Argo: A Real-Time Network-on-Chip Architecture With an Efficient GALS Implementation

Evangelia Kasapaki, Martin Schoeberl, *Member, IEEE*, Rasmus Bo Sørensen, *Student Member, IEEE*, Christoph Müller, Kees Goossens, *Member, IEEE*, and Jens Sparsø, *Member, IEEE*

***Abstract*— In this paper, we present an area-efficient, globally asynchronous, locally synchronous network-on-chip (NoC) architecture for a hard real-time multiprocessor platform. The NoC implements message-passing communication between processor cores. It uses statically scheduled time-division multiplexing (TDM) to control the communication over a structure of routers, links, and network interfaces (NIs) to offer real-time guarantees. The area-efficient design is a result of two contributions: 1) asynchronous routers combined with TDM scheduling and 2) a novel NI microarchitecture. Together they result in a design in which data are transferred in a pipelined fashion, from the local memory of the sending core to the local memory of the receiving core, without any dynamic arbitration, buffering, and clock synchronization. The routers use two-phase bundled-data handshake latches based on the Mousetrap latch controller and are extended with a clock gating mechanism to reduce the energy consumption. The NIs integrate the direct memory access functionality and the TDM schedule, and use dual-ported local memories to avoid buffering, flow-control, and synchronization. To verify the design, we have implemented a 4 × 4 bitorus NoC in 65-nm CMOS technology and we present results on area, speed, and energy consumption for the router, NI, NoC, and postlayout.**

***Index Terms*— Asynchronous design, multiprocessor interconnection networks, real-time systems, time-division multiplexing (TDM).**

I. Introduction

IT IS generally agreed that an appropriate architecture for a multiprocessor system-on-chip (SoC) must support a globally asynchronous, locally synchronous (GALS) implementation and that a packet-switched network-on-chip (NoC) is a good fit to this. The concept of using packet-switched networks for intrachip communication was introduced in [1] and [2], and examples of NoC and GALS-based SoCs are found in [3]–[6].

Manuscript received August 1, 2014; revised November 20, 2014 and January 20, 2015; accepted February 1, 2015. This work was supported in part by the European Union Seventh Framework Programme through the project Time-Predictable Multi-Core Architecture for Embedded Systems under Grant 288008, and in part by the Danish Research Council for Technology and Production Sciences through Project "Hard Real-Time Embedded Multiprocessor Platform - RTEMP" under Contract 12-127600.

E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. Müller, and J. Sparsø are with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby 2800, Denmark (e-mail: evka@dtu.dk; masca@dtu.dk; rboso@dtu.dk; chmy@dtu.dk; jspa@dtu.dk).

K. Goossens is with the Faculty of Electrical Engineering, Eindhoven University of Technology, Eindhoven 5612 AZ, The Netherlands (e-mail: k.g.w.goossens@tue.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2015.2405614

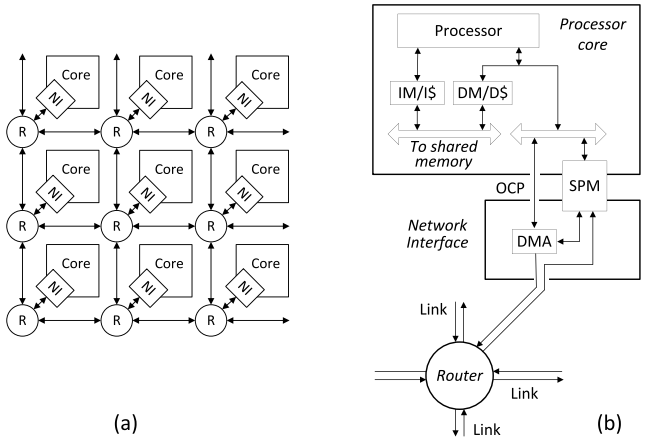


Fig. 1. Example of an Argo NoC-based multiprocessor. (a) 2-D mesh NoC topology. (b) Details of a processor core and its NI. The processor core consists of the processor itself, several caches, and an SPM.

In this paper, we present a novel GALS architecture for a general-purpose multiprocessor platform that is intended specifically for the use in hard real-time systems. The platform contains one NoC offering access to a shared memory [7] and one NoC supporting message passing, each of them optimized for its purpose of use. The focus of this paper is the message-passing NoC, Argo, whose architecture and implementation reflect the two main requirements: 1) support for hard real-time applications and 2) implementation of GALS.

Fig. 1 shows an example of an Argo multiprocessor platform using a 2-D-mesh topology. Processor cores contain some amount of local memory, e.g., caches and explicitly managed scratchpad memories (SPMs). The SPM serves as the source and target for the message passing between cores. As shown in Fig. 1(b), the processing cores are connected to the routers through a network interface (NI). A direct memory access (DMA) controller in the NI initiates message transfers from the SPM of one core to the SPM of another, generating a (NoC specific) packet stream toward the network of routers. A standard read–write transaction interface (Open Core Protocol (OCP) [8]) toward the cores is used to set up the DMA controllers.

In hard real-time systems, it is necessary to guarantee the worst case execution time (WCET) of a task executing on a processor as well as the execution time of an application consisting of a set of communicating tasks that execute on a set of processors. To enable this, all the components, including

the NoC, must be analyzable and time-predictable. Most published NoCs support only the best effort traffic [9]–[11], and some NoCs offer multiple priority levels for different qualities of service [12], [13], but none of these are adequate for hard real-time systems. To avoid interference of traffic and to obtain full time predictability, the NoC must provide some form of end-to-end (virtual) circuits, for which latency and throughput guarantees can be given. Examples of NoCs in this category—that will be discussed in more detail in Section II—are: 1) Æthereal and aelite [14] and 2) MANGO [15].

The message-passing NoC in our platform uses statically scheduled time-division multiplexing (TDM) to implement end-to-end circuits and support time predictability. The primary reason for choosing TDM over alternative approaches for time predictability is simplicity: 1) simplicity of calculating communication latency and 2) simplicity of the hardware implementation of the routers—pipelined crossbars (Xbars) without any circuitry for arbitration or buffering. Intuitively this simplicity may result in disadvantages as well, for example: 1) increased complexity of the NIs because they must now include schedule tables and data buffers and 2) increased communication latency following from the fact that TDM is not work-conserving. We have been able to combat the first drawback by a novel microarchitecture of the NIs. The second drawback is not an issue in the context of hard real-time systems where focus is on WCET.

The requirement for a GALS organization and what follows in terms of synchronization and clock domain crossings greatly affect the design of the NoC. The typical mindset when designing a GALS-based system is the one of composing modules of synchronous circuitry using glue components that implement synchronization or clock domain crossing. With this mindset, the glue components become overhead. Our architecture uses components that implement the required functionality to also implement the mesochronous synchronization and true clock domain crossing.

TDM requires a common notion of time across the platform to ensure that all NIs and routers in the NoC operate in sync with the global TDM schedule. This leads to mesochronous implementations using clocked routers. Mesochronous means operating on the same frequency with bounded skew. The typical way of supporting mesochronous synchronization is using mesochronous synchronizers. These are first-input first-output (FIFO) structures that are needed in all links of the NoC. In contrast, our design uses pipelined asynchronous routers. Any asynchronous pipeline also offers some inherent FIFO capability [16]. By exploiting this combined pipeline and FIFO behavior, we avoid the per-link synchronizers needed in clocked mesochronous NoCs.

In our design, the network of asynchronous routers is embedded in a shell of mesochronous NIs that drive the asynchronous routers below their maximum speed. In every clock cycle, the NIs transmit and receive a phit (a data token that can be valid or void) to and from the network of routers. In combination with the local join-fork (JF) synchronization that is performed inside the asynchronous routers, this establishes the synchronicity necessary to implement TDM as well as the elasticity that is necessary to support a GALS implementation using mesochronous NIs. At the same time, data are transferred in a simple pipelined fashion from the local memory of the sending processor core to the local memory of the receiving processor core, without passing through any arbitration, buffering, or clock domain synchronization circuitry.

Elements of the design have been published previously in [17]–[20]. The contributions of this paper are as follows.

1) It gives for the first time a detailed presentation of all aspects of the design. This is important for a full understanding of the qualities and characteristics of the design—the originality of the design rests in the combination of the TDM principle, the NI microarchitecture, and the asynchronous router, and the way in which synchronization is handled across mesochronous and fully asynchronous clock domain boundaries.
2) The design of a new and efficient two-phase bundled-data router. The router uses the Mousetrap latch controller [21] and has been extended with a mechanism that resembles clock gating in order to reduce the energy consumption when router ports are idle.
3) Synthesis and layout of a complete 4×4 bitorus NoC in a 65-nm CMOS technology. From this, we derive post place-and-route figures for area, speed, and energy.

This paper is organized as follows. In Section II, we present the background and related work in the field of real-time NoCs, and specifically in the TDM scheme, GALS, and traditional architectures. In Section III, we present the overall architecture and the key ideas of our Argo NoC. In Section IV, we present the new asynchronous router and NI design. In Section V, we present the schedule generation for Argo and how to compute the message latency. In Section VI, we give the implementation details and results showing the efficiency of the architecture. Finally, the conclusion is drawn in Section VII.

II. Background and Related Work

This section reviews related work in the area of NoCs for real-time systems, GALS timing organization, and NI design.

A. Real-Time NoCs

The fact that a NoC is a shared communication medium comprising multiple, independently arbitrated resources (routers and links) may severely complicate timing analysis. The seemingly simple question "what is the latency that the NoC adds to a read or write transaction toward a memory in a remote core?" can be very difficult to answer. To give guarantees on bandwidth and/or latency for individual transactions, some form of end-to-end connection, physical or virtual, is needed. We decided to implement virtual circuits using statically scheduled TDM.

Examples of NoCs offering circuit switching with physical connections are the NoC used in the 4S-platform [22] that offers initialization-time field-programmable gate array (FPGA)-style configurable connections and

SoCBUS [23] that implements a dial-up mechanism. In both cases, connections, when established, own resources exclusively, and thus they can easily provide real-time guarantees. The downside is a (potential) low utilization of resources. Furthermore, in SoCBUS, a dial-up attempt is not guaranteed to succeed. Our TDM approach makes more efficient use of resources, sharing them among many virtual circuits in a time-multiplexed fashion. Moreover, the predefined static schedule guarantees the connections defined in it, making the NoC suitable for hard real-time systems.

The use of virtual circuits allows sharing of resources. Virtual channels can be implemented in two fundamentally different ways. One approach is to use TDM, where the resources (routers and links) are used in a time-multiplexed fashion according to a static schedule. Examples are Æthereal [24], aelite [25], Nostrum [26], and TTNoC [27], [28]. Since the routers operate on a predefined schedule, there is no arbitration, buffering, and flow control, resulting in a very simple router implementation. The routers in our approach also avoid buffering, arbitration, and flow control.

The other approach is to use nonblocking routers with rate control [29]. Examples of this approach are the asynchronous MANGO NoC [15] and the Kalray NoC [30]. In these NoCs, several connections may share a link, but each connection has a private (virtual-channel) buffer in every router along the connection. The Kalray NoC uses static paths for virtual circuits, constraining them when a throughput limit is reached, enforcing a throughput rate over a time interval. The downside of both these designs is the high hardware complexity caused by the virtual channel buffers and a larger Xbar. The large Xbar comes from the use of virtual channel buffers and from the arbitration and flow control needed in every output port. It is interesting to observe that a typical MANGO router is 10 times larger than an aelite router [14], [31]. The Argo router avoids all buffering and hardware complexity from arbitration resulting in a simpler and more efficient router implementation.

In a different perspective, other approaches use network calculus [32], [33] to estimate arrival curves for traffic flows in order to derive the performance guarantees. Traffic flows are used in schedulability analysis [34], [35] to solve the contention problem and to evaluate the throughput of the NoC. Scheduling techniques attempt to give guarantees by enforcing throughput limits or by assigning priorities to traffic flows. The scheduling techniques can be simple (round robin) or more elaborate (weighted or priority) and they may rely on network calculus to derive more precise priorities based on traffic loads [36]. In our opinion, statically scheduled TDM is a simpler technique, is easier to analyze and implement, and offers hard real-time guarantees.

A number of recent NoC designs support both packet switching and circuit switching [37]–[39]. The main motivation behind these designs is that circuit switching is used as a means of reducing both power consumption and end-to-end latency; there is no focus on supporting real-time traffic. For example, the hybrid NoC developed in [38] requires a channel reservation phase before using the circuit-switched NoC resources. Like for SoCBUS discussed above, there are no guarantees that a circuit can be established.

B. Time-Division Multiplexing

TDM is an arbitration scheme to share resources over time. TDM partitions time into fixed-duration units called time slots. A TDM schedule assigns resources to channels at the granularity of time slots. In this way, different (end-to-end) virtual circuits with different latency and bandwidth guarantees can be provided on top of a shared hardware resource. Nostrum [26] and Æthereal [14] families of NoCs use TDM. Like most other NoCs, they use packet switching and wormhole routing—either source routing, with routing tables in the NIs, or distributed routing, where each router contains a local routing table.

In every time slot, the TDM router forwards one or more packets from its input ports to its output ports according to a static schedule that is predetermined. Thus, it is never the case that two packets compete for the same output port of a router at the same time. Since there is no congestion, there is no need for dynamic arbitration, flow control, and buffering in the routers. This leads to a very simple router implementation.

The TDM scheme requires a global notion of time. Since the entire NoC is operating on a static schedule, all components of the network, i.e., routers, links, and NIs, are tightly connected. In a globally synchronous and mesochronous NoC, the global clock enforces the synchronization. In an asynchronous NoC, the global synchronicity is maintained by preserving the TDM schedule at the end points of the network while allowing the internal synchronicity to be relaxed (local synchronization). The idea of local synchronization was proposed in [40], but the implementation of an asynchronous TDM NoC has not been explored.

The TDM schedule for a TDM NoC is calculated such that there is no contention and such that bandwidth guarantees are given for each connection. Deadlock cannot occur, since packets never wait. TDM scheduling can be applied at the level of packets, flits (flow-control-digits), or phits. A flit, i.e., the smallest unit that is individually routed may consist of several phits, i.e., the basic unit of the physical layer. In Argo, the terms packet and flit are synonymous, and a packet consists of three phits.

In the CompSoC [25] platform, the flit size is three phits, and scheduling is performed at the flit level. Therefore, the pipeline depth of the TDM router matches the size of a flit, such that a router can store a whole flit. In addition, the requirement means that pipeline registers can only be added or deleted in groups of three.

In contrast to CompSoC, we calculate the schedule at the phit level. This removes the restriction of matching router and link pipeline depth to the flit size, offering more flexibility to optimize the hardware for size or for throughput. Packets are routed individually and phits of the same packet travel in immediate succession, but a packet can be scheduled with a delay of zero, one, or two clock cycles relative to the time slot.

This scheduler was presented in [20] and is further explained in Section V-A.

C. GALS Timing Organization

Distributing a clock signal across a chip and achieving timing closure throughout the whole chip is becoming increasingly difficult due to parameter uncertainty and variability [41]. GALS-style architectures reduce these problems by dividing the design into several independent clock domains and by implementing some form of asynchronous communication among these. A NoC-based multiprocessor platform naturally supports a timing organization where the processor cores and the NIs constitute separate clock domains.

A mesochronous timing organization represents the first step away from a globally synchronous design. There are numerous ways to achieve mesochronous functionality [42]: 1) DSPIN [43] and 2) aelite [14], [25] use bisynchronous FIFOs [44], [45] on the links between routers. The routers are clocked with the same clock, but a bounded phase difference is allowed. The bisynchronous FIFOs are needed in every link and this has a significant penalty in terms of increased area and power consumption. The addition of the FIFOs in the aelite NoC is reported to more than double the area of a router [25]. In contrast to aelite, our design contains FIFOs only between the NI and the router.

A more straightforward solution would be an entirely asynchronous implementation of the routers and links. CHAIN [46], MANGO [15], QNoC [12], and ANoC [10] are some of the asynchronous NoCs that explore this approach. Among the asynchronous NoCs mentioned, only MANGO offers hard real-time guarantees. However, as stated in Section I, its hardware cost is considerable compared with the simple TDM-based aelite router.

The solutions proposed above refer to the internal communication of the packet-switched NoC, i.e., between routers. Synchronization and timing elasticity are also needed between the NIs and the packet-switched NoC. For this purpose, bisynchronous FIFOs can be used between NIs and routers to provide for synchronization of fully independent clock domains or for elasticity between mesochronous components.

For these reasons, we decided to explore how to implement TDM efficiently in a GALS context using an asynchronous implementation.

D. Network Interfaces and End-to-End Communication

The NIs in a NoC play a key role in implementing end-to-end data transfer between two communicating processor cores. Fig. 2 shows a typical multiprocessor architecture. The figure also shows the points where clock domain crossing and synchronization are needed in order to provide a GALS organization using mesochronous routers and NIs. In this section, we briefly address the overall architecture, the functionality of the NI, and the end-to-end data transfer.

As shown in Fig. 2, each processor core contains some private memory: 1) caches and explicitly managed SPMs and 2) a DMA controller. The use of SPMs and DMA controllers has been adopted in a number of multiprocessors for embedded systems [25], [47]. The use of SPMs makes data transfers explicitly visible to the application programmer and avoids the implicit cost (area, power, and latency) of using cache memories. The DMA controllers are intended to implement background DMA-driven block transfers from the local memory (SPM) of a processor core and into the local memory (SPM) of a remote processor core. In this way, program execution in the processor core and data transfer across the NoC are completely isolated from each other, offloading the processors and simplifying WCET analysis.

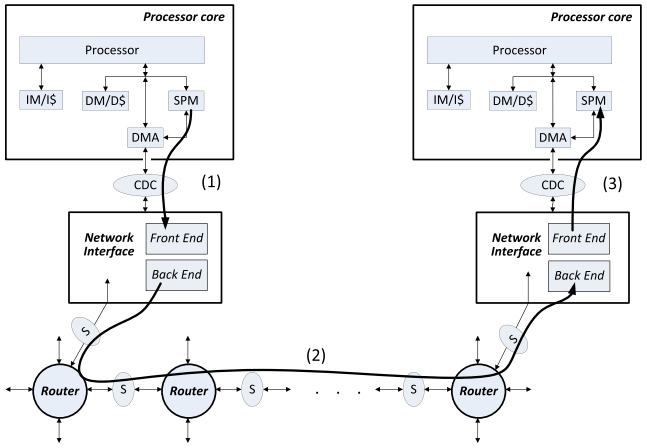


Fig. 2. Block diagram of a traditional GALS implementation of multiprocessor architecture illustrating the end-to-end message passing.

NIs are typically designed with a mindset focusing on layering, encapsulation, and interfaces, and a general treatment of this topic as well as descriptions of some specific NI designs may be found in [48]. A NI is generally divided into a front end and a back end, as shown in Fig. 2. Toward the attached core, the front end provides one or more ports implementing standard bus-style read–write transaction interfaces. The NI front-end transforms these transactions into some form of connection-oriented streaming of packets. The NI back end deals with lower-level issues that are specific for the specific NoC that is used, e.g., packetization, routing, buffering, and flow control—often implemented using some form of credit-based scheme. Read or write transactions involving larger amounts of data may need to be transmitted as a sequence of smaller request and/or response packets. The back end also handles this splitting and reassembly.

Let us now consider in more detail a core-to-core message transfer. Three autonomous processes perform the message transfer, as shown in Fig. 2.

1) A DMA controller in the source node transfers the message data into a buffer in the NI.
2) The NI in the source node packetizes the data and sends them. The packets traverse the NoC and are received by the destination NI.
3) A DMA controller in the destination node transfers the data from the NI and into the SPM.

In a GALS-implementation, where processor cores and NIs may operate at different and independent clock rates, the three processes are completely asynchronous and for this

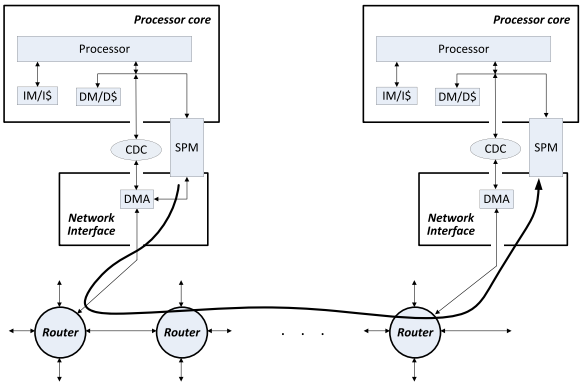


Fig. 3. Block diagram of the new architecture illustrating the direct end-to-end message passing.

reason, flow control is needed. Toward the DMA controllers, the NIs must signal buffer status (full/empty) and between the NIs, some kind of credit-based flow control is typically used. In addition to this flow control, the end-to-end path involves two clock domain crossings (between the processor cores and their NIs) and FIFO synchronizers between the mesochronous NIs and the routers. In Fig. 2, these are marked CDC and S, respectively.

As hinted by the above discussion, the hardware implementation of NIs can be quite large due to the buffering and flow control. In many cases, the size of a NI is comparable with the size of a processor [31]. As described in Section III Argo uses a novel NI microarchitecture that transfers data from the SPM in the source node across the NoC and into the destination SPM without any buffering and flow control.

III. Overall Architecture

This section presents the overall architecture and the key underlying ideas, i.e., the integration of DMA controllers in the NIs, the use of asynchronous routers, and the timing organization of the entire NoC.

A. Network Interfaces With TDM-Driven DMA Controllers

The very essence of TDM is that it avoids buffering, flow-control, and dynamic arbitration. This implies that it should be possible to transfer data all the way from the SPM of one processor core into the SPM of another processor core without any buffering, flow control, and dynamic arbitration. However, as explained in Section II-D and shown in Fig. 2, this is not the case. The problem is that the DMA controllers operate autonomously and independently of the TDM-schedule.

The first key feature of Argo is that we have moved the DMA controllers from the processor nodes into the NIs, where they are integrated with the TDM schedule, as shown in Fig. 3. In a TDM-based NoC, all communication channels are assigned some time slots in the TDM schedule, such that each channel has some bandwidth guarantees. DMA transfers have to be interleaved correspondingly, with one DMA controller per outgoing channel. On the other hand, the NI can only inject one packet at a time into the NoC,

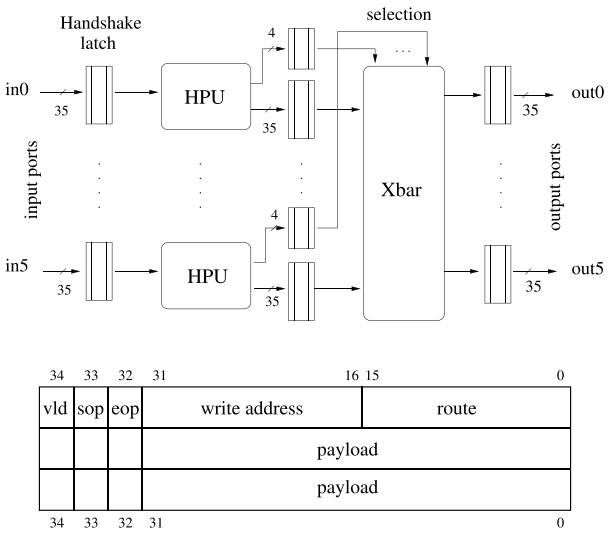


Fig. 4. Block diagram showing the microarchitecture of the Argo asynchronous router and the packet format. For clarity, the request and acknowledge handshake signals are not shown.

and consequently, only one DMA controller can be active at a time. This allows a single table-based implementation of all the DMA controllers, where each entry of the table refers to one DMA controller, i.e., one outgoing channel.

In addition, the dual-ported SPMs can be used for clock domain crossing. In this way, explicit clock domain crossings are needed only to program the DMA controllers; the actual transfer of message data does not require any synchronization. The combination of interleaved DMAs and dual-ported SPMs creates a direct path from the SPM to the network of routers, avoiding the need for flow control and buffering in the NIs as well as latency for clock domain crossing for the message data. Flow control between NIs can be omitted, since the receiving SPMs always accept incoming transactions, i.e., offer a consumption guarantee. As shown in Fig. 3, data are transferred from the source SPM across the NoC and into the destination SPM without any buffering, flow control, or clock domain crossings. The result is a very small and efficient NI implementation.

Since NIs have the responsibility of enforcing the TDM schedule, we have chosen a clocked mesochronous implementation. In contrast to an asynchronous implementation, a clocked one simplifies the synthesis process and the task of evaluating WCETs.

B. Asynchronous Argo Router

The second key idea is to use asynchronous routers instead of clocked mesochronous routers. A clocked mesochronous router, as used in [25], consists of a clocked router extended with bisynchronous FIFOs on all input ports. The use of an asynchronous router avoids the need for FIFOs—the necessary timing elasticity is provided by the router itself. The result is that the area and power consumption is reduced significantly, as we will see later.

The Argo TDM router uses source routing, and in combination with the TDM scheme, which requires no flow control or buffering, the routers become very simple and efficient.

The Argo TDM router, shown in Fig. 4, is an asynchronous implementation using handshake latches instead of clocked registers. The router is a three-stage pipeline: 1) link traversal; 2) header parsing unit (HPU); and 3) traversal of the Xbar switch. A router typically has five ports, and this allows the construction of mesh-type topologies.

A packet consists of three phits, i.e., one header phit and two payload phits, as shown in Fig. 4. A phit is the basic unit that is held in a pipeline stage. Each phit is a 32-bit data word along with three control bits, indicating whether the phit is valid or not (vld), the start of packet, and the end of packet, respectively. The header phit contains the destination write address and the route that the packet follows. The route is encoded with 2 bits for every hop the packet traverses.

Each input port has an HPU that extracts 2 bits from the route to be decoded to a 4 bit one-hot encoding, selecting the destination output port of the router. At the same time, the HPU shifts the route field of the header phit two positions to align the header for the next router along the path. The path through the Xbar is locked until the last phit of the packet has propagated through the Xbar.

The overall timing behavior of a structure of these asynchronous routers is fundamentally different from a clocked synchronous (or mesochronous) design. Lacking a clock to explicitly define the time slots of the TDM scheduling, the asynchronous router design enforces this synchronization in an implicit and distributed manner using a strongly indicating [49] implementation of the Xbar switch in the router. In every handshake cycle, the Xbar consumes a phit on each input channel (join), and in every handshake cycle, it outputs a phit on all output channels (fork). With this JF mechanism of the Xbar, it is possible to mimic the ticking of a global clock, but this is done in a self-timed distributed manner that avoids enforcing synchronization to a clock signal inside every router.

For a given TDM schedule, there will be slots where no traffic is scheduled on some links and router ports. At the same time, the synchronization in the Xbar is achieved through handshaking on all input ports in every cycle. For this reason, we use two types of phits: 1) valid phits and 2) void phits. Void phits are transmitted on ports and links where no traffic is scheduled in a given cycle, and, as explained in Section IV-A, we use clock gating in individual router ports to save power when void phits are propagated.

C. Timing Organization

Argo supports a GALS organization with independently clocked processor cores and mesochronous NIs, and the NI clock is the time base for the TDM scheduling. The asynchronous routers provide the timing elasticity necessary to cover for skew among the mesochronous NIs. As will be clear from the following text, the interfaces between the asynchronous routers and the NIs do not require synchronization. The only clock domain crossing where synchronization is required is between the processor and the DMA controllers in the NIs. This interface is only used to set up the DMA controllers; the actual transfer of data from a source SPM, across the NoC, and into a destination SPM happens without any synchronization. It is this property in combination with the use of statically scheduled TDM that renders all buffering and flow control unnecessary and thereby results in a very simple, small, and efficient design.

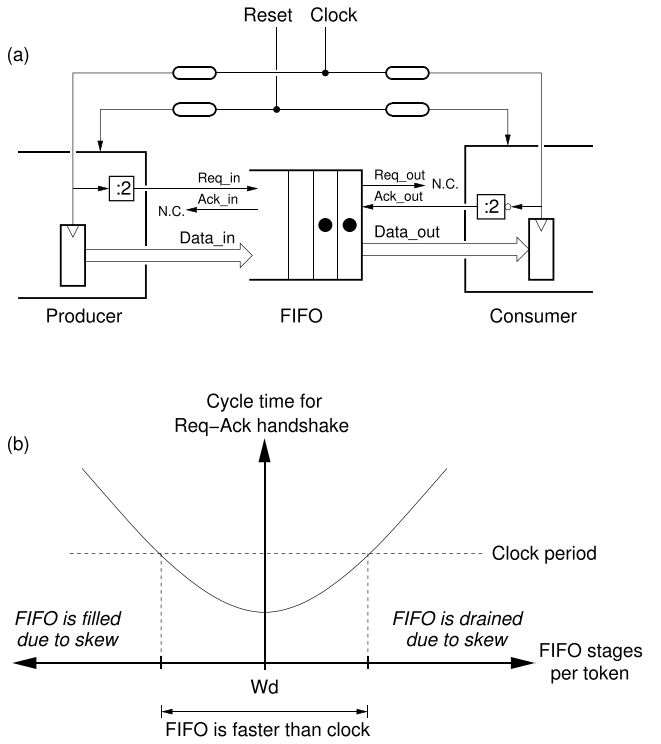


Fig. 5. (a) Asynchronous FIFO connecting a mesochronous producer–consumer pair. (b) Influence of skew (positive or negative) on the speed of the FIFO.

The skew between the NIs includes not only clock skew but also the skew related to the deassertion of reset. In combination, these two factors result in skew among the TDM slot counters in the different NIs. This skew can exceed one cycle. As explained below, we use the inherent ripple FIFO nature of the network of asynchronous routers and links to accommodate such skew.

Fig. 5(a) shows a mesochronous producer–consumer pair connected by an asynchronous ripple FIFO. Being mesochronous, the producer and the consumer operate at the same rate. If the FIFO is initialized to a state where it is neither completely full nor completely empty, it can accommodate some amount of skew between the producer and the consumer. Assuming an upper bound on the skew, it is possible to determine a FIFO depth that ensures that the FIFO never runs completely full or completely empty. This again means that the FIFO never stalls completely; it will be able to input and output data at some minimum rate. Under the assumption that the frequency of the clock signal driving the mesochronous producer and consumer is lower than this minimum data rate of the FIFO, the handshake signals Ack_in and Req_out can be ignored, as shown in Fig. 5(a). The corresponding timing assumptions $\text{Req\_in}^{(i)} \prec \text{Ack\_in}^{(i+1)}$ and $\text{Req\_out}^{(i)} \prec \text{Ack\_out}^{(i)}$ (where i enumerates the sequence of handshake cycles) can always be satisfied by setting the clock frequency to a sufficiently low value.

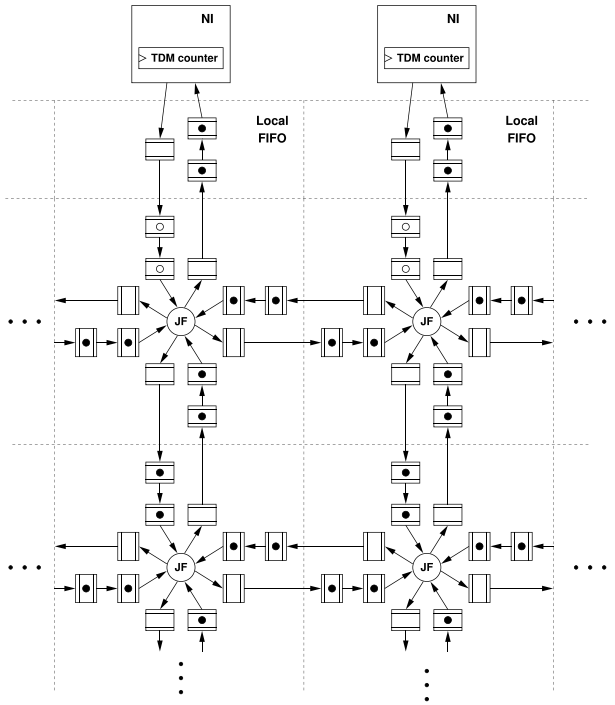


Fig. 6. FIFO-based model of the structure of the Argo NoC.

The speed of an asynchronous FIFO that is being used in a context where the producer and consumer operate at the same rate depends on the number of FIFO-stages per data item. The speed of the FIFO is at a maximum when the number of stages per data item matches the dynamic wavelength, W_d [49], as shown in Fig. 5(b). If, due to skew, the FIFO becomes fuller or emptier, it will exhibit a slowdown. As long as the slowdown does not cause the handshake cycle time of the FIFO to become larger than the period of the clock, the arrangement in Fig. 5(a) is safe, meaning that the FIFO delivers data to the consumer well before the next clock tick and that the FIFO is ready to receive data from the producer well before the data are actually delivered at the next clock tick.

These are the key ideas underlying the architecture, and the design does not require any form of synchronization in the interfaces between the producer and the FIFO and between the FIFO and the consumer. The exact amount of skew that a particular FIFO implementation can tolerate depends on the number of stages in the FIFO, the initial number of data items in the FIFO, the frequency of the clock, and the technology used for the implementation (worst case corner and operating conditions). A slower clock gives more headroom for skew-induced slowdown of the FIFO. Using realistic gate and link delays, we found that for our design, $W_d \approx 1.5$. For this reason, we decided for a three-stage router initialized to hold two data items. As packets are three phits long, it follows that packets traverse routers in an unaligned fashion and for this reason, we use phit-level scheduling instead of packet-level scheduling. The scheduler is further discussed in Section V-A.

The timing behavior of a complete Argo NoC can be understood as a mesh of FIFOs connected by JF synchronization points, as shown in Fig. 6. The JF nodes represent the strongly

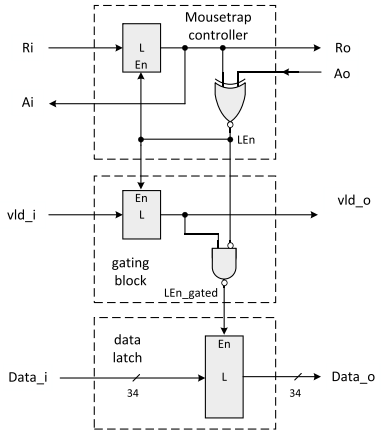


Fig. 7. Argo handshake latch consisting of three blocks: Mousetrap controller, gating block, and normal data enable latch.

indicating Xbars in the routers. As shown in Fig. 4, a router has two pipeline stages before the Xbar and one pipeline stage after the Xbar. This structure can be recognized in Fig. 6.

To balance the design and increase skew tolerance, we have added additional pipeline stages on the local port that connects an NI to its router. Our preferred design has one FIFO stage in the channel from an NI to a router and two FIFO stages in the channel from a router to an NI. This provides ample time elasticity and it makes the mesh-structure symmetric—all FIFO segments contain three pipeline stages initialized to hold two tokens—a fact that simplifies the timing analysis.

We conducted an analysis to evaluate the elasticity of Argo, i.e., the phase difference that can be tolerated at the end points of the NoC and the mesochronous NIs. The details of the analysis can be found in [19]. The results show that a 2×2 instance of Argo can tolerate up to three clock cycles of skew, depending on the operating frequency. This implies that, at the layer of NIs, Argo can tolerate a shift in the TDM schedule of more than one clock cycle without compromising the correct behavior of the NoC.

IV. Design

In this section, we describe the details of the router design, the NI design, and how to initialize the Argo NoC.

A. Router

As shown in Fig. 4, the Argo TDM router is a three-stage pipeline. The pipeline stages use two-phase bundled-data handshake-latches [49] that are implemented using conventional enable latches and a Mousetrap controller [21], as shown in Fig. 7. The Mousetrap controller was chosen after studying a range of alternative designs [18], as it is very efficient, small, fast, and easy to implement. As shown in Fig. 7, the Mousetrap controller consists of a conventional latch and an XNOR gate. No special asynchronous cells, such as C-elements, are used. The entire router contains only two C-elements. These are in the Xbar where they implement the join and fork functionality. They are not on the critical path and are implemented using conventional logic gates.

We have extended the Mousetrap controller with a clock-gating scheme in order to reduce power consumption.

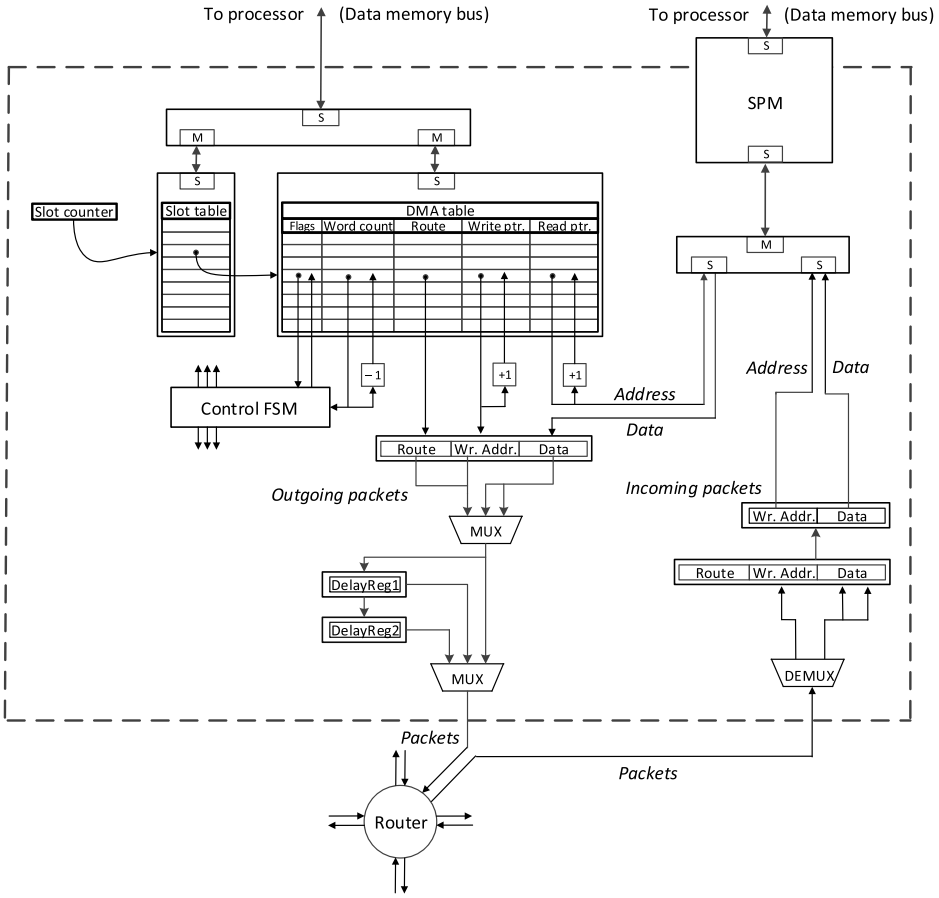


Fig. 8. Block diagram showing the microarchitecture of the Argo NI design.

The phits forwarded with the asynchronous handshaking may represent either valid phits or void phits, indicated by the valid bit (vld) in the packet format. In the case of a void-phit, the latch-enable signal that controls all the data latches is disabled. This functionality is implemented by the gating block shown in Fig. 7. This block uses the latch that is already part of the handshake latch to hold the vld bit of the phit and an additional logic gate to provide the gating. The cost of the additional gate is minimal, as a buffer is anyway needed to drive the enable signals to all the data latches.

Having a three-stage bundled-data design, the Argo router uses three delay elements: one matching the link delay, one matching the HPU combinational logic, and one matching the Xbar logic. To cover delay fluctuations from process variations, we added a safety margin of 20%. This value is typical and is discussed in [50] for a 65-nm CMOS process operating at 1 V. If a chip is to be fabricated, the margin has to be verified. The delays were implemented as a series of buffers and inverters of the same technology. A repeated process of setting the delay elements, synthesizing and generating a layout, analyzing the timing, and adjusting the delay elements leads to timing closure of the design.

B. Network Interface

Fig. 8 shows the design of the NI. The key elements of the microarchitecture are the slot counter, slot table, DMA table, and SPM. The TDM schedule is stored in the slot table, while the route information for every channel is stored in the DMA table. The slot counter is reset and it is incremented in all NIs using the same (mesochronous) clock. The slot counter defines the current slot in the TDM period. The slot counter indexes a slot table, where each entry consists of a valid bit and an index into the DMA table. The valid bit indicates whether or not this time slot is assigned to an outgoing channel. If the valid bit is true, the entry also holds a pointer to the relevant entry in the DMA table. An entry in the DMA table holds all the registers that are found in a normal DMA controller (control bits, a read pointer, a write pointer, and a word count). In addition, source routing requires the route for a remote transaction, which is also included in the DMA entry. When a DMA is active, the data are read from the SPM starting at the read pointer address. At the destination, the received data are written straight to the SPM at the write pointer address. The read and write of the SPM are done on the NI clock, as the SPM is dual ported.

Since Argo uses phit level scheduling, it follows that packets and TDM time slots are not aligned. The NIs support this misalignment by a controlled phase shift (relative to the three cycle TDM slot) of packets leaving the NI. This phase shift can be zero, one, or two clock cycles, and it is encoded by 2 bits in the slot table entries (not shown in Fig. 8). In the incoming path, a buffer stores the write address and the two words of payload data of arriving packets in a buffer

until the clock cycle allocated for writing the double word to the SPM.

The NIs are mesochronously clocked components, and the clock signal in a NI is used to drive the handshaking on the interfaces toward the asynchronous router, as explained in Section III-C. The connection from the NI toward the router is similar to the interface between the producer and the FIFO in Fig. 5 and the connection from the router toward the NI is similar to the interface between the FIFO and the consumer in Fig. 5. Delay elements are added to the request signal to ensure the bundling constraint.

The NI has two interfaces toward the processor side. Both follow the OCP specification [8] and both support single word read and write transactions. One interface is used to configure the NI and to write commands to and read status from the DMA controllers. If the processor core and the NI are in different clock domains, a clock domain crossing is required on this interface. The second interface is for the data payload and is toward the SPM. Using standard OCP to interface the processor with the NI provides decoupling form the processor design and simplifies integration.

C. Initialization of Argo

Initialization of Argo involves resetting of the asynchronous routers and links and the synchronous NIs, and configuring the NIs with the desired TDM schedule. The resetting is done using a global reset signal. In the routers, this initialize void phits in the pipeline stages marked with solid tokens in Fig. 6.

Lacking phits in segments connecting the NIs to the J/F nodes in Fig. 6, the network of routers and links is in a deadlock state; the Xbars (the J/F nodes) in the routers are waiting for phits from the local router ports. This allows the reset signal to be deasserted with any amount of skew. When reset is deasserted, the free-running TDM counters in the NIs start. After a small and fixed number of cycles, the NIs inject two void phits (illustrated by unfilled tokens in Fig. 6) into the network before entering the steady state where they input and consume a phit in every cycle.

At this point, each processor loads the schedule into its NI. This involves writing the slot table entries and the route column in the DMA table. The schedule is obtained from the shared memory that is accessed using a memory-tree network (not discussed in this paper). A barrier-synchronization implemented using the shared memory is used to determine when all NIs have been configured. The NoC is now ready for use.

To send a message across a virtual channel, the processor sets up the corresponding DMA controller. This involves writing the read address for the source SPM, the write address for the destination SPM, the word count, and finally setting a start control bit. The processor can check the status of the transfer by reading the word counter.

V. Scheduling and Message Latency

A. Scheduling

We developed a tool to generate the static TDM schedule for the Argo NoC [20]. For each application that is executed on a multiprocessor system, the workload is distributed into tasks, which are then mapped to a set of processors for execution. Depending on the mapping of tasks to processors, certain pairs of processors have particular bandwidth requirements. The bandwidth requirements between all pairs of processors give the overall communication requirement that needs to be scheduled.

In TDM, the bandwidth and the wait time for at time slot are inversely proportional. If the message is larger than what can be sent in one TDM period, then the message latency depends mainly on the allocated bandwidth. Increasing the number of time slots in a TDM period for a specific communication channel makes the time between consecutive time slots shorter, thus decreasing the wait time.

As mentioned in Section III-C, the router design and the packet length require scheduling to be done at the level of phits. The main advantage of such phit scheduling is that it allows more freedom in pipelining the design. The design can be pipelined holding single additional tokens, rather than multiples of three additional tokens, keeping the hardware cost low.

The scheduler produces a communication schedule at the level of phits, with the requirements that communication channels are only routed on the shortest paths, links are only allocated to one communication channel in any time slot, and all communication channels in the input specification are routed. These three requirements ensure in-order arrival of packets, that no packets collide, and that no channels are starved from access to the network, respectively. The schedule contains the route that each packet shall follow and the time slot in which each packet shall leave the NI.

The scheduler always creates a schedule that implements the communication channel specified in the bandwidth graph, but in some cases, the period of the created schedule is so long that the network frequency it requires to run is higher than the maximum frequency of the network. This means that the application should be restructured or a larger platform should be used.

As an example, we have generated an all-to-all schedule with equal bandwidth for the example platform of 4×4 nodes. For this configuration, the TDM period is 23 time slots. This schedule is used in the evaluation of the complete 4×4 Argo NoC.

B. Latency Analysis

When programming a real-time application using Argo, the worst case latency L_{msg} of sending a message from one processor to another is the primary concern. We show below how to calculate this latency using a simplified version of the equations from [51] that do not assume any particular distribution of slots in a schedule, and we derive numbers for the presented platform with an all-to-all schedule with equal bandwidth between all cores.

L_{msg} is the latency, in NI clock cycles from the time the source processor has set up a DMA to the time the destination processor has received the full message. L_{msg} consists of two parts: 1) the time to send all packets into the network and 2) the time for the last packet to travel from the source to the destination. The time to send all packets into the network

TABLE I
WORST CASE MESSAGE LATENCY IN A 4 × 4 BITORUS NETWORK FOR DIFFERENT MESSAGE SIZES

S_{msg}(byte)	8	16	32	64	128	256	512	1024
L_{msg}(cyc)	79	148	286	562	1114	2218	4426	8842

is the waiting time for a time slot plus the number of TDM periods needed to send the whole message; the formula is

$$L_{\text{msg}} = \left(T_{\text{wait}} + \left\lceil \frac{S_{\text{msg}} - S_{\text{pkt}}}{S_{\text{chan}}} \right\rceil \cdot P_{\text{sched}}\right) \cdot C_{\text{slot}} + H \cdot D \quad (1)$$

where T_{wait} is the worst case waiting time until the first packet has been injected to the network toward the destination, S_{msg} is the size of the transmitted message, S_{pkt} is the number of payload bytes in a packet, S_{chan} is the number of payload bytes that can be sent in one TDM period from the source processor to the destination processor, P_{sched} is the length of the TDM schedule, C_{slot} is the number of clock cycles in a TDM slot, H is the number of hops from the source to the destination processor, and D is the number of phits that can be stored in one hop.

For the presented 4 × 4 core platform, S_{pkt} is 8 bytes, C_{slot} is 3 clock cycles, H is maximally 5 hops, and D is 2 phits. With an all-to-all schedule for this platform, T_{wait} is the full TDM period of 23 time slots, S_{chan} is 8 bytes, and P_{sched} is 23 time slots. The remaining variable is the size of a message. We see that the worst cast message latency L_{msg} is simple to calculate and does not depend on the behavior of other processors. Table I shows L_{msg} for different message sizes on the presented platform under the above assumptions. With the values from Table I, we can see that it takes ∼9 cycles per byte to transmit. This latency can be reduced by generating an application specific schedule.

VI. RESULTS AND DISCUSSION

The Argo NoC is operational and has been used in the T-CREST project by two industry partners [52]. Hard real-time applications from two industry domains (a railway application and several applications from the avionics domain) have been ported to use the Argo NoC message passing instead of shared memory communication. The improvement on the WCET has been reported in [52] and is outside the scope of this paper. In this section, we focus on the evaluation of the Argo NoC.

To evaluate and characterize Argo, we have implemented its individual components, i.e., the asynchronous router and NI, as well as a complete 4 × 4 instance of the Argo NoC, including the NIs, the asynchronous routers, and the additional FIFO stages, as described in Sections III and IV. The designs were described in VHDL, synthesized in an STMicroelectronics 65-nm CMOS technology library, and laid out in a floor plan. The NI was additionally implemented in FPGA technology using an ALTERA DE2-70 FPGA board with a Cyclone II EP3C70 chip. We used Synopsys Design Compiler for synthesis, ModelSim for simulation, SoC Encounter for layout, Synopsys Prime Time for power analysis, and Altera Quartus for the FPGA implementation.

TABLE II
RESULTS FOR THE ROUTER IMPLEMENTATIONS

	Post-synthesis			Post-layout	
	Area (μm^2)	Freq. (MHz)	Energy (pJ/cyc)	Freq. (MHz)	Energy (pJ/cyc)
Mesochronous	24239	1111	20.00	724	26.27
sync. router	8026				
FIFOs [44]	16213				
2ph-bd	7594	998		711	
link util. 70 %			7.92		8.03
2ph-bd-g	7536	900		593	
link util. 0 %			1.64		2.11
link util. 70 %			6.51		7.23
link util. 100 %			8.77		9.66
Argo	7715	1126		746	
link util. 0 %			1.28		2.17
link util. 70 %			6.45		6.54
link util. 100 %			8.24		8.39

A. Router

The Argo router, being an asynchronous design, requires special treatment through the design flow. We used conventional design tools and synthesized with an aim for speed, but we applied special constraints for synthesis and optimization of the design. We applied local timing constraints that aim to optimize the combinational logic of each pipeline stage and to place delay elements in the request lines to match the combinational logic of every stage as required by the bundled-data protocol.

Before deciding for a particular router design, we explored a number of alternatives, including a clocked mesochronous design [18]. Table II shows postsynthesis and postlayout results for frequency, area, and power consumption for all the router designs. The designs shown in Table II are a mesochronous router with the characterization of resources spent on the router and FIFOs, a 2-phase bundled-data router (2ph-bd), a gated 2-phase bundled-data router (2ph-bd-g), and the Argo router (Argo).

The cell area of the Argo router is 7715 μm^2, as the first column of Table II shows, which is significantly smaller than the mesochronous version. The mesochronous router consists of a synchronous router that has an area comparable to the Argo router (8026 μm^2) and FIFOs on every link that creates a large area overhead (16 213 μm^2). Alternative implementations of the input FIFOs are possible but still create a considerable area overhead. The area of the Argo router is similar to the other asynchronous designs. The 2ph-bd router and the gated version (2ph-bd-g) consume about the same area, as the gating uses logic gates that are already required in the nongated design. The small area difference, 2ph-bd-g router is 1% smaller than 2ph-bd router, is due to heuristics applied by the synthesis tool.

The second and fourth columns show the frequency achieved postsynthesis and postlayout in an environment of eager producers and consumers. The Argo router is faster, due to its simple controller implementation. Finally, the energy per cycle consumed in the routers was evaluated based on the

TABLE III
AREA OF DIFFERENT NI CONFIGURATIONS IMPLEMENTED IN 65-nm CMOS std. CELL TECHNOLOGY

#DMA ctrls	#time slots	Area Breakdown (μm^2) NI logic	DMA tbl	Slot tbl	Total Area (μm^2)
4	8	6131	5466	795	12632
16	23	6655	19565	4215	30395

switching activity by simulating test cases of different link utilization. The third and fifth columns of Table II show the energy per cycle for link utilization of 0%, 70%, and 100%. The energy consumption of the Argo router is much lower than the mesochronous router and in general lower than the other designs that use gating. Additional energy is saved on idle traffic.

B. Network Interface

The NI was synthesized in 65-nm CMOS technology for different NoC sizes. Table III shows figures for two instances of the NI, one for a 2×2 and one for a 4×4 NoC. With all-to-all communication requirements, NIs need one DMA controller, i.e., one outgoing channel per processor. The number of time slots indicates the length of the TDM schedule period required for the communication. The increased cell area of the second instance compared with the first is due to the increased number of DMAs and the bigger schedule. The frequency achieved by the NI for the 2×2 NoC is $\sim$1 GHz. The frequency of the NI for the 4×4 NoC is roughly the same, as it is not affected by the different sizes of the tables.

There are surprisingly few papers addressing the design of NIs, and often the area measures reported can be difficult to compare. Notable exceptions include [25], [53], and [54]. Implementation results of a specific aelite NoC instance designed for a TV-set platform [55] are presented in [25, Sec. 8.1]. This instance comprises 6 routers and 11 NIs and supports 45 bidirectional channels, and was synthesized in 90-nm technology. Buffers in the NIs related to the channel end points account for 85% of the area of the entire NoC [25]. Depending on the buffer implementation, the average area of one NI is calculated to 0.49 mm^2, 0.22 mm^2, or 0.13 mm^2. These figures do not include DMA controllers that are normally placed in the processors. We implemented an instance of our NI, which avoids the buffers, in 90-nm technology for comparison. Including DMA controllers, the area of our NI is 0.024 mm^2; a significantly smaller figure.

For a generalized, technology-independent comparison, Saponara *et al.* [53] reports area measures for typical-size NIs to range from 7 to 50 kgates—a gate being a minimum size two-input NAND. The area of a typical NI instance for the original Æthereal NoC [54], supporting both GS and BE traffic, corresponds to 21 kgates [53]. Considering the area of a minimum drive-strength two-input NAND cell to be 4.4 μm^2, the total size of the Argo NI, including the DMA table, corresponds to 5.5 kgates; a significantly smaller figure.

TABLE IV
AREA FIGURES FOR A SELECTION OF NI IMPLEMENTATIONS WHEN SYNTHESIZED FOR AN ALTERA EP2C70 FPGA

NI design size		NI Logic		TDM and DMA tables		
TDM Period	DMAs	LUTs	FFs	LUTs	FFs	BRAM bits
16	4	326	162	237	374	-
	8	337	162	450	647	-
	16	341	162	116	88	1024
32	4	326	163	286	422	-
	8	339	163	378	579	128
	32	346	163	34	3	2240
64	4	328	164	175	323	192
	8	340	164	378	579	256
	64	351	164	34	3	4544

The static TDM schedule and the channel information need tables in each NI. For each message-passing channel, one entry in the DMA table in the NI of the sending core is needed. With more cores, we expect more channels used per core. For the TDM schedule, one table needs to be sized to the maximum length of the TDM period. With the number of cores increasing, the number of all communication channels increases, and the TDM period increases. To explore the cost of these tables and the scalability with the number of cores, we synthesized different NI configurations to an FPGA. Table IV shows the resource consumption of a single NI for different configurations. Small memories can be implemented in flip-flops; larger ones use the FPGA on-chip memories. Looking at the three configurations with 16, 32, and 64 cycles TDM periods and the 16, 32, and 64 DMA channels, we see a linear increase of the table with $\sim$1, 2, and 4 kb of memory for the DMA and slot tables. These three configurations should be a good fit for 16, 32, and 64 multiprocessor systems. Compared with local memory requirements for the communication SPM and caches for the processor cores, these tables $<$1 kB are of negligible size and we conclude that the NI design scales well up to medium-sized many cores.

C. Complete 4 × 4 Argo NoC

To test the entire design of the Argo, a 4×4 instance in a bitorus topology was synthesized in 65-nm technology. The design includes 16 NIs, 16 routers, and the local FIFOs between NIs and routers (input/output FIFO). The NIs for this design includes 16 DMA controllers to serve one channel per processor, and 23 time slots, for an all-to-all schedule. We also generated a layout of the design to consider the wireload effects after place-and-route. To provide a more realistic example, tiles of 1.5×1.5 mm^2 have been defined. The 2-D bitorus topology of the NoC has been folded to even out link lengths. Details of the layout process are presented in [58].

The first section of Table V presents the area breakdown after place and route, revealing 75% share for the NI. The 18% of the area is attributed to the routers, while the FIFOs contribute only 2%. The links contribute 6%, due to the buffers that the tool added to drive the links.

TABLE V
POST PLACE AND ROUTE AREA BREAKDOWN AND ENERGY DISTRIBUTION FOR THE 4 × 4 ARGO

	Routers	NIs	Links	FIFOs	total
Cell area					
total (μm^2)	127446	537397	43289	12613	720745
per node (μm^2)	7965	33587	2706	788	45047
relative (%)	17.68	74.56	6.01	1.75	100
Energy					
total (pJ/cyc)	96.40	430.80	363.20	13.41	903.81
per node (pJ/cyc)	6.03	26.93	22.70	0.84	56.49
relative (%)	10.67	47.66	40.19	1.48	100

The second section of Table V presents the distribution of the energy per cycle. The captured test case was set up to transmit four packets of random data from every node to every other node based on an all-to-all schedule. This synthetic application models eager traffic producers and consumers. This setup results in average link utilization of 23% as measured during simulation. The table shows that the majority of the energy is consumed by the NIs, with a share of roughly 47%, and by the link driving buffers contributing ∼40%. The routers contribute only ∼10% of the energy, a fact attributed to the gating used in the routers.

Performance evaluation of the overall 4 × 4 instance after layout presents a performance degradation to 450 MHz from 746 MHz achieved in the synthetic environment of eager consumers and producers used for the analysis shown in Table II. This is due to the more realistic environment as well as the delays added by the wires. Adding pipeline stages might be a feasible way to overcome the latter limitation.

The aelite [25], as a virtual circuit switching TDM-based NoC, is a comparable approach to the Argo NoC. The implementation results of an example platform are shown in [25, Sec. 8.1]. The area of this platform, including 6 routers, 11 NIs, and 45 bidirectional channels, synthesized in a 65-nm technology library is calculated to be 2.5 mm^2. The area of our 4 × 4 implementation of Argo, including 16 routers, 16 NIs, and 240 unidirectional channels (all-to-all), in the same technology is 0.72 mm^2, as shown in Table V. Overall, the implementation numbers are not directly comparable, as the configurations differ. Nevertheless, from the above results, we can conclude that depending on the configuration (DMA controllers, TDM schedule, and number of FIFOs) the Argo NoC is roughly 3.5 times smaller.

D. Source Access

We provide the whole NoC, scripts to synthesize it, and the scheduling tool in open source with the industry-friendly BSD license. The NoC can be found at https://github.com/t-crest/argo and the scheduler at https://github.com/t-crest/poseidon.

E. Future Work

The current version of Argo, especially the NIs, supports homogeneous architectures with standard processors connected to the NIs. However, the requirements for digital signal processing (e.g., fast Fourier transform) are better handled by dedicated components than by a general-purpose processor. An extension of the Argo NI will be to support heterogeneous multiprocessor architectures with application-specific accelerators. We started to explore the use of accelerators with the Argo NoC by developing an integration technique [57]. The approach mimics the processor interface by finite state machines to implement transmit and receive streaming channels and to manage the accelerator tasks execution.

Currently, the finishing of a transfer can be queried from the NI by reading out device registers. While this form of polling is a viable technique for hard real-time systems, soft real-time systems can benefit avoiding polling and receiving an interrupt on message transfer. We will extend the NI to deliver interrupts on end of transmitting or receiving a message.

VII. CONCLUSION

This paper presented the Argo NoC; a statically scheduled, time-division-multiplexed NoC supporting message passing among processors in a multiprocessor platform optimized for use in hard real-time systems. Argo supports a GALS timing organization using independently clocked processor cores and mesochronous NIs.

NoCs are typically designed with a mindset of encapsulation and layering, and resources for buffering, flow control, and clock domain synchronization normally account for a significant fraction of the implementation cost (area). Argo avoids almost all of this. A novel NI design that integrates DMA controllers and the TDM scheduling, and the use of asynchronous routers instead of clocked mesochronous routers, has resulted in a design in which messages are transferred end-to-end without any buffering, flow control, or synchronization.

In this paper, we have presented the architecture, design, implementation, and layout of Argo, and provided extensive results characterizing speed, area, and power. To complete the picture, the paper also briefly covered TDM scheduling and the end-to-end latency seen by communicating processor cores.

The message passing is completely time-predictable and the message end-to-end latency statically analyzable. The Argo NoC is at least 3.5 times smaller than existing designs with similar functionality. This makes Argo a communication solution suited to future multiprocessor systems for hard real-time applications.

ACKNOWLEDGMENT

The authors would like to thank the T-CREST and RTEMP project partners for their support, and J. Rodrigues and O. Andersson from the Department of Electrical and Information Technology, Lund University, for their help with the EDA tools.

REFERENCES

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Design Autom. Conf.*, Jun. 2001, pp. 684–689.

[2] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[3] F. Clermidy *et al.*, "A 477 mW NoC-based digital baseband for MIMO 4G SDR," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2010, pp. 278–279.

[4] J. Howard *et al.*, "A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[5] L. A. Plana *et al.*, "SpiNNaker: Design and implementation of a GALS multicore system-on-chip," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 7, no. 4, 2011, Art. ID 17.

[6] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Proc. Design, Autom. Test Eur. (DATE)*, Mar. 2012, pp. 983–987.

[7] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A time-predictable memory network-on-chip," in *Proc. 14th Int. Workshop Worst-Case Execution Time Anal. (WCET)*, 2014, p. 53.

[8] Accellera Systems Initiative. (2013). *Open Core Protocol Specification, Release 3.0*. [Online]. Available: http://www.accellera.org/downloads/standards/ocp/ocp_3.0/

[9] F. G. Moraes, A. Mello, L. Möller, L. Ost, and N. L. V. Calazans, "A low area overhead packet-switched network on chip: Architecture and prototyping," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SOC)*, Dec. 2003, pp. 318–323.

[10] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in *Proc. 11th IEEE Int. Symp. Asynchron. Circuits Syst. (ASYNC)*, Mar. 2005, pp. 54–63.

[11] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs," in *Proc. IEEE 30th Int. Conf. Comput. Design (ICCD)*, Sep. 2012, pp. 45–48.

[12] R. Dobkin, V. Vishnyakov, E. Friedman, and R. Ginosar, "An asynchronous router for multiple service levels networks on chip," in *Proc. 11th IEEE Int. Symp. Asynchron. Circuits Syst. (ASYNC)*, Mar. 2005, pp. 44–53.

[13] T. Felicijan and S. B. Furber, "An asynchronous on-chip network router with quality-of-service (QoS) support," in *Proc. IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2004, pp. 274–277.

[14] K. Goossens and A. Hansson, "The Æthereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2010, pp. 306–311.

[15] T. Bjerregaard and J. Sparsø, "Scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip," in *Proc. 11th IEEE Int. Symp. Asynchron. Circuits Syst. (ASYNC)*, Mar. 2005, pp. 34–43.

[16] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[17] J. Sparsø, E. Kasapaki, and M. Schoeberl, "An area-efficient network interface for a TDM-based network-on-chip," in *Proc. Design, Autom. Test Eur. (DATE)*, Mar. 2013, pp. 1044–1047.

[18] E. Kasapaki, J. Sparsø, R. B. Sørensen, and K. Goossens, "Router designs for an asynchronous time-division-multiplexed network-on-chip," in *Proc. Euromicro Conf. Digital Syst. Design (DSD)*, Sep. 2013, pp. 319–326.

[19] E. Kasapaki and J. Sparsø, "Argo: A time-elastic time-division-multiplexed NOC using asynchronous routers," in *Proc. 20th IEEE Int. Symp. Asynchron. Circuits Syst. (ASYNC)*, May 2014, pp. 45–52.

[20] R. B. Sørensen, J. Sparsø, M. R. Pedersen, and J. Højgaard, "A metaheuristic scheduler for time division multiplexed networks-on-chip," in *Proc. IEEE/IFIP Workshop Softw. Technol. Future Embedded Ubiquitous Syst. (SEUS)*, Jun. 2014, pp. 309–316.

[21] M. Singh and S. Nowick, "MOUSETRAP: High-speed transition-signaling asynchronous pipelines," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 6, pp. 684–698, Jun. 2007.

[22] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2005, p. 155a.

[23] D. Wiklund and D. Liu, "SoCBUS: Switched network on chip for hard real time embedded systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Apr. 2003, p. 78a.

[24] K. Goossens, J. Dielissen, and A. Rădulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test. Comput.*, vol. 22, no. 5, pp. 414–421, Sep./Oct. 2005.

[25] A. Hansson and K. Goossens, *On-Chip Interconnect With Aelite/Composable and Predictable Systems*. New York, NY, USA: Springer-Verlag, 2011.

[26] 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 *Proc. Design, Autom. Test Eur. (DATE)*, Feb. 2004, pp. 890–895.

[27] M. Schoeberl, "A time-triggered network-on-chip," in *Proc. Int. Conf. Field-Program. Logic Appl. (FPL)*, Aug. 2007, pp. 377–382.

[28] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *Proc. 14th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2008, pp. 120–129.

[29] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, Oct. 1995.

[30] M. Harrand and Y. Durand, "Network on chip with quality of service," U.S. Patent 8 619 622, Dec. 31, 2013. [Online]. Available: http://www.google.com/patents/US8619622

[31] J. Sparsø, "Networks-on-chip for real-time multi-processor systems-on-chip," in *Proc. Int. Conf. Appl. Concurrency Syst. Design (ACSD)*, Jun. 2012, pp. 1–5.

[32] J.-Y. Le Boudec, "Application of network calculus to guaranteed service networks," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1087–1096, May 1998.

[33] M. Bakhouya, S. Suboh, J. Gaber, and T. El-Ghazawi, "Analytical modeling and evaluation of on-chip interconnects using network calculus," in *Proc. 3rd ACM/IEEE Int. Symp. Netw.-Chip (NOCS)*, May 2009, pp. 74–79.

[34] S. Zheng, A. Burns, and L. S. Indrusiak, "Schedulability analysis for real time on-chip communication with wormhole switching," *Int. J. Embedded Real-Time Commun. Syst.*, vol. 1, no. 2, pp. 1–22, May 2010.

[35] L. S. Indrusiak, "End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration," *J. Syst. Archit.*, vol. 60, no. 7, pp. 553–561, 2014.

[36] Y. Qian, Z. Lu, and Q. Dou, "QoS scheduling for NoCs: Strict priority queueing versus weighted round robin," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Oct. 2010, pp. 52–59.

[37] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand, "A hybrid packet-circuit switched on-chip network based on SDM," in *Proc. Conf. Design, Autom. Test Eur. (DATE)*, Apr. 2009, pp. 566–569.

[38] G. Chen *et al.*, "16.1 A 340 mV-to-0.9 V 20.2 Tb/s source-synchronous hybrid packet/circuit-switched 16×16 network-on-chip in 22 nm tri-gate CMOS," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2014, pp. 276–277.

[39] P. Ou *et al.*, "A 65 nm 39 GOPS/W 24-core processor with 11 Tb/s/W packet-controlled circuit-switched double-layer network-on-chip and heterogeneous execution array," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2013, pp. 56–57.

[40] K. Goossens, J. Dielissen, A. Radulescu, E. Rijpkema, and P. Wielage, "Electronic device and a method for arbitrating shared resources," WO Patent 2 006 092 768, Sep. 8, 2006.

[41] The International Technology Roadmap for Semiconductors. (2011). *ITRS 2011 Edition—Design*. [Online]. Available: http://www.itrs.net/

[42] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 1998.

[43] I. M. Panades, A. Greiner, and A. Sheibanyrad, "A low cost network-on-chip with guaranteed service well suited to the GALS approach," in *Proc. 1st Int. Conf. Nano-Netw. (Nano-Net)*, Sep. 2006, pp. 1–5.

[44] I. M. Panades and A. Greiner, "Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in GALS architectures," in *Proc. IEEE/ACM Int. Symp. Netw.-Chip (NOCS)*, May 2007, pp. 83–92.

[45] P. Wielage, E. J. Marinissen, M. Altheimer, and C. Wouters, "Design and DfT of a high-speed area-efficient embedded asynchronous FIFO," in *Proc. Design, Autom. Test Eur. (DATE)*, Apr. 2007, pp. 1–6.

[46] J. Bainbridge and S. Furber, "Chain: A delay-insensitive chip area interconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, Sep./Oct. 2002.

[47] B. Flachs *et al.*, "The microarchitecture of the synergistic processor for a cell processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 63–70, Jan. 2006.

[48] D. Berozzi, "Network interface architecture and design issues," in *Networks on Chips*, G. DeMicheli and L. Benini, Eds. San Mateo, CA, USA: Morgan Kaufmann, 2006, ch. 6, pp. 203–284.

[49] J. Sparsø, "Asynchronous circuit design—A tutorial," in *Principles of Asynchronous Circuit Design—A Systems Perspective*, J. Sparsø and S. Furber, Eds. Norwell, MA, USA: Kluwer, 2001, chs. 1–8, pp. 1–152.

[50] J. Liu, S. M. Nowick, and M. Seokl, "Soft MOUSETRAP: A bundled-data asynchronous pipeline scheme tolerant to random variations at ultra-low supply voltages," in *Proc. IEEE 19th Int. Symp. Asynchron. Circuits Syst. (ASYNC)*, May 2013, pp. 1–7.

[51] O. Gangwal, A. Rădulescu, K. Goossens, S. G. Pestana, and E. Rijpkema, "Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip," in *Dynamic and Robust Streaming in and Between Connected Consumer-Electronic Devices* (Philips Research), P. van der Stok, Ed. Amsterdam, The Netherlands: Springer-Verlag, 2005, vol. 3, pp. 1–36.

[52] The Open Group, "D 9.5—Final project report," Tech. Univ. Denmark, Copenhagen, Denmark, Tech. Rep., 2014. [Online]. Available: http://www.t-crest.org/page/results

[53] S. Saponara, L. Fanucci, and M. Coppola, "Design and coverage-driven verification of a novel network-interface IP macrocell for network-on-chip interconnects," *Microprocess. Microsyst.*, vol. 35, no. 6, pp. 579–592, 2011.

[54] A. Radulescu *et al.*, "An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, pp. 4–17, Jan. 2005.

[55] P. Kollig, C. Osborne, and T. Henriksson, "Heterogeneous multi-core platform for consumer multimedia applications," in *Proc. Design, Autom. Test Eur. (DATE)*, Apr. 2009, pp. 1254–1259.

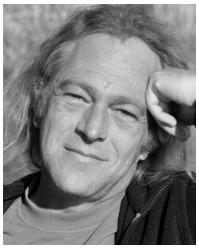
[56] S. R. Vangal *et al.*, "An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[57] L. Pezzarossa, "Hardware accelerators in network-on-chip based multi-core platforms," M.S. thesis, Dept. Appl. Math. Comput. Sci., Tech. Univ. Denmark, Kongens Lyngby, Denmark, 2014.

[58] C. T. Müller, E. Kasapaki, R. B. Sørensen, and J. Sparsø, "Synthesis and layout of an asynchronous network-on-chip using standard EDA tools," in *Proc. NORCHIP*, Oct. 2014, pp. 1–6.

Evangelia Kasapaki received the B.Sc. and M.Sc. degrees from the Department of Computer Science, University of Crete, Rethymno, Greece, in 2006 and 2008, respectively. She is currently pursuing the Ph.D. degree with the Technical University of Denmark, Kongens Lyngby, Denmark.

She has been an Electronic Design Automation Software Engineer with Nanochronous Logic, Inc., San Jose, CA, USA, from 2008 to 2011, when she started her Ph.D. degree. Her current research interests include asynchronous design, networks-on-chip and system-on-chip design, real-time systems, and electronic design automation.



Martin Schoeberl (M'01) received the Ph.D. degree from the Vienna University of Technology, Vienna, Austria, in 2005.

He was an Assistant Professor with the Institute of Computer Engineering from 2005 to 2010. He is currently an Associate Professor with the Technical University of Denmark, Kongens Lyngby, Denmark. He has been involved in a number of national and international research projects, such as JEOPARD, CJ4ES, T-CREST, RTEMP, and the TACLe COST action. He has been the Technical Lead of the EC funded project T-CREST. He has over 100 publications in peer-reviewed journals, conferences, and books. His current research interests include hard real-time systems, time-predictable computer architecture, and real-time Java.

Rasmus Bo Sørensen (S'09) received the M.Sc. degree in computer science from the Technical University of Denmark, Kongens Lyngby, Denmark, in 2012, where he is currently pursuing the Ph.D. degree.

He is involved in time-predictable network-on-chip architectures and programming models.

Christoph Müller was born in Leipzig, Germany, in 1986. He received the bachelor's degree in information technology from the Schmalkalden University of Applied Sciences, Schmalkalden, Germany, in 2011, and the master's degree in system-on-chip from Lund University, Lund, Sweden, in 2013.

He has been a Project Assistant with Lund University, and a Research Assistant with the Technical University of Denmark, Kongens Lyngby, Denmark. His current research interests include digital implementation with an emphasis on low power and design methodology.

Kees Goossens (M'03) received the Ph.D. degree in computer science from the University of Edinburgh, Edinburgh, U.K., in 1993, with a focus on hardware verification using embeddings of formal semantics of hardware description languages in proof systems.

He was with Philips/NXP Research, Eindhoven, The Netherlands, from 1995 to 2010, where he was involved in networks-on-chip for consumer electronics, where real-time performance, predictability, and costs are major constraints. He was a part-time Professor with the Delft University of Technology, Delft, The Netherlands, from 2007 to 2010. He is currently a Professor with the Eindhoven University of Technology, Eindhoven, The Netherlands. He has authored three books and over 100 papers, and holds 24 patents. His current research interests include composable (virtualized), predictable (real-time), low-power embedded systems, and supporting multiple models of computation.

Jens Sparsø (M'98) is currently a Professor with the Technical University of Denmark, Kongens Lyngby, Denmark. He has authored over 70 refereed conference and journal papers, and co-authored a book entitled *Principles of Asynchronous Circuit Design—A Systems Perspective* (Kluwer, 2001), which has become the standard textbook on the topic. His current research interests include design of digital circuits and systems, design of asynchronous circuits, low-power design techniques, application-specific computing structures, computer organization, multicore processors, and networks-on-chips—in short, hardware platforms for embedded and cyber-physical systems.

Prof. Sparsø was a recipient of the Radio-Parts Award and the Reinholdt W. Jorck Award in 1992 and 2003, in recognition of his research on integrated circuits and systems. He received the best paper award at the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC) in 2005. One of his papers was selected as one of the 30 most influential papers of 10 years of the DATE Conference. He is a member of the Steering Committees of ASYNC and the ACM/IEEE International Symposium on Networks-on-Chip.