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.

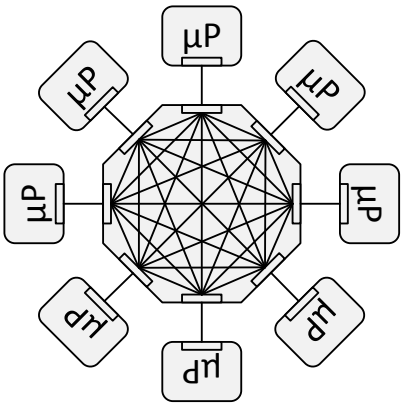


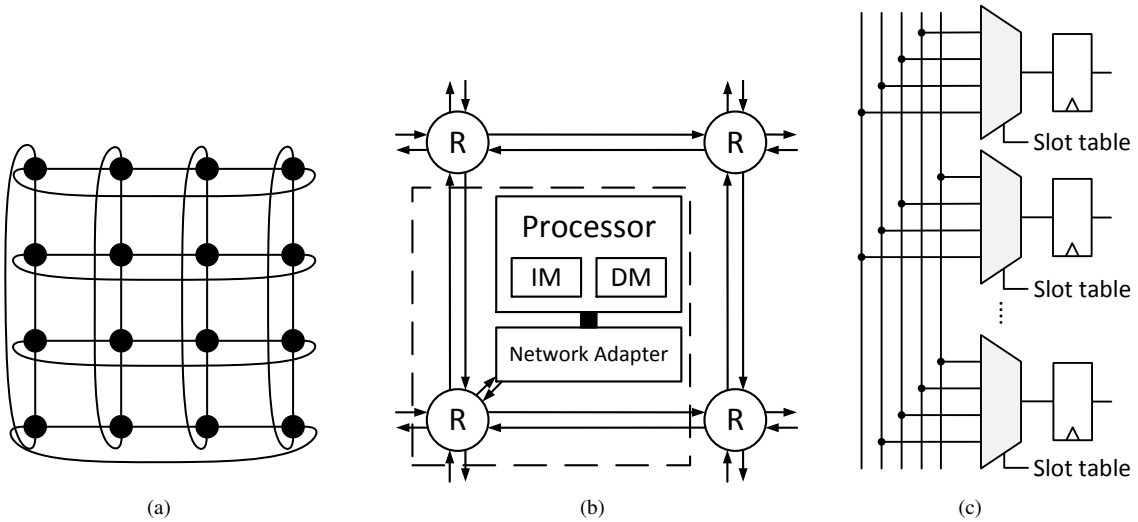
Fig. 1. A conceptual interconnect providing all-to-all connection between micro processors (μP).

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



(a) (b) (c)

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

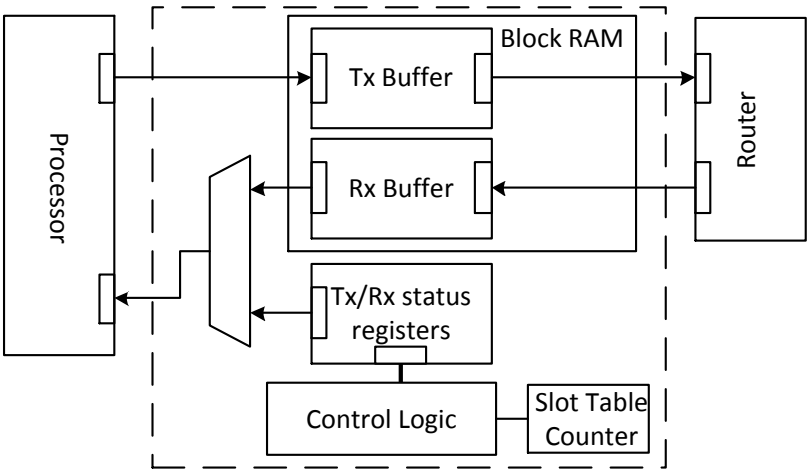


Fig. 3. A tile including the Leros processor, the network adapter with receive, transmit, and status registers, the interface to the router, and the router.

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.