelined='1' then
    mux_mem <= dly_mux_mem;
  end if;

  -- read MUX
  case mux_mem is
    when "10" =>
      --sc_ctrl_mem_in <= sc_scratch_in;
      sc_ctrl_mem_in <= sc_noc_in;
    when "11" =>
      sc_ctrl_mem_in <= sc_io_in;
    when others =>
      sc_ctrl_mem_in <= sc_mem_in;
  end case;

  -- select
  case sc_ctrl_mem_out.address(SC_ADDR_SIZE-1 downto SC_ADDR_SIZE
       -2) is
    when "10" =>
      noc_access <= '1';
    when "11" =>
      io_access <= '1';
    when others =>
      mem_access <= '1';
  end case;

end process;

  sc_mem_out.address <= sc_ctrl_mem_out.address;
  sc_mem_out.wr_data <= sc_ctrl_mem_out.wr_data;
```

```
    sc_mem_out.wr <= sc_ctrl_mem_out.wr and mem_access;
323 sc_mem_out.rd <= sc_ctrl_mem_out.rd and mem_access;
    sc_mem_out.atomic <= sc_ctrl_mem_out.atomic;
325 sc_mem_out.cache <= sc_ctrl_mem_out.cache;
    sc_mem_out.cinval <= sc_ctrl_mem_out.cinval;
327 sc_mem_out.tm_cache <= sc_ctrl_mem_out.tm_cache;

329 --sc_scratch_out.address <= sc_ctrl_mem_out.address;
    --sc_scratch_out.wr_data <= sc_ctrl_mem_out.wr_data;
331 --sc_scratch_out.wr <= sc_ctrl_mem_out.wr and scratch_access;
    --sc_scratch_out.rd <= sc_ctrl_mem_out.rd and scratch_access;
333 --sc_scratch_out.atomic <= sc_ctrl_mem_out.atomic;
    --sc_scratch_out.cinval <= sc_ctrl_mem_out.cinval;
335 --sc_scratch_out.cache <= sc_ctrl_mem_out.cache;

337 sc_noc_out.address <= sc_ctrl_mem_out.address;
    sc_noc_out.wr_data <= sc_ctrl_mem_out.wr_data;
339 sc_noc_out.wr <= sc_ctrl_mem_out.wr and noc_access;
    sc_noc_out.rd <= sc_ctrl_mem_out.rd and noc_access;
341 sc_noc_out.atomic <= sc_ctrl_mem_out.atomic;
    sc_noc_out.cinval <= sc_ctrl_mem_out.cinval;
343 sc_noc_out.cache <= sc_ctrl_mem_out.cache;

345 sc_io_out.address <= sc_ctrl_mem_out.address;
    sc_io_out.wr_data <= sc_ctrl_mem_out.wr_data;
347 sc_io_out.wr <= sc_ctrl_mem_out.wr and io_access;
    sc_io_out.rd <= sc_ctrl_mem_out.rd and io_access;
349 sc_io_out.atomic <= sc_ctrl_mem_out.atomic;
    sc_io_out.cinval <= sc_ctrl_mem_out.cinval;
351 sc_io_out.cache <= sc_ctrl_mem_out.cache;

353 -- *************** code from extension ****************

355 ml : entity work.mul
        port map (
357       clk => clk,
          ain => stack_tos,
359       bin => stack_nos,
          wr => mul_wr,
361       dout => mul_dout
      );
363
    stack_din <= exr;
365
    --
367 --  TODO: the following code is degenerated to decode functions
    --  should probably go to decode.vhd
369 --


371 --
    --  read
373 --
    --  TODO: the read MUX could be set by using the
375 --  according wr/mmu_instr from JOP and not the
    --  following rd/mmu_instr
```

```
377 --  Than no intermixing of mul/mem and io operations
    --  is allowed. But we are not using interleaved mul/mem/io
379 --  operations in jvm.asm anyway.
    --
381 --  TAKE CARE when mem_out.bcstart is read!
    --
383 --   ** bcstart is also read without a mem_bc_rd JOP wr !!! ***
    --    => a combinatorial mux select on rd and ext_adr==7!
385 --
    --    The rest could be set with JOP wr start transaction
387 --    Is this also true for io_data?
    --
389 --  29.11.2005 evening: I think this solution driving the exr
    --  mux from mmu_instr is quite ok. The pipelining from rd/ext_adr
391 --  to A is fixed.
    --
393 process(clk, reset)
    begin
395   if (reset='1') then
        exr <= (others => '0');
397   elsif rising_edge(clk) then

399     if (mmu_instr=LDMRD) then
          exr <= mem_out.dout;
401     elsif (mmu_instr=LDMUL) then
          exr <= mul_dout;
403     -- elsif (mmu_instr=LDBCSTART) then
      else
405       exr <= mem_out.bcstart;
      end if;
407
    end if;
409 end process;

411
    -- a JOP wr generates the first bsy cycle
413 -- the following are generated by the memory
    -- system or the SimpCon device
415 bsy <= wr_dly or mem_out.bsy;

417 end rtl;
```

Listing C.2: jopmul_512x32.vhd

```
1 --
  --
3 --  This file is a part of JOP, the Java Optimized Processor
  --
5 --  Copyright (C) 2001-2008, Martin Schoeberl (martin@jopdesign.com
      )
  --
7 --  This program is free software: you can redistribute it and/or
      modify
  --  it under the terms of the GNU General Public License as
      published by
```

```
--  the Free Software Foundation, either version 3 of the License, or
--  (at your option) any later version.
--
--  This program is distributed in the hope that it will be useful,
--  but WITHOUT ANY WARRANTY; without even the implied warranty of
--  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
--  GNU General Public License for more details.
--
--  You should have received a copy of the GNU General Public License
--  along with this program.  If not, see <http://www.gnu.org/licenses/>.
--


--
--  jopmul_512x32.vhd
--
--  top level for a 512x32 SSRAM board (e.g. Altera DE2-70 board)
--
--  2006-08-06  adapted from jopcyc.vhd
--  2007-06-04  Use jopcpu and change component interface to records
--  2010-06-25  Working version with SSRAM
--


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.jop_types.all;
use work.sc_pack.all;
use work.sc_arbiter_pack.all;
use work.jop_config.all;
use work.defs.all;


entity jop is

generic (
  ram_cnt    : integer := 3;   -- clock cycles for external ram
--  rom_cnt    : integer := 3;   -- clock cycles for external rom OK for 20 MHz
  rom_cnt    : integer := 15;  -- clock cycles for external rom for 100 MHz
  jpc_width  : integer := 12;  -- address bits of java bytecode pc = cache size
  block_bits  : integer := 5;   -- 2*block_bits is number of cache blocks
  spm_width : integer := 7;   -- size of scratchpad RAM (in number of address bits for 32-bit words)
  cpu_cnt    : integer := 9    -- number of cpus
);
```

```
55
port (
57   clk        : in std_logic;
--
59 --   serial interface
--
61   ser_txd      : out std_logic;
   ser_rxd      : in std_logic;
63   oUART_CTS    : in std_logic;
   iUART_RTS    : out std_logic;
65
--
67 --   watchdog
--
69   wd         : out std_logic;

71 --
--   LEDs
73 --
   oLEDR    : out std_logic_vector(17 downto 0);
75 --   oLEDG    : out std_logic_vector(7 downto 0);

77



79 --
--   Switches
81 --
   iSW        : in std_logic_vector(17 downto 0);
83
--
85 --   only one ram bank
--
87   oSRAM_A     : out std_logic_vector(18 downto 0);   -- edit
   SRAM_DQ     : inout std_logic_vector(31 downto 0);   -- edit
89   oSRAM_CE1_N   : out std_logic;
   oSRAM_OE_N    : out std_logic;
91   oSRAM_BE_N    : out std_logic_vector(3 downto 0);
   oSRAM_WE_N    : out std_logic;
93   oSRAM_GW_N    : out std_logic;
   oSRAM_CLK   : out std_logic;
95   oSRAM_ADSC_N : out std_logic;
   oSRAM_ADSP_N : out std_logic;
97   oSRAM_ADV_N  : out std_logic;
   oSRAM_CE2   : out std_logic;
99   oSRAM_CE3_N  : out std_logic
);
101 end jop;

103 architecture rtl of jop is

105 --
--   components:
107 --

109 component pll is
```

```
  generic (multiply_by : natural; divide_by : natural);
111 port (
    inclk0     : in std_logic;
113   c0       : out std_logic;
    c1       : out std_logic;
115   locked     : out std_logic
  );
117 end component;

119

  --
121 --  Signals
  --
123   signal clk_int       : std_logic;
    signal clk_int_inv     : std_logic;
125   signal pll_lock      : std_logic;

127   signal int_res       : std_logic;
    signal res_cnt       : unsigned(2 downto 0) := "000";  -- for the
        simulation
129
    attribute altera_attribute : string;
131   attribute altera_attribute of res_cnt : signal is "POWER_UP_LEVEL
        =LOW";

133 --
  --  jopcpu connections
135 --
    signal sc_arb_out    : arb_out_type(0 to cpu_cnt-1);
137   signal sc_arb_in     : arb_in_type(0 to cpu_cnt-1);

139   signal sc_mem_out    : sc_out_type;
    signal sc_mem_in     : sc_in_type;
141   signal sc_local_mem_out    : sc_out_array_type(0 to cpu_cnt-1);
    signal sc_local_mem_in     : sc_in_array_type(0 to cpu_cnt-1);
143
    signal sc_io_out     : sc_out_array_type(0 to cpu_cnt-1);
145   signal sc_io_in      : sc_in_array_type(0 to cpu_cnt-1);
    signal sc_noc_out    : sc_out_array_type(0 to cpu_cnt-1);
147   signal sc_noc_in     : sc_in_array_type(0 to cpu_cnt-1);

149   signal irq_in      : irq_in_array_type(0 to cpu_cnt-1);
    signal irq_out       : irq_out_array_type(0 to cpu_cnt-1);
151   signal exc_req       : exception_array_type(0 to cpu_cnt-1);

153 --
  --  IO interface
155 --
    signal ser_in      : ser_in_type;
157   signal ser_out       : ser_out_type;
    type wd_out_array is array (0 to cpu_cnt-1) of std_logic;
159   signal wd_out      : wd_out_array;

161   -- for generation of internal reset
  -- memory interface
```

```
163
   signal ram_addr      : std_logic_vector(18 downto 0);
165   signal ram_dout      : std_logic_vector(31 downto 0);
   signal ram_din       : std_logic_vector(31 downto 0);
167   signal ram_dout_en   : std_logic;
   signal ram_clk       : std_logic;
169   signal ram_nsc       : std_logic;
   signal ram_ncs       : std_logic;
171   signal ram_noe       : std_logic;
   signal ram_nwe       : std_logic;
173
-- cmpsync
175
   signal sync_in_array  : sync_in_array_type(0 to cpu_cnt-1);
177   signal sync_out_array : sync_out_array_type(0 to cpu_cnt-1);

179 -- not available at this board:
   signal ser_ncts      : std_logic;
181   signal ser_nrts      : std_logic;

183 -- remove the comment for RAM access counting
-- signal ram_count    : std_logic;
185
-- NoC signals
187   signal procM : procMasters;
   signal procS : procSlaves;
189
begin
191
  ser_ncts <= '0';
193 --
--  intern reset
195 --  no extern reset, epm7064 has too less pins
--
197
-- should also use PLL lock signal
199 process(clk_int)
begin
201   if rising_edge(clk_int) then
    if (res_cnt/="111") then
203       res_cnt <= res_cnt+1;
    end if;
205
    int_res <= not res_cnt(0) or not res_cnt(1) or not res_cnt(2);
207   end if;
end process;
209
--
211 --  components of jop
--
213   pll_inst : pll generic map(
    multiply_by => pll_mult,
215     divide_by => pll_div
   )
217   port map (
```

```
    inclk0   => clk,
    c0   => clk_int,
    c1   => clk_int_inv,
    locked => pll_lock
  );
-- clk_int <= clk;

-- process(wd_out)
-- variable wd_help : std_logic;
--  begin
--    wd_help := '0';
--    for i in 0 to cpu_cnt-1 loop
--      wd_help := wd_help or wd_out(i);
--    end loop;
--    wd <= wd_help;
-- end process;

  wd <= wd_out(0);

  gen_cpu: for i in 0 to cpu_cnt-1 generate
    cpu: entity work.jopcpu
      generic map(
        jpc_width => jpc_width,
        block_bits => block_bits,
        spm_width => spm_width
      )
      port map(clk_int, int_res,
        sc_arb_out(i), sc_arb_in(i),
        sc_io_out(i), sc_io_in(i),
        sc_noc_out(i), sc_noc_in(i),
        irq_in(i), irq_out(i), exc_req(i));
  end generate;

  sc_noc : entity work.sc2ocp_noc
    port map(
      clk => clk_int,
      reset => int_res,
      sc_noc_out => sc_noc_out,
      sc_noc_in => sc_noc_in
    );


  arbiter: entity work.arbiter
    generic map(
      addr_bits => SC_ADDR_SIZE,
      cpu_cnt => cpu_cnt,
      write_gap => 2,
      read_gap => 2,
      slot_length => 3
    )
    port map(clk_int, int_res,
      sc_arb_out, sc_arb_in,
      sc_mem_out, sc_mem_in);

  -- io for processor 0
```

```
273   io: entity work.scio generic map (
        cpu_id => 0,
275     cpu_cnt => cpu_cnt
      )
277   port map (clk_int, int_res,
        sc_io_out(0), sc_io_in(0),
279     irq_in(0), irq_out(0), exc_req(0),

281     sync_out => sync_out_array(0),
        sync_in => sync_in_array(0),
283
        txd => ser_txd,
285     rxd => ser_rxd,
        ncts => oUART_CTS,
287     nrts => iUART_RTS,

289     oLEDR => oLEDR,
--        oLEDG => oLEDG,
291     iSW => iSW,

293     wd => wd_out(0),
        l => open,
295     r => open,
        t => open,
297     b => open
        -- remove the comment for RAM access counting
299     -- ram_cnt => ram_count
      );
301
  -- io for processors with only sc_sys
303 gen_io: for i in 1 to cpu_cnt-1 generate
    io2: entity work.sc_sys generic map (
305     addr_bits => 4,
        clk_freq => clk_freq,
307     cpu_id => i,
        cpu_cnt => cpu_cnt
309   )
      port map(
311     clk => clk_int,
        reset => int_res,
313     address => sc_io_out(i).address(3 downto 0),
        wr_data => sc_io_out(i).wr_data,
315     rd => sc_io_out(i).rd,
        wr => sc_io_out(i).wr,
317     rd_data => sc_io_in(i).rd_data,
        rdy_cnt => sc_io_in(i).rdy_cnt,
319
        irq_in => irq_in(i),
321     irq_out => irq_out(i),
        exc_req => exc_req(i),
323
        sync_out => sync_out_array(i),
325     sync_in => sync_in_array(i),
        wd => wd_out(i)
327     -- remove the comment for RAM access counting
```

```
      -- ram_count => ram_count
    );
  end generate;

  scm: entity work.sc_mem_if
    generate map (
      ram_ws => ram_cnt-1,
      addr_bits => 19
    )
    port map (clk_int, int_res,
      clk_int_inv,
      sc_mem_out, sc_mem_in,

      ram_addr => ram_addr,
      ram_dout => ram_dout,
      ram_din => ram_din,
      ram_dout_en => ram_dout_en,
      ram_clk => ram_clk,
      ram_nsc => ram_nsc,
      ram_ncs => ram_ncs,
      ram_noe => ram_noe,
      ram_nwe => ram_nwe
    );


  -- syncronization of processors
  sync: entity work.cmpsync generic map (
    cpu_cnt => cpu_cnt)
    port map
    (
      clk => clk_int,
      reset => int_res,
      sync_in_array => sync_in_array,
      sync_out_array => sync_out_array
    );


  process(ram_dout_en, ram_dout)
  begin
    if ram_dout_en='1' then
      SRAM_DQ <= ram_dout;
    else
      SRAM_DQ <= (others => 'Z');
    end if;
  end process;

  ram_din <= SRAM_DQ;

  -- remove the comment for RAM access counting
  -- ram_count <= ram_ncs;

--
--  To put this RAM address in an output register
--  we have to make an assignment (FAST_OUTPUT_REGISTER)
--
```

```
383   oSRAM_A <= ram_addr;
      oSRAM_CE1_N <= ram_ncs;
385   oSRAM_OE_N <= ram_noe;
      oSRAM_WE_N <= ram_nwe;
387   oSRAM_BE_N <= (others => '0');
      oSRAM_GW_N <= '1';
389   oSRAM_CLK <= ram_clk;

391   oSRAM_ADSC_N <= ram_nsc;
      oSRAM_ADSP_N <= '1';
393   oSRAM_ADV_N <= '1';

395   oSRAM_CE2 <= not ram_ncs;
        oSRAM_CE3_N <= ram_ncs;
397
    end rtl;
```

Appendix D

TDM scheduler source code

This appendix contains the following files:

IOutput.h is the abstract class that describes the interface of an Output class, the file starts on page 112.

xmlOutput.h is the header file of the Output class that writes the xml output, the file starts on page 112.

xmlOutput.cpp is the Output class that writes the xml output, the file starts on page 113.

vhdlOutput.h is the header file of the Output class that writes the vhdl output, the file starts on page 117.

vhdlOutput.cpp is the Output class that writes the vhdl output, the file starts on page 118.

ScheduleConverter.java is the main class of the schedule converter that converts the xml file to a Java array, the file starts on page 125.

SchedulePrinter.java is the SchedulePrinter class that formats and prints the Java file, the file starts on page 128.

TileCoord.java is a class for saving the coordinate of a tile, the file starts on page 131.

Listing D.1: IOutput.h

```
/*
 * File:   IOutput.h
 * Author: Rasmus Bo Soerensen
 *
 * Created on 6. august 2012, 11:07
 */

#ifndef IOUTPUT_H
#define IOUTPUT_H

#include "schedule.hpp"

class IOutput
{
public:
    virtual bool output_schedule(const network_t& n) =0;
};

#endif  /* IOUTPUT_H */
```

Listing D.2: xmlOutput.h

```
/*
 * File:   xmlOutput.h
 * Author: T410s
 *
 * Created on 6. august 2012, 11:13
 */

#ifndef XMLOUTPUT_H
#define XMLOUTPUT_H

#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <stdlib.h>
#include <unordered_map>
#include <cstdio>
#include "IOutput.h"
#include "lex_cast.h"
#include "pugixml.hpp"


class xmlOutput: public IOutput {
private:
    string output_dir;

    char p2c(port_id p);
  void print_coord(const pair<int, int> r, char* co, const size_t
      buffer_size);

public:
```

```
    xmlOutput(string _output_dir);
32  ~xmlOutput();

34  bool output_schedule(const network_t& n);

36 };

38 #endif  /* XMLOUTPUT_H */
```

Listing D.3: xmlOutput.cpp

```
1  /*
   * File:   xmlOutput.cpp
3  * Author: Rasmus
   *
5  * Created on 6. august 2012, 11:13
   */
7
   #include <string.h>
9
   #include "xmlOutput.h"
11

13 bool xmlOutput::output_schedule(const network_t& n)
   {
15   int numOfNodes = n.routers().size();
     int countWidth = ceil(log2(n.best));
17
     xml_document doc;
19   xml_node schedule = doc.append_child("schedule");
     schedule.append_attribute("length").set_value(n.best);
21
     for(vector<router_t*>::const_iterator r = n.routers().begin(); r
         != n.routers().end(); r++){ // For each router, write Network
          Adapter Table and Router Table
23     // New xml tile
       xml_node tile = schedule.append_child("tile");
25     char co[10];
       print_coord((*r)->address,co,sizeof(co));
27     tile.append_attribute("id") = co;
       // Vector for saving data to calculate Worst-Case Latencies
29     vector<router_id> destinations(n.best,(*r)->address);
       for(timeslot t = 0; t < n.best; t++){ // Write table row for
           each timeslot
31       // New timeslot
         xml_node ts = tile.append_child("timeslot");
33       ts.append_attribute("value") = t;
         int t0 = t-1;
35       int t1 = t;
         if(t == 0){
37         t0 = n.best-1;
           t1 = n.best;
39       }
         // Write row in Network Adapter table
41       router_id dest_id = (*r)->address;
```

```
        router_id src_id = (*r)->address;
43      if ((*r)->local_in_best_schedule.has((t+2)%n.best)){
          dest_id = (*r)->local_in_best_schedule.get((t+2)%n.best)->
              to;
45      }
        if ((*r)->local_out_best_schedule.has(t1))
47        src_id = (*r)->local_out_best_schedule.get(t1)->from;
        destinations[t] = dest_id;
49      // New na
        xml_node na = ts.append_child("na");
51      print_coord(src_id,co,sizeof(co));
        na.append_attribute("rx") = co;
53      print_coord(dest_id,co,sizeof(co));
        na.append_attribute("tx") = co;
55
        // Write row in Router table
57      port_id ports[5] = {__NUM_PORTS, __NUM_PORTS, __NUM_PORTS,
            __NUM_PORTS, __NUM_PORTS};
        // New router
59      xml_node router = ts.append_child("router");


61
        for(int out_p = 0; out_p < __NUM_PORTS-1; out_p++){
63        // For all 4 output ports not being the local port.
          if(!(*r)->out((port_id)out_p).has_link()){
65          continue; // No outgoing link from the port.
          }
67        if(!(*r)->out((port_id)out_p).link().best_schedule.has(t)){
            ports[(port_id)out_p] = __NUM_PORTS; // No outgoing
                channel on link
69          continue;
          }
71        // If there is a channel comming out of the port, find the
          // input port from which the channel is comming from.
73        const channel* out_c =(*r)->out((port_id)out_p).link().
              best_schedule.get(t);
          for(int in_p = 0; in_p < __NUM_PORTS-1; in_p++){
75          // For all 4 input ports not being the local port.
            if(!(*r)->in((port_id)in_p).has_link())
77            continue; // No link into this port
            if((*r)->in((port_id)in_p).link().best_schedule.has(t0)){
79            const channel* in_c =(*r)->in((port_id)in_p).link().
                  best_schedule.get(t0);
              if(out_c == in_c){
81              // The correct link found
                ports[(port_id)out_p] = (port_id)in_p;
83              break;
              }
85          }
          }
87        if(ports[(port_id)out_p] != __NUM_PORTS){
            continue; // Channel was found on one of the input ports.
89        }
          // It should be on the local in port, but we test it anyway
              .
```

```
91          if((*r)->local_in_best_schedule.has(t)){
              const channel* in_c = (*r)->local_in_best_schedule.get(t)
                  ;
93            if(out_c == in_c){
                // The correct link found
95              ports[(port_id)out_p] = L;
              } else {
97              cout << "Failure: Channel rose from nothing like a
                    fenix." << endl;
              }
99          }
        }
101
        if((*r)->local_out_best_schedule.has(t1)){ // For the local
            out port.
103       const channel* out_c = (*r)->local_out_best_schedule.get(t1
              );
          for(int in_p = 0; in_p < __NUM_PORTS-1; in_p++){
105         // For all 4 input ports not being the local port.
            if(!(*r)->in((port_id)in_p).has_link())
107           continue; // No link into this port
            if((*r)->in((port_id)in_p).link().best_schedule.has(t0)){
109           const channel* in_c =(*r)->in((port_id)in_p).link().
                  best_schedule.get(t0);
              if(out_c == in_c){
111             // The correct link found
                ports[L] = (port_id)in_p;
113             break;
              }
115         }
          }
117       if(ports[L] == __NUM_PORTS){
            // If channel was not found on any of the 4 input ports.
119         // It should be on the local in port, but we test it
                anyway.
            cout << "Failure: Not allowed to route back in to local."
                 << endl;
121       }
          // and so on...
123     }
        for(int p = 0; p < __NUM_PORTS; p++){
125       // New output
          xml_node output = router.append_child("output");
127       sprintf(co,"%c",p2c((port_id)p)); // Should be snprintf,
              avoiding buffer overflow
//          sprintf(co,sizeof(co),"%c",p2c((port_id)p));
129       output.append_attribute("id") = co;
          sprintf(co,"%c",p2c(ports[(port_id)p])); // Should be
              snprintf, avoiding buffer overflow
131 //        sprintf(co,sizeof(co),"%c",p2c(ports[(port_id)p]));
          output.append_attribute("input") = co;
133     }

135   }
      xml_node latency = tile.append_child("latency");
```

```
137     // The following for loop is slow and unnecessary, can be
            changed to improve runtime
        for_each(n.channels(), [&](const channel & c) {
139       if(c.from != (*r)->address){
            return; // Channel not from router
141       }
          // For each channel from router
143       int WCL = 0;
          int late = 0;
145       int inlate = 0;
          bool init = true;
147       for(int i = 0; i < n.best; i++){
            if(c.to != destinations[i]){
149           // Increment latency
              late++;
151           continue;
            }
153         // Correct destination
            if(init){
155           init = false;
              inlate = late;
157         }
            if(late > WCL){
159           WCL = late;
            }
161         late = 0;
          }
163       late += inlate;
          if(late > WCL){
165         WCL = late;
          }
167       // Analyze the latency
          xml_node destination = latency.append_child("destination");
169       print_coord(c.to,co,sizeof(co));
          destination.append_attribute("id") = co;
171       destination.append_attribute("WCL") = WCL;
        });
173   }
      char co[500];
175   sprintf(co,"%soutput.xml",output_dir.c_str()); // Should be
          snprintf, avoiding buffer overflow
      //sprintf(co,sizeof(co),"%soutput.xml",output_dir.c_str());
177   doc.save_file(co);

179   delete this;
      return true;
181 }

183 void xmlOutput::print_coord(const pair<int, int> r,char* co, const
        size_t buffe_size){
      sprintf(co,"(%i,%i)",r.first,r.second); // Should be snprintf,
          avoiding buffer overflow
185 //  sprintf(co,buffer_size,"(%i,%i)",c.to.first,c.to.second);
    }
187
```

```
char xmlOutput::p2c(port_id p){
    char c;
    if (p == N) c = 'N';
    if (p == E) c = 'E';
    if (p == S) c = 'S';
    if (p == W) c = 'W';
    if (p == L) c = 'L';
    if (p == __NUM_PORTS) c = 'D';

    return c;
  }

xmlOutput::xmlOutput(string _output_dir) : output_dir(_output_dir){
}

xmlOutput::~xmlOutput(){

}
```

Listing D.4: vhdlOutput.h

```
/*
 * File:   vhdlOutput.h
 * Author: T410s
 *
 * Created on 6. august 2012, 11:13
 */

#ifndef VHDLOUTPUT_H
#define VHDLOUTPUT_H

#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <unordered_map>
#include "IOutput.h"
#include "lex_cast.h"


class vhdlOutput: public IOutput {
private:
    enum port {North,East,South,West,Local,DC};

//    class STslot{
//    public:
//            port ports[5];
//            int x_dest;
//            int y_dest;
//            int x_src;
//            int y_src;
//            STslot(){
//                    for(int i = 0; i < 5; i++){
//                            ports[i] = DC;
//                    }
```

```
35 //                      x_dest = 0;
   //                      y_dest = 0;
37 //                      x_src = 0;
   //                      y_src = 0;
39 //              }
   //
41 //     };

43     ofstream niST;
       ofstream routerST;
45     string numOfNodesStr;

47     string bin(int val, int bits);
       char p2c(port_id p);
49     void startST(int num, ofstream* ST);
       void writeHeaderRouter(int countWidth);
51     void endArchRouter();
       void writeSlotRouter(int slotNum, int countWidth, port_id*
           ports);
53     void writeHeaderNI(int countWidth, int numOfNodes);

55     void writeSlotNIDest(int slotNum, int countWidth, int dest);
       void writeSlotNISrc(int src);
57
       void startniST(int num);
59     void startrouterST(int num);
       void endniST(int num);
61     void endrouterST(int num);

63     void endArchNI();
   public:
65     vhdlOutput(string output_dir);
       ~vhdlOutput();
67
   //     bool output_schedule(network_t& n);
69     bool output_schedule(const network_t& n);

71 };

73 #endif  /* VHDLOUTPUT_H */
```

Listing D.5: vhdlOutput.cpp

```
   /*
2   * File:   vhdlOutput.cpp
    * Author: T410s
4   *
    * Created on 6. august 2012, 11:13
6   */

8  #include "vhdlOutput.h"


10
   bool vhdlOutput::output_schedule(const network_t& n)
12 {
```

```
  int numOfNodes = n.routers().size();
14 numOfNodesStr = ::lex_cast<string>(numOfNodes);
  int countWidth = ceil(log2(n.best));
16
  this->writeHeaderRouter(countWidth);
18 this->writeHeaderNI(countWidth,numOfNodes);

20 for(vector<router_t*>::const_iterator r = n.routers().begin(); r
       != n.routers().end(); r++){ // For each router, write Network
       Adapter Table and Router Table
    int r_id = (*r)->address.first + (*r)->address.second * n.cols
       ();
22  this->startniST(r_id);
    this->startrouterST(r_id);
24  for(timeslot t = 0; t < n.best; t++){ // Write table row for
       each timeslot
      int t0 = t-1;
26    int t1 = t;
      if(t == 0){
28      t0 = n.best-1;
        t1 = n.best;
30    }
      // Write row in Network Adapter table
32    router_id dest_id = (*r)->address;
      router_id src_id = (*r)->address;
34    if ((*r)->local_in_best_schedule.has((t+2)%n.best)){
        dest_id = (*r)->local_in_best_schedule.get((t+2)%n.best)->
            to;
36    }
      if ((*r)->local_out_best_schedule.has(t1))
38      src_id = (*r)->local_out_best_schedule.get(t1)->from;

40    int dest = dest_id.first + dest_id.second * n.cols();
      int src = src_id.first + src_id.second * n.cols();
42
      this->writeSlotNIDest(t,countWidth,dest);
44    this->writeSlotNISrc(src);
      // Write row in Router table
46    //this->writeSlotRouter(t,countWidth,ports);
      port_id ports[5] = {__NUM_PORTS, __NUM_PORTS, __NUM_PORTS,
          __NUM_PORTS, __NUM_PORTS};
48
      for(int out_p = 0; out_p < __NUM_PORTS-1; out_p++){
50      // For all 4 output ports not being the local port.
        if(!(*r)->out((port_id)out_p).has_link()){
52        continue; // No outgoing channel from the port.
        }
54      if((*r)->out((port_id)out_p).link().best_schedule.has(t)){
          // If there is a channel comming out of the port, find
              the
56        // input port from which the channel is comming from.
          const channel* out_c =(*r)->out((port_id)out_p).link().
              best_schedule.get(t);
58        for(int in_p = 0; in_p < __NUM_PORTS-1; in_p++){
            // For all 4 input ports not being the local port.
```

```
60              if(!(*r)->in((port_id)in_p).has_link())
                  continue; // No link into this port
62              if((*r)->in((port_id)in_p).link().best_schedule.has(t0)
                    ){ // REMEMBER: Change back t-1 -> t
                  const channel* in_c =(*r)->in((port_id)in_p).link().
                      best_schedule.get(t0);
64                if(out_c == in_c){
                    // The correct link found
66                  ports[(port_id)out_p] = (port_id)in_p;
                    break;
68                }
                }
70            }
              if(ports[(port_id)out_p] == __NUM_PORTS){
72              // If channel was not found on any of the 4 input ports
                    .
                // It should be on the local in port, but we test it
                    anyway.
74              if((*r)->local_in_best_schedule.has(t)){
                  const channel* in_c = (*r)->local_in_best_schedule.
                      get(t);
76                if(out_c == in_c){
                    // The correct link found
78                  ports[(port_id)out_p] = L;
                  } else {
80                  cout << "Failure: Channel rose from nothing like a
                        fenix." << endl;
                  }
82              }
              }
84          } else {
              ports[(port_id)out_p] = __NUM_PORTS;
86          }
            //ports[N] = ;
88
        }
90      if((*r)->local_out_best_schedule.has(t1)){ // For the local
            out port.
          const channel* out_c = (*r)->local_out_best_schedule.get(t1
              );
92        for(int in_p = 0; in_p < __NUM_PORTS-1; in_p++){
            // For all 4 input ports not being the local port.
94          if(!(*r)->in((port_id)in_p).has_link())
              continue; // No link into this port
96          if((*r)->in((port_id)in_p).link().best_schedule.has(t0)){
              const channel* in_c =(*r)->in((port_id)in_p).link().
                  best_schedule.get(t0);
98            if(out_c == in_c){
                // The correct link found
100             ports[L] = (port_id)in_p;
                break;
102           }
            }
104       }
          if(ports[L] == __NUM_PORTS){
```

```
106             // If channel was not found on any of the 4 input ports.
                // It should be on the local in port, but we test it
                    anyway.
108             cout << "Failure: Not allowed to route back in to local."
                     << endl;
              }
110           // and so on...
            }
112         this->writeSlotRouter(t,countWidth,ports);

114       }
          this->endniST(r_id);
116       this->endrouterST(r_id);
        }
118 //    n.router(e)->next;
        //n.routers().at(n).local_out_best_schedule.get(t).from
120

122     this->endArchRouter();
        this->endArchNI();
124     delete this;
        return true;
126 }

128 string vhdlOutput::bin(int val, int bits) {
        int max = (int)pow(2.0,bits-1);
130     string s = "";
        for(int i = 0; i < bits; i++){
132       if(val/max >= 1){
            val -= max;
134         s += "1";
          } else {
136         s += "0";
          }
138       max = max / 2;
        }
140     return s;
    }
142
    char vhdlOutput::p2c(port_id p){
144       char c;
          if (p == N) c = 'N';
146       if (p == E) c = 'E';
          if (p == S) c = 'S';
148       if (p == W) c = 'W';
          if (p == L) c = 'L';
150       if (p == __NUM_PORTS) c = 'D';

152       return c;
        }
154
    vhdlOutput::vhdlOutput(string output_dir){
156     niST.open(output_dir + "ni_ST_" + numOfNodesStr + ".vhd", ios::
            trunc);
```

```
    routerST.open(output_dir + "router_ST_" + numOfNodesStr + ".vhd",
        ios::trunc);
158 if(!niST.good()){
      niST.close();
160   string new_file = output_dir + ::lex_cast<string>((int)time(
          NULL)) + "ni_ST_" + numOfNodesStr + ".vhd";
      cout << "Warning: Output failure, new output name: " + new_file
          << endl;
162   niST.open(new_file, ios::trunc);
    }
164 if(!routerST.good()){
      routerST.close();
166   string new_file = output_dir + ::lex_cast<string>((int)time(
          NULL)) + "router_ST_" + numOfNodesStr + ".vhd";
      cout << "Warning: Output failure, new output name: " + new_file
          << endl;
168   routerST.open(new_file, ios::trunc);
    }
170 // TODO: Error handling + Specify output file name
  }
172
  vhdlOutput::~vhdlOutput(){
174 niST.close();
    routerST.close();
176 }

178 void vhdlOutput::writeHeaderRouter(int countWidth){
    routerST << "
        ------------------------------------------------------------\
        n";
180 routerST << "-- router_ST_" << numOfNodesStr << ".vhd\n";
    routerST << "-- This is an auto generated file, do not edit by
        hand.\n";
182 routerST << "-- These tables were generated from an application
        specific\n";
    routerST << "-- schedule by the SNTs project.\n";
184 routerST << "-- https://github.com/rbscloud/SNTs\n";
    routerST << "
        ------------------------------------------------------------\
        n";
186 routerST << "library ieee;\n";
    routerST << "use ieee.std_logic_1164.all;\n";
188 routerST << "use ieee.numeric_std.all;\n\n";

190 routerST << "use work.noc_types.all;\n\n";

192 routerST << "entity router_ST_" << numOfNodesStr << " is\n";
    routerST << "\tgeneric (\n";
194 routerST << "\t\tNI_NUM\t: natural);\n";
    routerST << "\tport (\n";
196 routerST << "\t\tcount\t: in unsigned(" << countWidth-1 << "
        downto 0);\n";
    routerST << "\t\tsels\t: out select_signals\n";
198 routerST << "\t\t);\n";
    routerST << "end router_ST_" << numOfNodesStr << ";\n\n";
```

```
200
    routerST << "architecture data of router_ST_" << numOfNodesStr <<
        " is\n";
202   routerST << "begin -- data\n\n";

204 }

206 void vhdlOutput::endArchRouter(){
    routerST << "end data;\n";
208 }

210 void vhdlOutput::writeSlotRouter(int slotNum, int countWidth,
      port_id* ports){
    routerST << "\t\twhen \"" << bin(slotNum,countWidth) << "\" =>\n"
        ;
212   routerST << "\t\t\tsels(N) <= " << p2c(ports[N]) << ";\n";
    routerST << "\t\t\tsels(E) <= " << p2c(ports[E]) << ";\n";
214   routerST << "\t\t\tsels(S) <= " << p2c(ports[S]) << ";\n";
    routerST << "\t\t\tsels(W) <= " << p2c(ports[W]) << ";\n";
216   routerST << "\t\t\tsels(L) <= " << p2c(ports[L]) << ";\n";
  }
218
  void vhdlOutput::writeHeaderNI(int countWidth, int numOfNodes){
220   niST << "
        ------------------------------------------------------------------\
        n";
    niST << "-- ni_ST_" << numOfNodesStr << ".vhd\n";
222   niST << "-- This is an auto generated file, do not edit by hand.\
        n";
    niST << "-- These tables were generated from an application
        specific\n";
224   niST << "-- schedule by the SNTs project.\n";
    niST << "-- https://github.com/rbscloud/SNTs\n";
226   niST << "
        ------------------------------------------------------------------\
        n";
    niST << "library ieee;\n";
228   niST << "use ieee.std_logic_1164.all;\n";
    niST << "use ieee.numeric_std.all;\n\n";
230
    niST << "use work.noc_types.all;\n\n";
232
    niST << "entity ni_ST_" << numOfNodesStr << " is\n";
234   niST << "\tgeneric (\n";
    niST << "\t\tNI_NUM\t: natural);\n";
236   niST << "\tport (\n";
    niST << "\t\tcount\t: in unsigned(" << countWidth-1 << " downto
        0);\n";
238   niST << "\t\tdest\t: out integer range 0 to " << numOfNodes-1 <<
        ";\n";
    niST << "\t\tsrc\t: out integer range 0 to " << numOfNodes-1 << "
        \n";
240   niST << "\t\t);\n";

242   niST << "end ni_ST_" << numOfNodesStr << ";\n\n";
```

```
  niST << "architecture data of ni_ST_" << numOfNodesStr << " is\n"
      ;
244  niST << "begin -- data\n\n";

246 }

248 void vhdlOutput::startniST(int num){
  startST(num,&this->niST);
250 }

252 void vhdlOutput::startrouterST(int num){
  startST(num,&this->routerST);
254 }

256 void vhdlOutput::startST(int num, ofstream* ST){
  *ST << "\tNI_NUM" << num << " : if NI_NUM = " << num << "
      generate\n";
258  *ST << "\tprocess(count) begin\n\n";
  *ST << "\t\tcase count is\n\n";
260 }

262 void vhdlOutput::writeSlotNIDest(int slotNum, int countWidth, int
    dest){
  niST << "\t\t\twhen \"" << bin(slotNum,countWidth) << "\" =>\n";
264  niST << "\t\t\t\tdest <= " << dest << ";\n";
}
266
void vhdlOutput::writeSlotNISrc(int src){
268  niST << "\t\t\t\tsrc <= " << src << ";\n";
}
270
void vhdlOutput::endrouterST(int num){
272  routerST << "\t\twhen others =>\n";
  routerST << "\t\t\tsels(N) <= D;\n";
274  routerST << "\t\t\tsels(E) <= D;\n";
  routerST << "\t\t\tsels(S) <= D;\n";
276  routerST << "\t\t\tsels(W) <= D;\n";
  routerST << "\t\t\tsels(L) <= D;\n";
278  routerST << "\t\tend case;\n";
  routerST << "\tend process;\n\n";
280  routerST << "\tend generate NI_NUM" << num << ";\n\n";
}
282
void vhdlOutput::endniST(int num){
284  niST << "\t\t\twhen others =>\n";
  niST << "\t\t\t\tdest <= " << num << ";\n";
286  niST << "\t\t\t\tsrc <= " << num << ";\n\n";
  niST << "\t\tend case;\n";
288  niST << "\tend process;\n\n";
  niST << "\tend generate NI_NUM" << num << ";\n\n";
290 }

292 void vhdlOutput::endArchNI(){
  niST << "end data;\n";
294 }
```

Listing D.6: ScheduleConverter.java

```
2  package dk.rbscloud.tcrest.SNTs;

4  import java.io.File;
   import java.util.ArrayList;
6  import java.util.List;
   import javax.xml.parsers.*;
8  import org.w3c.dom.*;

10 /**
   * A converter from xml format to Setup of DMA tables
12 * @author Rasmus
   */
14 public class ScheduleConverter{
     private enum Port {N, E, S, W, L, D}
16   private static List<List<List<Integer> > > initArray;
     private static final int SLOT_TABLE = 0;
18   private static final int ROUTE_TABLE = 1;
     private static Document doc;
20   private static NodeList tList;

22   public static void main(String[] args) {
       if(args.length < 1){
24       System.out.println("No input file specified!");
         return;
26     }
       parseXml(args[0]);
28     try {
         initializeArray(tList.getLength());
30       int numOfNodes = tList.getLength();
         new TileCoord(0,0,(int)Math.sqrt(numOfNodes)); //
             Initializing the static sideLength variable in TileCoord
32
         /* For each tile, slot table and route table is written. */
34       for (int tileIdx = 0; tileIdx < tList.getLength(); tileIdx++)
             {
           Node tile = tList.item(tileIdx);
36         if (tile.getNodeType() == Node.ELEMENT_NODE) {
             Element tileE = (Element) tile;
38           TileCoord tileCoord = getTileCoord(tile);
             /* For each tile write the slot table */
40           NodeList slotList = tileE.getElementsByTagName("timeslot"
                 );

42           int slotTableWidth = (int)Math.ceil(Math.log(numOfNodes)/
                 Math.log(2));
             /* For each time slot, slot table and route table is
                 written. */
44           for (int slotIdx = 0; slotIdx < slotList.getLength();
                 slotIdx++) {
               Node slot = slotList.item(slotIdx);
46             if (slot.getNodeType() == Node.ELEMENT_NODE) {
                 Element slotE = (Element) slot;
```

```
48                // Get the coordinates of the receiver for this
                      timeslot
                  TileCoord destCoord = getDestCoord(slotE);
50
                  // Write the destination ID in the slot table.
52                int slotVal = destCoord.getTileId() | (1 <<
                      slotTableWidth);
                  initArray.get(tileCoord.getTileId()).get(SLOT_TABLE).
                      add(slotIdx,slotVal);
54
                  /* For each transmission slot write an entry in the
                      route table */
56                String binRoute = "";
                  if (destCoord.getTileId() != tileCoord.getTileId()){
58                  TileCoord tempTileCoord = new TileCoord(tileCoord.x
                        ,tileCoord.y);
                    char inPort = 'L';
60                  for(int i = 0; tempTileCoord.getTileId() !=
                        destCoord.getTileId(); i++){
                    NodeList ports = getPorts(tempTileCoord,slotIdx+i
                        );
62                  char outPort = findOutputPort(ports,inPort);
                    binRoute = port2bin(outPort) + binRoute;
64                  inPort = oppositPort(outPort);
                    nextTile(tempTileCoord,outPort);
66                  }
                    // Route to local port
68                  binRoute = port2bin(inPort) + binRoute;
                  }
70                int route = Integer.parseInt("0" + binRoute, 2);
                  // Write the route to the route table in the
                      initArray.
72                initArray.get(tileCoord.getTileId()).get(ROUTE_TABLE)
                      .set(destCoord.getTileId(), route);
                }
74            }
            }
76        }

78    } catch (Exception e) {
        e.printStackTrace();
80    }
      SchedulePrinter printer = new SchedulePrinter();
82    printer.printData(initArray);
      printer.printFooter();
84  }

86  private static NodeList getPorts(TileCoord tileCoord, int slotIdx
        ){
      Element tileE = (Element) getTile(tileCoord);
88    NodeList sList = tileE.getElementsByTagName("timeslot");
      slotIdx = (slotIdx + 2) % sList.getLength(); // The schedule
          takes pipelining into acount.
90                          // Shifting by 2 gets rid of the
                                pipelining.
```

```
    Element slotE = (Element) sList.item(slotIdx);
92  NodeList rList = slotE.getElementsByTagName("router");
    Node router = rList.item(0);
94  Element routerE = (Element) router;
    return routerE.getElementsByTagName("output");
96  }

98  private static char findOutputPort(NodeList ports, char inPort){
    char outPort = ' ';
100 for(int nodeIdx = 0; nodeIdx < ports.getLength(); nodeIdx++){
      if(ports.item(nodeIdx).getAttributes().getNamedItem("input").
          getNodeValue().charAt(0) == inPort){
102     outPort = ports.item(nodeIdx).getAttributes().getNamedItem(
            "id").getNodeValue().charAt(0);
      }
104 }
    return outPort;
106 }

108 private static char oppositPort(char p){
    if(p == 'N'){p = 'S';}
110 else if(p == 'E'){p = 'W';}
    else if(p == 'S'){p = 'N';}
112 else if(p == 'W'){p = 'E';}
    else{p = 'L';}
114 return p;
    }
116
    private static void nextTile(TileCoord tileCoord, char outPort){
118 if(outPort == 'N'){tileCoord.moveNorth();}
    else if(outPort == 'E'){tileCoord.moveEast();}
120 else if(outPort == 'S'){tileCoord.moveSouth();}
    else if(outPort == 'W'){tileCoord.moveWest();}
122 // If local port do nothing
    }
124
    private static String port2bin(char p){
126 String bin;
    if(p == 'N'){bin = "10";}
128 else if(p == 'E'){bin = "11";}
    else if(p == 'S'){bin = "00";}
130 else if(p == 'W'){bin = "01";}
    else{bin = "00";}
132 return bin;
    }
134
    private static Node getTile(TileCoord tileCoord){
136 for (int tileIdx = 0; tileIdx < tList.getLength(); tileIdx++) {
        // For each tile
      Node tile = tList.item(tileIdx);
138   if (tile.getNodeType() == Node.ELEMENT_NODE) {
        if(tileCoord.getTileId() == getTileCoord(tile).getTileId())
            {
140       return tile;
        }
```

```
142         }
      }
144     return tList.item(0);
    }
146
    private static TileCoord getTileCoord(Node tile){
148     String[] tileCoord = tile.getAttributes().getNamedItem("id").
            getNodeValue().split("\\D");
      TileCoord tileId = new TileCoord(Integer.parseInt(tileCoord[1])
          , Integer.parseInt(tileCoord[2]));
150     return tileId;
    }
152
    private static TileCoord getDestCoord(Element slotE) {
154     String[] coord = slotE.getElementsByTagName("na").item(0).
            getAttributes().getNamedItem("tx").getNodeValue().split("\\
            D");
      TileCoord destCoord = new TileCoord(Integer.parseInt(coord[1]),
           Integer.parseInt(coord[2]));
156     return destCoord;
    }
158
    private static void parseXml(String inputFile){
160     try{
        File fXmlFile = new File(inputFile);
162       DocumentBuilderFactory dbFactory = DocumentBuilderFactory.
              newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
164       doc = dBuilder.parse(fXmlFile);
        doc.getDocumentElement().normalize();
166       tList = doc.getElementsByTagName("tile");
      } catch (Exception e) {
168       e.printStackTrace();
      }
170   }

172   private static void initializeArray(int nrCpu){
      initArray = new ArrayList<List<List<Integer> > >(nrCpu);
174     for (int i = 0; i < nrCpu; i++) {
        initArray.add(new ArrayList<List<Integer> >(2));
176       initArray.get(i).add(new ArrayList<Integer>());
        initArray.get(i).add(new ArrayList<Integer>(nrCpu));
178       for(int j = 0; j < nrCpu; j++){
          initArray.get(i).get(ROUTE_TABLE).add(0);
180       }
      }
182   }
}
```

Listing D.7: SchedulePrinter.java

```
package dk.rbscloud.tcrest.SNTs;
2
import java.io.*;
4 import java.util.List;
```

```
6  public class SchedulePrinter{
   private static FileWriter ofile;
8  private static int indent = 0;
   private static final int SLOT_TABLE = 0;
10 private static final int ROUTE_TABLE = 1;

12
   public SchedulePrinter(){
14   try{
       ofile = new FileWriter(new File("../Tables.java"));
16     String str =  ind() + "/**"
             + ind() + " * AUTO-Generated file DO NOT EDIT!!!"
18           + ind() + " * Loads the pre calculated schedule into
                 the Slot and Route tables."
             + ind() + " * @author package dk.rbscloud.tcrest.SNTs"
20           + ind() + " */"
             + ind() + "package dk.rbscloud.tcrest.API;"
22           + ind() + "import com.jopdesign.sys.Native;"
             + ind() + ""
24           + ind() + "public class Tables" + openBrac()
             + ind() + "public static final int[][][] initArray = "
                 + openBrac()
26           + ind() + "";
/*             public static final int[][][] initArray = {
28             {
                 {1,2,3,4,5},
30               {1,2,3}
               },
32             {
                 {1,2,3,4,5},
34               {1,2,3}
               }
36           };
*/
38     ofile.write(str);
     } catch (Exception e) {
40     e.printStackTrace();
     }
42 }

44 public void printFooter(){
     try{
46     String str =  closeBrac() + ";"
             + ind() + "private static int[] getSlotTable(int cpuId)
                 " + openBrac()
48           + ind() + "return initArray[cpuId][0];"
             + closeBrac()
50           + ind() + ""
             + ind() + "private static int[] getDmaTable(int cpuId)"
                  + openBrac()
52           + ind() + "return initArray[cpuId][1];"
             + closeBrac()
54           + ind() + ""
```

```
            + ind() + "public static void load(int cpuId)" +
                openBrac()
            + ind() + "// Loading the slot table"
            + ind() + "int[] slotTable = Tables.getSlotTable(cpuId)
                ;"
            + ind() + "for(int i = 0; i < slotTable.length; i++)" +
                 openBrac()
            + ind() + "Native.wrMem(slotTable[i], Const.
                SLOT_TBL_BASE+i);"
            + closeBrac()
            + ind() + "// Loading the dma table"
            + ind() + "int[] dmaTable = Tables.getDmaTable(cpuId);"
            + ind() + "for(int i = 0; i < dmaTable.length; i++)" +
                openBrac()
            + ind() + "Native.wrMem(dmaTable[i], Const.DMA_P_BASE+i
                );"
            + closeBrac()
            + closeBrac()
            + ind() + ""
            + ind() + "public static boolean verify(int cpuId)" +
                openBrac()
            + ind() + "// Reading and verifying the dma table"
            + ind() + "int[] dmaTable = getDmaTable(cpuId);"
            + ind() + "for(int i = 0; i < dmaTable.length; i++)" +
                openBrac()
            + ind() + "int dmaData = Native.rdMem(Const.DMA_P_BASE+
                i);"
            + ind() + "if (dmaData != dmaTable[i])" + openBrac()
            + ind() + "System.out.println(\"DMA_P_BASE faliure\");"
            + ind() + "return false;"
            + closeBrac()
            + closeBrac()
            + ind() + "return true;"
            + closeBrac()
            + closeBrac()
            + ind();
        ofile.write(str);
        ofile.close();
      } catch(Exception e) {
        e.printStackTrace();
      }
    }

    public void printData(List<List<List<Integer>>> initArray){
      String str = "";
      for(List<List<Integer>> initCpu: initArray){
        str += ind() + openBrac()
            + ind() + "{";
        for(int slot : initCpu.get(SLOT_TABLE)){
          str += slot + ",";
        }
        str = str.substring(0, str.length()-1);
        str += "}," + ind() + "{";
        for(int route: initCpu.get(ROUTE_TABLE)){
```

```
        str += route + ",";
      }
      str = str.substring(0, str.length()-1);
      str += "}" + closeBrac() + ",";
    }
    str = str.substring(0, str.length()-1);
    try{
      ofile.write(str);
    } catch(Exception e){
      e.printStackTrace();
    }
  }


  private String ind(){
    String str = "\n";
    if (indent < 0) {
      return "";
    }
    for (int i = 0; i < indent; i++) {
      str += "\t";
    }
    return str;
  }

  private String openBrac(){
    indent++;
    return "{";
  }

  private String closeBrac(int b){
    String str = "";
    for (int i = 0 ; i < b; i++) {
      str += closeBrac();
    }
    return str;
  }

  private String closeBrac(){
    if (indent > 0) {
      indent--;
    }
    String str = ind();
    return str + "}";
  }
}
```

Listing D.8: TileCoord.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package dk.rbscloud.tcrest.SNTs;

```

```
/**
 *
 * @author Rasmus
 */
public class TileCoord {
  public int x, y;
  private static int sideLength;
  public TileCoord(int x, int y, int sideLength){
    this.x = x;
    this.y = y;
    this.sideLength = sideLength;
  }
  public TileCoord(int x, int y){
    this.x = x;
    this.y = y;
  }
  public int getTileId(){
    return x+y*sideLength;
  }

  public void moveNorth(){
    if(this.y == 0){
      this.y = sideLength-1;
    } else {
      this.y--;
    }
  }

  public void moveSouth(){
    if(this.y == sideLength-1){
      this.y = 0;
    } else {
      this.y++;
    }
  }
  public void moveEast(){
    if(this.x == sideLength-1){
      this.x = 0;
    } else {
      this.x++;
    }
  }
  public void moveWest(){
    if(this.x == 0){
      this.x = sideLength-1;
    } else {
      this.x--;
    }
  }

}
```

Appendix E

MPI source code

This appendix contains the following files:

NoC.java is the static Java class implementing the communication primitives, the file starts on page 133.

Const.java is a Java file with constants describing the address space of the MPI, the file starts on page 136.

Tables.java is an example file generated by the schedule converter, it starts on page 137.

Listing E.1: NoC.java

```
1  package dk.rbscloud.tcrest.API;
2
3  import com.jopdesign.sys.Native;
4
5  /**
6   *
7   * @author Rasmus Bo Soerensen
8   */
9  public class NoC {
10   public static boolean send(int[] msg, int destCpuId, int cpuId){
11     if((destCpuId == cpuId) || (msg.length > 8)){ return false;}
12     int destDMA = Const.DMA_BASE+(destCpuId << 1);
```

```
13      while(!doneDMA(destDMA));
        int txBufAddr = getTxBuf(cpuId, destCpuId);
15  //    System.out.println("txBufAddr: " + txBufAddr);
        copyInMsg(msg,txBufAddr);
17      int localChanBuf = getChanBufAddr(cpuId,destCpuId);
        int destChanBuf = getChanBufAddr(destCpuId,cpuId);
19      int writePointer = destChanBuf*Const.CHANNEL_BUF_SIZE+swapBuf(
            Const.TX_ACT_BUF,localChanBuf);
        int readPointer = localChanBuf*Const.CHANNEL_BUF_SIZE + Const.
            TX_BUF;
21  //    System.out.println("SRC: "+cpuId+" DEST: "+destCpuId+"
        WritePointer: " + writePointer);
23  //    System.out.println("readPointer: " + readPointer);
  //    System.out.println("destDMA: " + destDMA);
        setupDMA(msg,readPointer,writePointer,destDMA);
25      return true;
    }
27
    public static boolean sendRdy(int destCpuId){
29    int destDMA = Const.DMA_BASE+(destCpuId << 1);
      return doneDMA(destDMA);
31  }

33  public static boolean recv(int[] msg, int srcCpuId, int cpuId){
      if(srcCpuId == cpuId){
35      return false;
      }
37    int rxBufAddr = getRxBuf(cpuId, srcCpuId);
      while(!doneRecv(rxBufAddr));
39    copyOutMsg(msg,rxBufAddr);
      int localChanBuf = getChanBufAddr(cpuId,srcCpuId);
41    swapBuf(Const.RX_ACT_BUF,localChanBuf);
      //System.out.println("Swap: "+);
43    //if(msg.length > 8){ return false;}
      return true;
45  }

47  public static boolean recvRdy(int srcCpuId, int cpuId){
      if(srcCpuId == cpuId){
49      return false;
      }
51    int rxBufAddr = getRxBuf(cpuId, srcCpuId);
      return doneRecv(rxBufAddr);
53  }

55  private static boolean doneRecv(int addr){
      int length = Native.rdMem(addr);
57    if(length != 0){
        if(Native.rdMem(addr+length+1) == -1){ return true;}
59    }
      return false;
61  }

63  private static void setupDMA(int[] msg, int txBufAddr, int
        rxBufAddr, int addrDMA){
```

```
    Native.wrMem((txBufAddr << 16) | rxBufAddr, addrDMA + 1);
    if((msg.length & 1) == 0){
      Native.wrMem(msg.length+2 | 32768, addrDMA);
    } else{
      Native.wrMem(msg.length+3 | 32768, addrDMA);
    }
  }

  private static boolean doneDMA(int addrDMA){
    int DMA = Native.rdMem(addrDMA);
    if((DMA & 32768) != 0){
      if((DMA & 16384) == 0){ return false;}
    }
    return true;
  }

  private static void copyInMsg(int[] msg, int addr){
    Native.wrMem(msg.length, addr);
    for(int i = 1; i < msg.length+1; i++){
      Native.wrMem(msg[i-1], addr+i);
    }
    Native.wrMem(-1, addr+msg.length+1);
  }

  private static void copyOutMsg(int[] msg, int addr){
    int length = Native.rdMem(addr);
    //Native.wrMem(0, addr);
    for(int i = 1; i < length+1; i++){
      msg[i-1] = Native.rdMem(addr+i);
    //   Native.wrMem(0, addr+i);
    }
    //Native.wrMem(0, addr+length+1);
    for(int i = 0; i < Const.BUFFER_SIZE; i++){
      Native.wrMem(0, addr+i);
    }
  }

  public static int getTxBuf(int cpuId, int destCpuId){
    int bufAddr = getChanBufAddr(cpuId, destCpuId);
    return Const.NI_BASE + (bufAddr * Const.CHANNEL_BUF_SIZE) +
        Const.TX_BUF;
  }

  public static int getRxBuf(int cpuId, int srcCpuId){
    int bufAddr = getChanBufAddr(cpuId, srcCpuId);
    return Const.NI_BASE + (bufAddr * Const.CHANNEL_BUF_SIZE) +
        getActBuf(bufAddr);
  }

  private static int getChanBufAddr(int cpuId, int channId){
    if(channId == Const.NUMBER_OF_CORES-1){
      channId = cpuId;
    }
    return channId;
  }
```

```
117
  private static int getActBuf(int bufAddr){
119   int actBufAddr = Const.NI_BASE + (bufAddr * Const.
        CHANNEL_BUF_SIZE) + Const.RX_ACT_BUF;
      int actBuf = Native.rdMem(actBufAddr);
121   if((actBuf & 4) == 0) { return Const.RX_BUF_1;}
      else { return Const.RX_BUF_2;}
123 }

125 private static int swapBuf(int statusAddr, int bufAddr){
      int actBufAddr = Const.NI_BASE + (bufAddr * Const.
        CHANNEL_BUF_SIZE) + statusAddr;
127   int actBuf = Native.rdMem(actBufAddr);
//    System.out.println("actBuf: "+actBuf);
129   int newActBuf = actBuf ^ 4;
//    System.out.println("newActBuf: "+newActBuf);
131 //    System.out.println("actBufAddr: "+actBufAddr);
      Native.wr(newActBuf, actBufAddr);
133 //    Native.wr(10, 4194432);
//    int j = 0;
135 //    for(int i = 0; i < 1000; i++){
//      j = i+2;
137 //    }
//    int ret = Native.rd(4194432);
139 //    System.out.println("Read from mem: " + ret);
//    System.out.println("newActBuf: "+Native.rdMem(actBufAddr));
141 //    System.out.println("j: "+j);

143   if((actBuf & 4) == 0){ return Const.RX_BUF_1;}
      else { return Const.RX_BUF_2;}
145 }

147 public static void checkSPM(){
      int verified = 0;
149   for(int i = 0; i < Const.COM_SPM_SIZE; i++){
        Native.wr(i, i+Const.COM_SPM);
151     if(Native.rd(i+Const.COM_SPM) != i){
          System.out.println("SPM error at address: " + i);
153     } else {
          verified++;
155     }
      }
157   System.out.println(verified + " Addresses verified out of " +
        Const.COM_SPM_SIZE + " Addresses.");
      for(int i = 0; i < Const.COM_SPM_SIZE; i++){
159     Native.wr(0, i+Const.COM_SPM);
      }
161 }
}
```

Listing E.2: Const.java

```
package dk.rbscloud.tcrest.API;
2

/**
```

```
 * Constants for the T-CREST DMA Network Interface
 * @author Rasmus Bo Soerensen
 */
class Const {
  /* DMA Network Interface addresses */
  public static final int NI_BASE = 0x400000;
  public static final int COM_SPM = NI_BASE;
  public static final int DMA_BASE = NI_BASE + 0x80000;
  public static final int DMA_P_BASE = NI_BASE + 0x100000;
  public static final int SLOT_TBL_BASE = NI_BASE + 0x180000;
  public static final int CONFIG_DONE = NI_BASE + 0x100;

  // Channel buffer constants
  public static final int COM_SPM_SIZE = 256;
  public static final int NUMBER_OF_CORES = 9;
  public static final int CHANNEL_BUF_SIZE = COM_SPM_SIZE/(
      NUMBER_OF_CORES-1);
  public static final int BUFFER_SIZE = (CHANNEL_BUF_SIZE - 2) / 3;
  public static final int RX_ACT_BUF = 0;
  public static final int TX_ACT_BUF = 1;
  public static final int RX_BUF_1 = 2 + BUFFER_SIZE*0;
  public static final int RX_BUF_2 = 2 + BUFFER_SIZE*1;
  public static final int TX_BUF = 2 + BUFFER_SIZE*2;
}
```

Listing E.3: Tables.java

```

/**
 * AUTO-Generated file DO NOT EDIT!!!
 * Loads the pre calculated schedule into the Slot and Route tables
     .
 * @author package dk.rbscloud.tcrest.SNTs
 */
package dk.rbscloud.tcrest.API;
import com.jopdesign.sys.Native;

public class Tables{
  public static final int[][][] initArray = {

    {
      {17,18,19,20,24},
      {0,7,13,8,28,0,0,0,54}
    },
    {
      {17,16,17,17,17},
      {13,0,0,0,0,0,0,0,0}
    },
    {
      {18,18,18,16,17},
      {7,13,0,0,0,0,0,0,0}
    },
    {
      {16,19,19,18,19},
      {2,0,54,0,0,0,0,0,0}
```

```
28      },
        {
30        {20,20,20,16,19},
          {54,0,0,13,0,0,0,0,0}
32      },
        {
34        {21,21,21,20,21},
          {0,0,0,0,13,0,0,0,0}
36      },
        {
38        {22,22,22,21,22},
          {0,0,0,0,0,54,0,0,0}
40      },
        {
42        {23,23,23,22,23},
          {0,0,0,0,0,0,13,0,0}
44      },
        {
46        {24,24,24,23,24},
          {0,0,0,0,0,0,0,13,0}
48      }
    };
50  private static int[] getSlotTable(int cpuId){
      return initArray[cpuId][0];
52  }

54  private static int[] getDmaTable(int cpuId){
      return initArray[cpuId][1];
56  }

58  public static void load(int cpuId){
      // Loading the slot table
60    int[] slotTable = Tables.getSlotTable(cpuId);
      for(int i = 0; i < slotTable.length; i++){
62      Native.wrMem(slotTable[i], Const.SLOT_TBL_BASE+i);
      }
64    // Loading the dma table
      int[] dmaTable = Tables.getDmaTable(cpuId);
66    for(int i = 0; i < dmaTable.length; i++){
        Native.wrMem(dmaTable[i], Const.DMA_P_BASE+i);
68    }
    }
70
    public static boolean verify(int cpuId){
72    // Reading and verifying the dma table
      int[] dmaTable = getDmaTable(cpuId);
74    for(int i = 0; i < dmaTable.length; i++){
        int dmaData = Native.rdMem(Const.DMA_P_BASE+i);
76      if (dmaData != dmaTable[i]){
          System.out.println("DMA_P_BASE faliure");
78        return false;
        }
80    }
      return true;
82  }
```

```
}
```

Appendix F

Test and benchmark source code

This appendix contains the following files:

HelloDMA.java is the Hello World program sending a message between all processors before writing Hello World to the console, the file starts on page 141.

DMABench.java is the microbenchmark program for the T-CREST NoC platform, the file starts on page 143.

Listing F.1: HelloDMA.java

```
1  /*
     This file is part of JOP, the Java Optimized Processor
3      see <http://www.jopdesign.com/>

5    Copyright (C) 2005-2008, Martin Schoeberl (martin@jopdesign.com)

7    This program is free software: you can redistribute it and/or
         modify
     it under the terms of the GNU General Public License as published
          by
9    the Free Software Foundation, either version 3 of the License, or
     (at your option) any later version.
11
```

```
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/
       licenses/>.
*/


/**
 *
 */
package cmp;

import java.util.Vector;

import joprt.RtThread;

import com.jopdesign.io.IOFactory;
import com.jopdesign.io.SysDevice;
import com.jopdesign.sys.Const;
import com.jopdesign.sys.Native;
import com.jopdesign.sys.Startup;
import dk.rbscloud.tcrest.API.*;

/**
 * A CMP version of Hello World
 *
 * @author Rasmus
 *
 */
public class HelloDMA implements Runnable {

  int id;

  static Vector msg;

  public HelloDMA(int i) {
    id = i;
  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    Tables.load(0);
    SysDevice sys = IOFactory.getFactory().getSysDevice();
    msg = new Vector();
    System.out.println("Core 0 started");
    for (int i=0; i<sys.nrCpu-1; ++i) {
      Runnable r = new HelloDMA(i+1);
      Startup.setRunnable(r, i);
    }
```

```
    // start the other CPUs
67  sys.signal = 1;
    int[] message = {0,1,2,3,4,5,6,7};
69  int[] rmessage = {0,0,0,0,0,0,0,0};
    NoC.send(message,sys.nrCpu-1,0);
71
    for (;;) {
73    int size = msg.size();
      if (size!=0) {
75      StringBuffer sb = (StringBuffer) msg.remove(0);
        System.out.println(sb);
77    }
      if(NoC.recvRdy(1,0)){
79      NoC.recv(rmessage,1,0);
        for(int i = 0; i < message.length; i++){
81        if(message[i] != rmessage[i]){ System.exit(1);}
        }
83      System.out.println("Hello World!");
      }
85  }
  }
87
  public void run() {
89  Tables.load(id);
    StringBuffer sb = new StringBuffer();
91  sb.append("Core ").append(id).append(" started");
    RtThread.sleepMs(300*id);
93  msg.addElement(sb);
    int[] message = {0,0,0,0,0,0,0,0};
95  int src = id+1;
    if(id == 8){ src = 0;}
97  for(;;){
      NoC.recv(message,src,id);
99    NoC.send(message,id-1,id);
    }
101 }
}
```

Listing F.2: DMABench.java

```
/*
2   This file is part of JOP, the Java Optimized Processor
      see <http://www.jopdesign.com/>
4
    Copyright (C) 2005-2008, Martin Schoeberl (martin@jopdesign.com)
6
    This program is free software: you can redistribute it and/or
        modify
8   it under the terms of the GNU General Public License as published
        by
    the Free Software Foundation, either version 3 of the License, or
10  (at your option) any later version.

12  This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
14   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
     GNU General Public License for more details.
16
     You should have received a copy of the GNU General Public License
18   along with this program.  If not, see <http://www.gnu.org/
         licenses/>.
*/
20

22 /**
 *
24 */
package cmp;
26
import java.util.Vector;
28
import joprt.RtThread;
30
import com.jopdesign.io.IOFactory;
32 import com.jopdesign.io.SysDevice;
import com.jopdesign.sys.Const;
34 import com.jopdesign.sys.Native;
import com.jopdesign.sys.Startup;
36 import dk.rbscloud.tcrest.API.*;

38 /**
 * A DMA Benchmark
40 *
 * @author Rasmus
42 *
 */
44 public class DMABench implements Runnable {

46   int id;

48   static Vector msg;

50   public DMABench(int i) {
       id = i;
52   }

54   /**
      * @param args
56    */
     public static void main(String[] args) {
58
       msg = new Vector();
60     System.out.println("Core 0 started");
       Tables.load(0);
62     SysDevice sys = IOFactory.getFactory().getSysDevice();
       int[] message = IOFactory.getFactory().getScratchpadMemory();
64     for(int i = 0; i < message.length; i++){
         message[i] = i;
66     }
//     int[] message = {4,4,7,8768,456,34,6,27}; // 8
```

```
68 //     int[] message = {4,4,7,8768,456,34,6};     // 7
   //     int[] message = {4,4,7,8768,456,34};     // 6
70 //     int[] message = {4,4,7,8768,456};      // 5
   //     int[] message = {4,4,7,8768};        // 4
72 //     int[] message = {4,4,7};           // 3
   //     int[] message = {4,4};             // 2
74 //     int[] message = {4};             // 1

76     for (int i=0; i<sys.nrCpu-1; ++i) {
         Runnable r = new DMABench(i+1);
78       Startup.setRunnable(r, i);
       }
80
       // start the other CPUs
82     sys.signal = 1;

84     // Print out start messages from other cores
       for (int printOut = 0; printOut < sys.nrCpu-1;) {
86       int size = msg.size();
         if (size!=0) {
88         StringBuffer sb = (StringBuffer) msg.remove(0);
           System.out.println(sb);
90         printOut++;
         }
92     }

94     int tRead, t0, t1;
       int time, sum, res;
96     int i;
       int iterations = 10;
98     // Find timing overhead
       t0 = Native.rdMem(Const.IO_CNT);
100    t1 = Native.rdMem(Const.IO_CNT);
       tRead = t1 - t0;
102    System.out.println("Timing overhead\t\t= " + tRead);
       // Find avarage read time
104    sum = 0;
       for(i = 0; i < iterations; i++){
106      t0 = Native.rdMem(Const.IO_CNT);
         res = Native.rdMem(0x400000);
108      t1 = Native.rdMem(Const.IO_CNT);
         time = t1-t0-tRead;
110 //      System.out.println("Read time = " + time);
         sum += time;
112    }
       time = sum/iterations;
114    System.out.println("Avg read time\t\t= " + time);
       // Find avarage write time
116    sum = 0;
       for(i = 0; i < iterations; i++){
118      t0 = Native.rdMem(Const.IO_CNT);
         Native.wrMem(0,0x40000A);
120      t1 = Native.rdMem(Const.IO_CNT);
         time = t1-t0-tRead;
122 //      System.out.println("Write time = " + time);
```

```
      sum += time;
    }
    time = sum/iterations;
    System.out.println("Avg write time\t\t= " + time);
    // Time for single calculation
    t0 = Native.rdMem(Const.IO_CNT);
    message[0] = message[0] + 1;
//    i = i + 1;
    t1 = Native.rdMem(Const.IO_CNT);
    time = t1-t0-tRead;
    System.out.println("Message[0]++\t\t= " + time);

    // Find round trip time
    sum = 0;
    for(i = 0; i < iterations; i++){
      t0 = Native.rdMem(Const.IO_CNT);
      NoC.send(message,sys.nrCpu-1,0);
      NoC.recv(message,1,0);
      t1 = Native.rdMem(Const.IO_CNT);
      time = t1-t0-tRead;
//      System.out.println("Write time = " + time);
      sum += time;
    }
    time = sum/iterations;
    System.out.println("Avg roundtrip time\t= " + time);

    // Find interleaved round trip time
    sum = 0;
    for(i = 0; i < iterations; i++){
      t0 = Native.rdMem(Const.IO_CNT);
      NoC.send(message,sys.nrCpu-1,0);
      NoC.send(message,sys.nrCpu-1,0);
      NoC.recv(message,1,0);
      NoC.recv(message,1,0);
      t1 = Native.rdMem(Const.IO_CNT);
      time = t1-t0-tRead;
//      System.out.println("Write time = " + time);
      sum += time;
    }
    time = sum/iterations;
    System.out.println("Avg interleaved roundtrip time\t= " + time)
        ;

    // Find Echo time
    sum = 0;
    for(i = 0; i < iterations; i++){
      t0 = Native.rdMem(Const.IO_CNT);
      NoC.send(message,1,0);
      NoC.recv(message,1,0);
      t1 = Native.rdMem(Const.IO_CNT);
      time = t1-t0-tRead;
//      System.out.println("Write time = " + time);
      sum += time;
    }
    time = sum/iterations;
```

```
    System.out.println("Avg echo time\t\t= " + time);
178
180     // Find interleaved echo time
        sum = 0;
182     for(i = 0; i < iterations; i++){
          t0 = Native.rdMem(Const.IO_CNT);
184       NoC.send(message,2,0);
          NoC.send(message,2,0);
186       NoC.recv(message,2,0);
          NoC.recv(message,2,0);
188       t1 = Native.rdMem(Const.IO_CNT);
          time = t1-t0-tRead;
190 //        System.out.println("Write time = " + time);
          sum += time;
192     }
        time = sum/iterations;
194     System.out.println("Avg interleaved echo time\t= " + time);

196     // Find time for send
        sum = 0;
198     for(i = 0; i < iterations; i++){
          t0 = Native.rdMem(Const.IO_CNT);
200       NoC.send(message,3,0);
          t1 = Native.rdMem(Const.IO_CNT);
202       NoC.recv(message,3,0);
          time = t1-t0-tRead;
204 //        System.out.println("Write time = " + time);
          sum += time;
206     }
        time = sum/iterations;
208     System.out.println("Avg send time\t\t= " + time);

210     // Find time for recv
        sum = 0;
212     for(i = 0; i < iterations; i++){
          NoC.send(message,4,0);
214       while(!NoC.recvRdy(4,0));
          t0 = Native.rdMem(Const.IO_CNT);
216       NoC.recv(message,4,0);
          t1 = Native.rdMem(Const.IO_CNT);
218       time = t1-t0-tRead;
//        System.out.println("Write time = " + time);
220       sum += time;
        }
222     time = sum/iterations;
        System.out.println("Avg recv time\t\t= " + time);
224

226 //    for (;;) {
//        if(NoC.recvRdy(1,0)){
228 //        t0 = Native.rdMem(Const.IO_CNT);
//          NoC.recv(message,3,0);
230 //        t1 = Native.rdMem(Const.IO_CNT);
//          time = t1-t0-tRead;
```

```
232 //          System.out.println("Receive time = " + time);
    //          t0 = Native.rdMem(Const.IO_CNT);
234 //          NoC.send(message,3,0);
    //          t1 = Native.rdMem(Const.IO_CNT);
236 //          time = t1-t0-tRead;
    //          System.out.println("Send time = " + time);
238 //          message[0] = message[0] + 1;
    //        }
240 //      }
      }

242
      public void run() {
244     Tables.load(id);
        //NoC.checkSPM();
246     StringBuffer sb = new StringBuffer();
        if(!Tables.verify(id)){
248       sb.append("Schedule failure: CPU ").append(id).append("\n");
        }
250     sb.append("Core ").append(id).append(" started");
        RtThread.sleepMs(300*id);
252     msg.addElement(sb);
        //sb.delete(0, sb.length());
254     int[] message = IOFactory.getFactory().getScratchpadMemory();
        for(int i = 0; i < message.length; i++){
256       message[i] = i;
        }
258 //      int[] message = {0*id,1*id,2*id,3*id,4*id,5*id,6*id,7*id};
        // 8
    //      int[] message = {0*id,1*id,2*id,3*id,4*id,5*id,6*id};    // 7
260 //      int[] message = {0*id,1*id,2*id,3*id,4*id,5*id};       // 6
    //      int[] message = {0*id,1*id,2*id,3*id,4*id};          // 5
262 //      int[] message = {0*id,1*id,2*id,3*id};             // 4
    //      int[] message = {0*id,1*id,2*id};                // 3
264 //      int[] message = {0*id,1*id};                   // 2
    //      int[] message = {0*id};                      // 1
266     int iterations = 10;

268     int src = id+1;
        if(id == 8){
270       src = 0;
        }
272     // Roundtrip measurments
        for(int i = 0; i < iterations*3; i++){
274       NoC.recv(message,src,id);
          message[0] = message[0] + 1;
276       NoC.send(message,id-1,id);
        }
278
        // Echo measurements
280     for(int i = 0; i < iterations*2; i++){
          NoC.recv(message,0,id);
282       message[0] = message[0] + 1;
          NoC.send(message,0,id);
284     }
      }
```

```
286 }
```

Bibliography

[1] Rasmus Bo Sørensen, Martin Schoeberl, and Jens Sparsø. A light-weight statically scheduled network-on-chip. In *30th NorChip Conference*, 2012.

[2] Scott Hansen. T-crest project. Project webpage http://t-crest.org, 2012.

[3] Martin Schoeberl, Pascal Schleuniger, Wolfgang Puffitsch, Florian Brandner, Christian W. Probst, Sven Karlsson, and Tommy Thorn. Towards a Time-predictable Dual-Issue Microprocessor: The Patmos Approach. In *Bringing Theory to Practice: Predictability and Performance in Embedded Systems*, volume 18 of *OASICS 18 Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik*, pages 11–21, Mar 2011.

[4] Martin Schoeberl. *JOP Reference Handbook: Building Embedded Systems with a Java Processor*. Number ISBN 978-1438239699. CreateSpace, August 2009. Available at `http://www.jopdesign.com/doc/handbook.pdf`.

[5] K. Goossens and A. Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 306 –311, jun 2010.

[6] T. Bjerregaard and J. Sparsø. A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip. In *date*, pages 1226–1231. IEEE Computer Society Press, 2005.

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

[8] Martin Schoeberl, Florian Brandner, Jens Sparsø, and Evangelia Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, Lyngby, Denmark, May 2012. IEEE.

[9] Martin Schoeberl. Leros: A tiny microcontroller for FPGAs. In *Proceedings of the 21st International Conference on Field Programmable Logic and Applications (FPL 2011)*, Chania, Crete, Greece, September 2011. IEEE Computer Society.

[10] Jens Sparsø, Evangelia Kasapaki, and Martin Schoeberl. An area-efficient network adaptor for a tdm-based network-on-chip. In *Design, Autimation and Test in Europe Conference and Exhibition 2013*, 2013. Accepted to DATE'13.

[11] Martin Schoeberl. SimpCon - a simple and efficient SoC interconnect. In *Proceedings of the 15th Austrian Workhop on Microelectronics, Austrochip 2007*, Graz, Austria, October 2007.

[12] OCP-IP. Open core protocol specification. Technical report, 2012. Available at `http://www.ocpip.org/uploads/dynamic_areas/Xu4qydXgbYWof7Ihz3Uh/947/Open%20Core%20Protocol%20Specification%203.0.pdf`.

[13] Maurizio Palesi, Shashi Kumar, and Rickard Holsmark. A method for router table compression for application specific routing in mesh topology noc architectures. In Stamatis Vassiliadis, Stephan Wong, and TimoD. Hämäläinen, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, volume 4017 of *Lecture Notes in Computer Science*, pages 373–384. Springer Berlin Heidelberg, 2006.

[14] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Routing table minimization for irregular mesh nocs. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 942–947, San Jose, CA, USA, 2007. EDA Consortium.

[15] Wikipedia.org. Rencontres numbers, 2012. `http://en.wikipedia.org/wiki/Rencontres_numbers`.

[16] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multicommodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184 –193, oct. 1975.

[17] Florian Brandner and Martin Schoeberl. Static routing in symmetric real-time network-on-chips. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, RTNS '12, pages 61–70, New York, NY, USA, 2012. ACM.

[18] K. Goossens, A. Radulescu, and A. Hansson. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on*, pages 75 –80, sept. 2005.

[19] Mark Ruvald Pedersen, Jaspur Højgaard, and Rasmus Bo Sørensen. Scheduling in a real-time network-on-chip. Technical report, Department of Informatics and Mathematical Modelling, Technical University of Denmark, 2012.

[20] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, November 2006.

[21] Thomas A. Feo and Mauricio G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, March 1995.

[22] `pugixml.org`. Light-weight, simple and fast xml parser for c++ with xpath support, 2012.

[23] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, AUG 1978. Reprinted in "Distributed Computing: Concepts and Implementations" edited by McEntire, O'Reilly and Larson, IEEE, 1984.

[24] G. Kahn. The semantics of a simple language for parallelprogramming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[25] MPI-forum. *MPI: A Message-Passing Interface Standard Version 3.0*. MPI-forum, 2012. Available at `http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf`.

[26] open mpi.org. Open mpi: Open source high performance computing, 2012. Available at `http://www.open-mpi.org/`.

[27] Cristian Grecu, Andrè Ivanov, Patha Pande, Axel Jantsch, Erno Salmimem, Umit Ogras, and Radu Marculescu. An initiative towards open network-on-chip benchmarks. Technical report, OCP-IP, 2007.