A TIME-TRIGGERED NETWORK-ON-CHIP

Martin Schoeberl

Institute of Computer Engineering
Vienna University of Technology, Austria
mschoebe@mail.tuwien.ac.at

ABSTRACT

In this paper we propose a time-triggered network-on-chip (NoC) for on-chip real-time systems. The NoC provides time predictable on- and off-chip communication, a mandatory feature for dependable real-time systems. A regular structured NoC with a pseudo-static communication schedule allows for a high bandwidth. In this paper we argue for a simple, time-triggered NoC structure to achieve maximum bandwidth. We have implemented the proposed TT-NoC in a low-cost FPGA. The base bandwidth is 29 Gbit/s and the peak bandwidth 230 Gbit/s for eight nodes. The idea is in line with current on-chip multiprocessor designs, such as the Cell processor. The simple design of the network and the network interface easies certification of the proposed NoC for safety critical applications.

1. INTRODUCTION

The concept of the time-triggered architecture (TTA) [1] is well established for dependable real-time systems. The main application of the TTA is on communication over a time-triggered bus. In this paper we apply the TTA concepts to an on-chip communication channel usually named network-on-chip (NoC). Using the TTA on-chip provides the same real-time guaranties in the communication as it provides for a *classical* time-triggered bus – an off-chip serial bus such as TTP/C [2] or TT-Ethernet [3]. Common within this time-triggered protocol (TTP) family is the fact that the TTP schedule determines the schedule of tasks within a node. This mechanism provides a straightforward synchronization between the communication controller and the host. The proposed NoC has to be validated against safety critical standards such as DO-178B [4]. We rely on the TTA for the fault containment. As certification is expensive we aim for a simple network design and a minimal *trusted area* to isolate the real-time nodes.

Figure 1 shows an example configuration of a time-triggered (TT) NoC with on-chip and off-chip nodes. All nodes are connected via the network interface (NI) to the on-chip network. In this example one node, the digital signal processing (DSP) unit, is directly connected to the NI. Two microprocessors, the on-chip CPU and the off-chip host, are connected via a message multiplexer (M-Mux) to the NI. One possibility for a high bandwidth connection is via a DMA unit. When the SoC is part of a larger TT cluster, communication and synchronization with the cluster time is provided via a TT-Ethernet [3] interface.

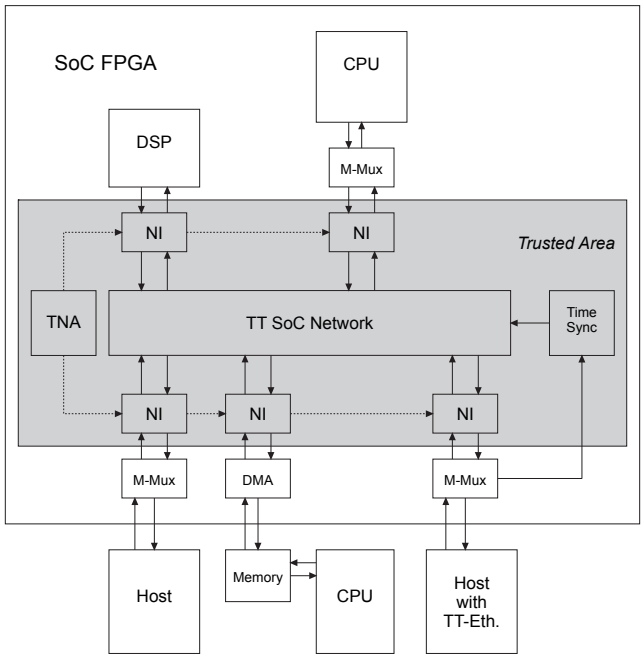

Fig. 1. TT SoC configuration example

The proposed network is intended to be part of a distributed, safety-critical real-time system. As a consequence the communication system has to be certified. We call the area that has to be certified the trusted area, shaded in Figure 1. It contains: the core network, the NI, the trusted network authority (TNA), and the time synchronization unit. To ease certification the M-Mux and other higher level components are not part of the trusted area.

Network-on-Chip (NoC) is an active research area. The various projects are described in a recent survey [5]. Sonics μNetwork [6] is based on a pipelined bus with a TDMA approach to guarantee latency and pipeline requirements. The TDMA frame is divided into 256 cycles that can be pre-allocated for real-time cores. The pre-allocation of send slots is similar to our proposal. The difference is in the organization of

the bus: The μNetwork is a pipelined bus, where we use a ring structure where each segment can be used individually (similar to the Cell architecture).

The Cell multiprocessor [7, 8, 9] shares some ideas with the proposed TT NoC. The Cell implements a very regular NoC. The Cell contains, beside a PowerPC microprocessor, 8 synergistic processors. The bus is clocked at half of the processor speed (1.6 GHz). It is organized in 4 rings each 128 bit wide, two in each direction. A maximum of 3 non-overlapping transfers on each ring are possible. This would result in a peak bandwidth of 307 Gbyte/s. The true limit results from the address resolution from the central arbiter: one address per cycle. With all messages of the maximum size of 128 bytes the theoretical upper limit is 128 byte $\times$ 1.6 GHz = 204.8 Gbyte/s. Although our NoC looks very similar to the bus architecture in the Cell there are major differences: Due to our pseudo-static schedule we have no arbitration phase. The bus can be switched each clock cycle between message forward and send. All bus segments can be used by independent transfers compared to maximal three concurrent transfers in the Cell. We use message buffers in the NI whereas in Cell the messages are directly streamed into and out of the node local memory.

Compared to the described NoC designs we use a network clock that is independent of the individual node clocks. Due to the simple interconnection structure this clock is usually higher than a node clock. We perform the clock domain crossing in the NI and the message synchronization at the higher level of the TTA. The clock domain crossing in the NI also simplifies system composition of nodes running at different local clocks. The network clock serves as the reference time base for the TTA. The time base can be synchronized (at a higher granularity) over SoC boundaries.

The rest of the paper is organized as follows. Section 2 introduces the time-triggered message scheduling. Section 3 discusses the proposed network topology and some possible variations. In Section 4 we describe the central part of the network, the network interface. We present the results from an implementation in a low-cost FPGA in Section 5. For the further discussion we assume a bus width of 128 bit and a network clock of 225 MHz as used in this implementation.

2. TIME AND MESSAGE SCHEDULING

Time-triggered messages are sent periodically. The period and the phase of the message are predefined and known a-priori by all nodes. This schedule is pseudo-static and can only be changed by the TNA to adapt to different bandwidth needs of an application through a mode change. This pseudo-static schedule eliminates any needs of dynamic bus arbitration. This property results in three important features of the network: (a) message transmission is time predictable, (b) no addresses or message identifiers have to be transmitted, and (c) it allows for a high network clock. Furthermore, the schedule table contains an entry for each clock cycle. That means we can transmit a different message from a different node at each clock tick resulting in a very fine grained resource allocation. The schedule table is part of the NI (see Section 4).

In a distributed system (a TT cluster), where the SoC is just one node, the internal periods are synchronized with the cluster time. The cluster time can by synchronized via GPS to the global time. In this configuration we accomplish a time-triggered architecture from the low-level NoC up to the system level.

The network clock and the clocks of the nodes are different and unrelated resulting in several clock domains. Only the network time is directly synchronized with the cluster time. To accomplish this synchronization we choose a slightly faster clock for the network than the nominal clock for the schedule. The synchronization of the network time to the cluster time is accomplished by inserting empty slots dynamically into the network schedule. The nodes are time synchronized at the scheduler level by the time-triggered interrupts received from the NI.

A new message is sent each network clock cycle. As the bus is pipelined the message latency is several cycles. Using a ring topology the fixed latency depends on the location of the sender and the receiver. Therefore, each NI has its own distinct schedule. The size of the schedule table (e.g. 1024 entries) restricts the maximum period to e.g. 4.6 μs. The minimum period, when two nodes consume the whole network bandwidth, is 4.4 ns.

To adapt the NoC to different modes with different bandwidth needs the bus schedule can be changed by the TNA. Update of the schedule shall not disturb slots that are not affected by the schedule change. The correct order of disabling and enabling the receivers and transmitters in several steps allow for a non disturbing schedule change.

3. THE NETWORK TOPOLOGY

The underlying network topology is simple and optimized for easy routing (wire routing, not message routing) and a high bandwidth. As we are interested in maximizing the bandwidth and can tolerate a known latency we will build a register intensive design. Using many registers is a natural design practice in FPGAs and also helps to compensate for long wire delays[1] between processing nodes. We choose an actual low-cost FPGA, Alteras Cyclone II [10], for the implementation.

When we go for maximum system frequency we shall use the simplest structure possible. A simple structure for a bus is a ring. A ring connecting regular nodes, as the NIs are expected to be, should be easy to route. This structure was also chosen in the Cell processor [8] with the very regular layout for the 8 processor elements.

[1] At lower feature sizes the wire delay constrains the maximum system frequency more than the combinatorial delay.

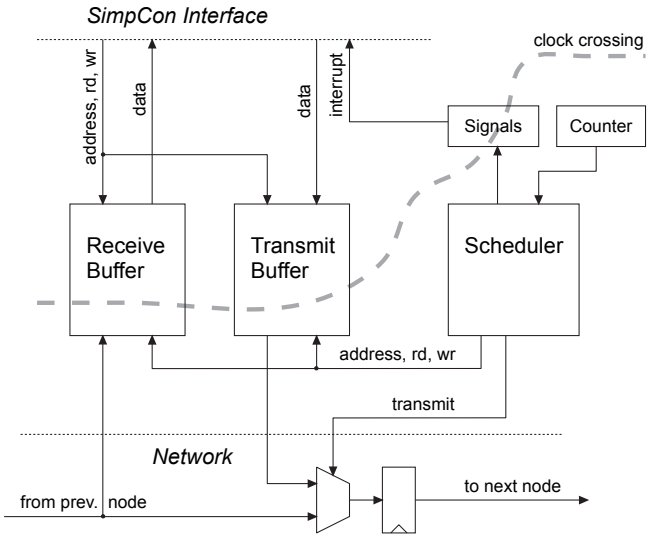

Fig. 2. Network interface for time-triggered messages

For more flexibility (and more bandwidth) we can dynamically cut (with the multiplexer as shown in Figure 2) the bus at each node when not using broadcast mode[2]. We achieve a theoretical peak bandwidth n (the number of nodes) times the broadcast bandwidth (BW). For our example design this results in:

$$\begin{aligned} BW &= 128 \text{ bit} \times 225 \text{ MHz} = 29 \text{ Gbit/s} \\ BW_{peak} &= n \times BW = 8 \times 29 \text{ Gbit/s} = 230 \text{ Gbit/s} \end{aligned}$$

The bandwidth for a single node is limited by the on-chip memory. However, with more than one ring we add the flexibility to come closer to the cumulative peak bandwidth for concurrent messages. At the other end of the spectrum we can reduce the bus size to the typical host size of 32 bit to reduce resource usage (especially on-chip memory). Further variations that can be considered: a bus that is wider than the message buffer data size – this is very similar to multiple busses; or a bus that is clocked higher than the message buffers.

4. THE NETWORK INTERFACE

The network interface (NI) is the link between a node (in the general case) and the on-chip network. The NI has following properties: standard interface for nodes; several message ports; single buffer for each port; scheduling of the messages; clock domain crossing; belongs to the trusted area.

Adaptation to different host types is provided by a NI/host translation unit, for an example see Section 4.3. As the NI belongs to the trusted area it has to be as simple as possible to ease verification.

Figure 2 shows the block diagram of the NI. The components that make up the network interconnection are the multiplexer and the register. The multiplexer decides whether a message is forwarded from the previous node or a message from the current node is inserted into the network. This decision is driven by the scheduler at each network clock cycle. The register implements the network pipeline with one pipeline stage per node.

[2]Broadcast mode is meant when a message is received by all nodes.

index	in	out	wr	rd	tx	intAB	intCD
0			0	0	0	0	0
1	A		1	0	0	0	0
2		B_1	0	1	1	0	0
3		B_2	0	1	1	1	0
4			0	0	0	0	0
...							
8	C	D	1	1	1	0	1
9			0	0	0	0	0
10			0	0	0	0	0
...							
16			0	0	0	0	0
17	A		1	0	0	0	0
18		B_1	0	1	1	0	0
19		B_2	0	1	1	1	0
20			0	0	0	0	0
...							

Table 1. Example of a schedule table

The transmit and receive buffers are connected via a standard SoC interface [11]. The data bus width for this interface can be configured between 8 and 128 bit. The interface provides atomic read and write operation at the network width (i.e. for 128 bit words) between the host and the message buffer. Furthermore, time-triggered interrupts (or signals) are provided by the NI. The scheduler decides when a message is transferred from the bus to the receive buffer and when a message from the transmit buffer is placed on the bus. The scheduler is a simple table indexed by a counter. When the on-chip network is synchronized to an off-chip network idle cycles have to be inserted. Therefore, the scheduler becomes slightly more complex. The size of the message buffers and the schedule table is configurable and can be adapted for different application domains.

An example of a schedule table is given in Table 1. The *index* is the address of the schedule entry as generated by the counter. The column *in* is the address into the receive buffer and the column *out* the address into the transmit buffer. Signals *wr* and *rd* select writing or reading a message into or from the buffers, signal *tx* controls the network multiplexer. The symbolic message names in Table 1 represent the addresses of the messages pointing into the transmit and receive buffers. *intAB* and *intCD* are two signals (interrupts in Figure 2) sent from the NI to the node to indicate that a message has been sent or received.

The example contains the four messages (or ports) A, B, C, and D. A and B both have a period of 16 clock cycles, C and D are messages with a longer period. A and C are input

messages – we can see that the *wr* signal for the receive buffer is 1 at index 1, 8, and 17. *B* and *D* are output messages – the *rd* signal goes to 1 for the transmit buffer and the *tx* signal selects the multiplexer. In our example *A* is a broadcast message. Therefore, it is read into the receive buffer *and* routed to the next node. *B* is a longer message that takes two send slots. Message *C* is a message that is not further routed to the next node. In that case we can use the same slot to transmit message *D*. All fixed, known latencies between the different nodes are incorporated into the schedule table. The schedule table is generated by the TNA and transmitted to the NI.

4.1. Schedule Update

To adapt the NoC to different application modes with different bandwidth needs the bus schedule can be updated by the TNA. Update of the schedule shall not disturb slots that are not affected by the schedule change. It has to be noted that each NI has its own distinct schedule table. As a consequence the update can not happen in one instant. It has to be performed incrementally.

With a carefully change of the schedule table on each node no bus idle time for a schedule change is necessary. The schedule update does not need to happen at the same time for all nodes.

4.2. Time-Triggered Messages

Time-triggered messages avoid resource conflicts per design. The host network interface is therefore very simple and needs no explicit synchronization. A dual-port message buffer with one buffer per host port is all we need (simultaneous access to the same address is prevented by the TT schedule). To further simplify the implementation (and increase the possible system frequency) we use one memory for output messages and a second memory for input messages. As an additional service we provide time-triggered signals (interrupts). The signals are used by the host as the reference time-base for the timer-triggered scheduler.

4.3. Message Multiplexer

Several different *hosts* can be connected to the TT-NoC: on-chip host (microprocessor), off-chip host via a memory type interface, DMA, TT-Ethernet controller, and on-chip special purpose hardware. From this list we can conclude that there will not be a single host interface that fits for all host types. However, we can use a minimal interface (the NI) that belongs to the trusted area and a translation unit to the actual host. One example of such a translation unit is the message multiplexer.

The bit-clock frequency of an off-chip network is usually lower (or equal) to the operation frequency of the CPU inside a node (e.g. a 100 Mbit network connection to a GHz CPU). On-chip busses can be designed much simpler than on-chip IP cores resulting in a higher frequency of the bus than the operating frequency from e.g. an on-chip CPU. As an example we can design an on-chip bus in a low-cost FPGA that runs at 400 MHz[3]. In the same technology a soft-core CPU can be clocked at several 10 MHz: A RISC processor (e.g. LEON3 SPARC V8 [12]) at 35 MHz, the Java processor JOP [13, 14] at 100 MHz. This gap is architecture inherent and will not be closed when we use an ASIC as implementation technology.

The bus width (e.g. 128 bit) of an on-chip bus is wider than the processors register size (32 bit). This factor further widens the bandwidth gap. The bandwidth gap is even larger when the nodes are off-chip and the SoC only serves as the network. It is at the limit on current FPGAs to provide a cumulative off-chip bandwidth of 100 Gbit/s by using all high-speed serial IO pins.

There is a big gap between the NoC bandwidth and a microprocessor memory interface. Furthermore, the maximum period in the NoC is in the range of several μs which is way too short for a time-triggered scheduling on the host. To bridge this gap we propose a message multiplexer (M-Mux). The M-Mux handles several ports of the host (each with a longer period) and multiplexes them into one slot of the NoC schedule. In the NoC scheduling we have a maximum period that is restricted by the simple schedule table. The M-Mux has to handle scheduling decisions every few μs instead of ns as the NoC scheduler. Therefore, we can use a more complex storage layout of the scheduling table to provide longer message periods at the M-Mux level with lower memory requirements.

5. IMPLEMENTATION

We have implemented the proposed time-triggered network in an FPGA (Cyclone II EP2C35 [10]) on the Altera DE2 board. The bus is a single 128 bit ring. The NI contains 128×128 bit transmit and receive buffers. The individual nodes are connected by a 32 bit SimpCon [11] interface with 128 bit atomic read and write.

The schedule table contains 512 entries with read and write addresses for the message buffers, read and write signals, and one interrupt. The network is clocked at 225 MHz, the maximum frequency we can achieve with the on-chip memories and the PLL clocked from a 50 MHz source. The individual nodes are clocked at 100 MHz. The clock domain crossing is performed in the message buffers and for the interrupt. The interrupt signal is widened to several network clock cycles at the network side for a safe transmission to the node side. The resulting broadcast bandwidth is 29 Gbit/s (225 MHz $\times$ 128 bit). The theoretical peak bandwidth for

[3] In our experiments we have implemented a 256 bit wide bus at 413 MHz in a Cyclone II device resulting in a bandwidth of 106 Gbit/s. A wider bus with 2048 bits can be clocked with 298 MHz in the same technology and results in a bandwidth of 610 Gbit/s.

n nodes is $n \times 29$ Gbit/s. For eight nodes this results in 230 Gbit/s.

5.1. Memory Blocks

Message buffers and the schedule table are implemented with on-chip memory blocks. A single on-chip memory block is 4 Kbit and can be configured for a data width between 1 and 32 (independent for both ports). For a 128 bit interface four RAM blocks are needed. With four RAM blocks for the message buffers the f_{max} is 235 MHz.

In our first design we used different port width for the on-chip memories to provide the 32/128 bit mapping between the host side and the network side. However, using independent clocks and independent port sizes can result in erroneous behavior in Cyclone II Rev. A and B devices [15]. The Quartus work-around results in an f_{max} of 210 MHz for the network and 203 MHz for the interface side (8 nodes). Quartus uses the memory with equal port sizes (the larger one), does not register the outputs of the memory, and inserts a registered multiplexer at the output. To compensate for this issue we use the memories with equal port size, fully registered and add the bus resizing logic explicitly within an additional pipeline stage. In this additional pipeline stage we also implement the atomic read and write functionality.

5.2. Demo Application

We have also implemented a simple demo application (a voting triple modular redundancy sensor – a classic safety critical component) that consists of following nodes: the Java processor JOP [13, 14], three sensor nodes, one voting node, and one actuator node.

The three sensor nodes send a periodic message with their actual value. The voting node performs the majority voting and the actuator node displays the result. The processor reads the sensor raw data and the voting result. On a change the values are reported by the processor.

The schedule for this example was created by a small tool written in Java. However, getting the schedule correct with the different latencies of this heavily pipelined design is quite a challenge. A more advanced tool to calculate and verify the schedule, even online in the TNA for mode changes, is of primary importance to render the proposed NoC useful.

5.3. Results

A full NI for the 128 bit bus needs just 480 logic cells (LC). This resource usage is quite low compared to other designs. PNoC [16] is a lightweight circuit-switched NoC. A four port router at a data width of only 32 bit (compared to the 128 bits we use in the example design) for PNoC is reported to consume 732 LCs (366 Virtex-II slices).

For the NI of a node that uses only one direction and not the full 128 bit, such as the sensor node, the resource usage drops down to about 170 LCs. To set these numbers into relation: the soft-core processor JOP consumes about 2,200 LCs in a standard configuration with the additional NI connection. The low-cost FPGA (medium size) used for this prototype contains 33,000 LCs.

# Nodes	# JOP	LC	Memory	f_{max}
6	1	4,068	174 Kbit	235 MHz
7	2	6,894	244 Kbit	235 MHz
9	4	12,543	383 Kbit	235 MHz
13	8	22,704	661 Kbit	235 MHz
17	12	33,558	939 Kbit	235 MHz

Table 2. Scalability of the NoC design with several non-trivial network nodes

The main resource limitation is in the message and schedule buffer. A complete NI needs 10 on-chip memory blocks of 4 Kbit each[4]. The EP2C35 contains 105 on-chip memory blocks. If we implement a message multiplexer (see Section 4.3) on top of the NI additional on-chip memory is necessary. The on-chip memory in the Cyclone II also limits the maximum bus frequency to 235 MHz. According to Altera this slow memory (compared to the predecessor Cyclone in an older technology where the memory can be clocked up to 250 MHz) is a tribute to the lower cost. Compiling the design for the high-performance/high-cost FPGA family Stratix II the network can be clocked at 390 MHz, resulting in a broadcast bandwidth of 50 Gbit/s and a peak bandwidth of 400 Gbit/s for 8 nodes. However, we want to provide an affordable solution and stay with the low-cost FPGA series.

5.4. Scalability

We have performed some experiments to evaluate the scalability of the design. The assumed main issue with scalability is the pressure on routing resources with several non-trivial network nodes. We start from our initial prototype with one processor (JOP) and 5 simple nodes and add additional processors. Table 2 shows the resulting resource usage (LCs and on-chip memory bits) and the maximum network frequency for the different configurations.

The number of network nodes in the table is # JOP plus 5. The first entry is the baseline from the prototype. The experiment was restricted by the available on-chip memory of 1 Mbit in the largest Cyclone II device (EP2C70) used for this compilation experiment. For all compilation runs we have set the network PLL to 250 MHz to force the compiler to optimize for speed and to report the maximum achievable frequency. With this setting the LC usage is 10–14% higher than with a setting to the realistic 225 MHz.

From this experiment we conclude that our simple and

[4] It has to be noted that the size of the on-chip memory blocks is determined by the FPGA architecture. For an ASIC implementation, or a different FPGA such as Stratix II, smaller memories for the message buffers can be used.

heavily pipelined NoC design scales quite well. The largest design with 12 processors and 5 simple nodes is still bandwidth limited by the on-chip memory and not the routing infrastructure.

5.5. Discussion

For a simple node, such as the sensor or voter, the 128 bit message buffer with the 32 bit atomic read/write interface is an overkill. Simple nodes that transmit or receive only a few messages can be connected to the bus without the message buffer to reduce the memory consumption. With careful coding those nodes can be implemented to run with the fast network clock rendering the logic for the clock domain crossing useless. As a result a hardware node can be quite cheap – an interesting aspect for future designs.

The trusted part of the scheduling to avoid wrong routing, the on-chip version of a collision, is actually only a single bit entry in the schedule table. That bit controls the multiplexer whether the node's own data or the data from the left neighbor is routed to the next node. Therefore, only this single bit entry and the generation of the correct schedule for this schedule entry have to be certified for a safety critical system.

6. CONCLUSION

With the proposed time-triggered NoC we provide a high-performance SoC interconnection with time predictable delivery of real-time messages. The NoC builds the basis for higher level services for distributed real-time systems. The combination of the TTA and a SoC network provides quite new opportunities for the system design. The TTA with a run-time static schedule provides real-time guaranties and simplifies the architecture of the on-chip network. As future work we consider adding the message multiplexer to the design for slower nodes and longer periods. We will integrate the TNA that calculates online new schedules for mode changes and updates the schedule tables in the NIs. When building a network with more nodes a second ring in the other direction can reduce the latency for half of the nodes. The proposed TT NoC is intended to be a part of a larger time-triggered network, a cluster of SoCs connected via TT-Ethernet [3]. In this configuration the on-chip time will be synchronized with the cluster time. The cluster time itself can be synchronized with the world time via GPS. In this case the on-chip network time and the message periods are in sync with the world time.

Acknowledgments

This work has been supported in part by the European IST project DECOS under project No. IST-511764. This work is only a small part of the research on a time-triggered, fault tolerant SoC. The author thanks Hermann Kopetz, Bernhard Huber, Christian El Salloum, and Roman Obermaisser for the discussions on the proposed NoC and network interface.

7. REFERENCES

[1] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[2] H. Kopetz and G. Grünsteidl, "TTP - A time-triggered protocol for fault-tolerant real-time systems," in *Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing (FTCS '93)*, J.-C. Laprie, Ed. Toulouse, France: IEEE Computer Society Press, June 1993, pp. 524–533.

[3] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (TTE) design," in *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 22–33.

[4] RTCA/DO-178B, "Software considerations in airborne systems and equipment certification," December 1992.

[5] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1, 2006.

[6] Sonics, "Sonics μNetwork technical overview," http://www.sonicsinc.com/, 2002.

[7] H. P. Hofstee, "Power efficient processor architecture and the cell processor," in *HPCA*, 2005, pp. 258–262.

[8] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," *j-IBM-JRD*, vol. 49, no. 4/5, pp. 589–604, 2005.

[9] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *Micro, IEEE*, vol. 26, pp. 10–25, 2006.

[10] Altera, "Cyclone II device handbook, volume 1," Altera Corporation, 2006.

[11] M. Schoeberl, "SimpCon - a simple SoC interconnect, draft," Available at: http://www.opencores.org/, December 2005.

[12] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2002, p. 409.

[13] M. Schoeberl, "Jop: A java optimized processor for embedded real-time systems," Ph.D. dissertation, Vienna University of Technology, 2005.

[14] ——, "Java technology in an FPGA," in *Proceedings of the International Conference on Field-Programmable Logic and its Applications (FPL 2004)*, Antwerp, Belgium, August 2004.

[15] Altera, "Cyclone II FPGA family, errata sheet," Altera Corporation, March 2006.

[16] C. Hilton and B. E. Nelson, "PNoC: a flexible circuit-switched NoC for FPGA-based systems," *Computers and Digital Techniques, IEE Proceedings*, vol. 153 (3), pp. 181–188, 2006.