A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems

Martin Schoeberl, Florian Brandner, Jens Sparsø, and Evangelia Kasapaki
Department of Informatics and Mathematical Modeling
Technical University of Denmark
Email: masca@imm.dtu.dk, flbr@imm.dtu.dk, jsp@imm.dtu.dk, evka@imm.dtu.dk

***Abstract*—This paper explores the design of a circuit-switched network-on-chip (NoC) based on time-division-multiplexing (TDM) for use in hard real-time systems. Previous work has primarily considered application-specific systems. The work presented here targets general-purpose hardware platforms. We consider a system with IP-cores, where the TDM-NoC must provide directed virtual circuits – all with the same bandwidth – between all nodes. This may not be a frequent scenario, but a general platform should provide this capability, and it is an interesting point in the design space to study. The paper presents an FPGA-friendly hardware design, which is simple, fast, and consumes minimal resources. Furthermore, an algorithm to find minimum-period schedules for all-to-all virtual circuits on top of typical physical NoC topologies like 2D-mesh, torus, bidirectional torus, tree, and fat-tree is presented. The static schedule makes the NoC time-predictable and enables worst-case execution time analysis of communicating real-time tasks.**

***Keywords*-real-time systems; network-on-chip**

I. Introduction

Network-on-chip (NoC) design has been an active area of research in academia and industry for the past decade. Today the interconnect fabric of single-chip multi-core systems is typically some form of packet or circuit switched interconnection network. This is the case for general-purpose chip-multi-processors (CMP) as well as for application-specific multi-processor systems-on-chips (MPSoC) used in a large variety of embedded systems. Despite the growing similarity in the way CMPs and MPSoCs are architected, there are some fundamental differences as well: CMPs are typically homogeneous, i.e., built from many identical processors, and the focus is generally on providing the highest possible performance. MPSoCs are typically used in embedded systems and hence dedicated to a specific product or class of products. Thus, they often use a heterogeneous set of processing cores in order to meet energy and performance constraints. In many cases, these systems additionally have to provide hard real-time guarantees.

Much of the recent research in NoCs for real-time MPSoCs is targeting the creation of application-specific hardware platforms. However, the very high and growing cost of developing and fabricating large integrated circuits makes this relevant only for high volume (consumer) products.

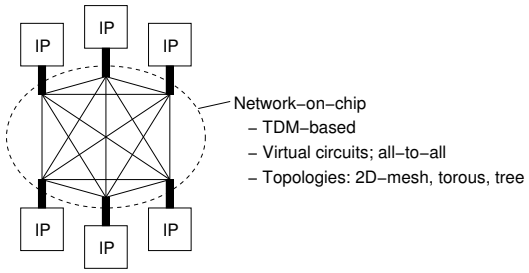

Figure 1. A TDM-based NoC providing all-to-all connections.

Systems that are to be manufactured in smaller quantities must therefore be based on more general-purpose platforms. In this paper we explore the design of a statically scheduled, time-division-multiplexed (TDM) network-on-chip for such general-purpose multi-processor platforms for use in real-time systems.

In order to give hard guarantees on the real-time properties of a system, the designer must be able to analyze and safely estimate the worst-case execution time (WCET) of the different tasks in the system. The fact that the NoC is a shared communication medium comprising multiple independently-arbitrated resources (routers and links) severely complicates timing analysis, and makes it hard to compute the WCET and guarantee real-time behavior. Real-time NoCs have the following options: (i) non-blocking routers with rate control (e.g. Mango [1]); (ii) circuit switching where a connection *owns* the resources providing the connection (SoCBUS [2]); and (iii) virtual circuit switching, for example using TDM (Æthereal [3], aelite [4], and Nostrum [5]).

The work presented in this paper is inspired by the aelite NoC as well as initial discussions within the T-CREST[1] project. In contrast to aelite and its associated CAD tools – aiming at creating *application-specific* platforms derived from use-cases and task-graphs – our aim is to explore the design of a TDM network-on-chip for use in *general-purpose*, real-time MPSoC platforms. Additional goals are: hardware simplicity, speed, and an FPGA-friendly design enabling both chip and FPGA implementations.

We assume a system with n IP-cores, where the TDM-NoC must provide directed virtual circuits – all with the

[1] `http://www.t-crest.org/`

same bandwidth – between all nodes, as shown in Figure 1. In the following we will address the hardware design of such a general-purpose TDM-NoC, and we will explore how to schedule packets and provide all-to-all point-to-point virtual circuits on top of typical physical NoC topologies like 2D-mesh, torus, bidirectional torus, tree, and fat-tree. One of the questions that we try to answer is the following: "What is the minimum period of a schedule that provides virtual circuits with identical bandwidth between all pairs of nodes?" The contributions of the paper are as follows:

- Design of a time-predictable NoC for real-time systems
- An algorithm to generate static TDM schedules
- Study of different topologies with respect to the minimum period of static schedules
- A small, fast, and FPGA-friendly router design

An all-to-all communication schedule might lead to a low utilization. However, a fixed schedule can be implemented in hardware. Therefore, our NoC uses less resource than one with programmable schedule support or additional best-effort traffic support. The performance/cost ratio is more important than fully utilization.

The paper is organized as follows. In the next section, we discuss related work. In Section III we motivate our design with a problem statement. The design of the time-predictable NoC is described in Section IV. In Section V we present an ILP-based algorithm to find minimum-period schedules. Analytical lower bounds of schedule periods are presented in Section VI. In Section VII we present evaluation results for a range of NoC topologies (2D-mesh, torus, bidirectional torus, tree, and fat tree) and provide details of a specific FPGA-based implementation. Section VIII concludes the paper.

II. Related Work

In the following we describe approaches to use NoCs for real-time systems and how to generate schedules for those.

A. Real-Time Network-on-Chips

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

SoCBUS [2] and Wolkotte [6] use a pure circuit-switching NoC, i.e., no resources, such as wires and router buffers, are shared between connections. This lowers utilization and increases costs. However, once a connection has been established, real-time guarantees are trivially achieved. It is, however, possible that a requested connection cannot be set up due to lack of resources (links) – this may compromise the real-time properties.

Æthereal [3] uses TDM, i.e., reserves resources for certain points in time. In each slot of time a block of data is forwarded through a router without waiting or blocking traffic, hence, contention cannot occur. Slot tables with routing information are contained in the routers and no arbitration or link-to-link flow control is required. Instead, a credit-based flow control is applied for end-to-end control, saving buffer space between links. Guaranteed services are combined with best effort routing in order to utilize unreserved resources. aelite, a light version of Æthereal, only offers guaranteed services resulting in a simpler router design [4]. Slot tables are placed in the network interfaces and routing is done through message headers. In the latest version of aelite, called dAElite [7], the static routing tables are back in the routers to support multicast routing.

A time-triggered NoC (TT-NoC) applies the concepts of the time-triggered architecture (TTA) [8] to NoCs [9]. The TT-NoC consists of a ring structure and is therefore only intended for a small number of IP-cores. As the ring is built out of simple multiplexers and registers, it is clocked at double the frequency of the computation nodes. Similar to our presented design, the communication schedule is static and predetermined.

Paukovits and Kopetz use a time-triggered NoC for the time-triggered system-on-chip (TTSoC) architecture [10]. The main difference to other NoC designs is the absolute time format for the TDM slotting, which is called a macro tick. The macro tick is *not* directly related to the clock frequency. The network interfaces (called TISS in TTSoC) are synchronized at the macro tick, which is distributed separately. The TDM-based slot at macro ticks may last several clock cycles at the network clock. The slotting ignores any pipeline effects within the NoC and therefore results in some idle time within links. Our design shares the idea of static scheduling based on TDM. However, we base our schedule on the finer granular network clock and take pipeline effects into account in the network.

Static scheduled communication between processors was explored in the NuMesh project [11]. It is argued that static schedules result in less hardware complexity and the control circuit for the static decisions can be arbitrarily deep pipelined. In contrast to NuMesh, we further simplify the hardware as the schedule is completely fixed and no hardware to support the reprogramming is needed.

The Raw machine [12] is a 16-tile architecture, where each tile contains a MIPS-like processing pipeline and a network router. Similar to our router architecture, the Raw router also contains static routes (besides dynamic routes). The network in Raw is tightly connected to the processing core: the network input and output buffers are mapped to processor registers. This allows a very low latency where an output of the ALU can be routed to the ALU input of

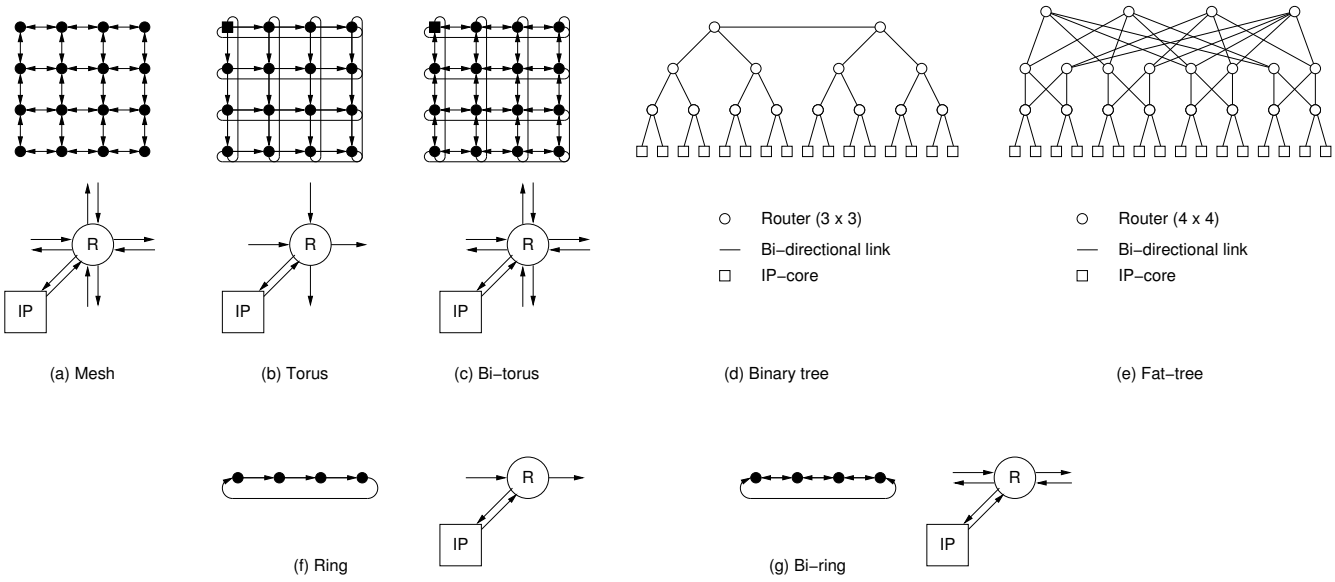

Figure 2. The NoC topologies considered: (a) mesh, (b) torus, (c) bidirectional torus, (d) binary tree using 3×3 routers, (e) fat-tree built from identical 4×4 routers, (f) ring, and (g) bidirectional ring. A filled circle represents a node, comprising a router and an IP-core as shown.

the neighbor processor in 3 cycles. This tight integration of the network and the processor pipeline is the basis for, so-called, software circuits, i.e., applications that resemble ASIC circuits.

B. Routing Schedule Construction

Lu and Jantsch [13] propose a configuration technique for the Nostrum NoC [5] that allows multiple virtual circuits to share buffers of the network. They present a problem formulation that defines a legal allocation of TDM time slots using a backtracking search algorithm. In contrast to our problem, only a single assignment of a given set of virtual circuits is needed that satisfies the required bandwidth and a conflict-free operation of the NoC.

A similar slot allocation problem appears for the Æthereal NoC. The allocation here proceeds in two steps. First, routing paths are determined through the NoC depending on a mapping of an application to the network and the application's communication requirements [14]. Given these paths, TDM time slots are allocated for each virtual circuit in turn [15]. This technique has been extended to split packets and deliver the individual fragments of the packet over multiple paths in order [16]. This approach provides a single solution satisfying the application-specific communication and bandwidth requirements.

The scheduling problem considered in this work can formally be stated as a *dynamic multi-commodity flow* problem over time. A seminal work by Ford and Fulkerson introduced *time-expanded flow networks* to model dynamic flow problems using equivalent static problems [17]. A time-expanded network is a structure containing replications of the network for several time instants (e.g., clock ticks). Fleischer and Skutella study variants of the NP-hard *quickest multi-commodity flow* problem [18] and present a polynomial 2-approximation algorithm. Although closely related, these results apply to general multi-commodity flow problems, where fractional solutions are acceptable. In the context of this work, however, integer solutions are required since the physical hardware resources are indivisible.

III. Real-Time Network-on-Chip

In dependable real-time systems it needs to be guaranteed that all deadlines will be met. This guarantee is performed by schedulability analysis. The input to this schedulability analysis is the worst-case execution time (WCET) of the tasks. To enable WCET analysis, all components of the system (the application software, the processor, the memory subsystem, and the communication network) need to be time-predictable. We aim for a time-predictable NoC that supports WCET analysis.

To enable time-predictable usage of a shared resource the resource arbitration has to be time-predictable. In the case of a NoC, statically scheduled TDM is a time-predictable solution. This static schedule is repeated and the length of the schedule is called the *period*. Like tasks in real-time systems, also the communication is organized in periods. One optimization point of the design is minimizing the period to minimize the latency of delivering flits and the size of the schedule tables.

Many NoCs are intended to be optimized for a given application or application domain. The NoC structure and/or the routing schedules are then optimized and are then

application-specific. While our proposed network can be optimized this way, we aim at a general-purpose solution. The general-purpose solution allows each core to communicate to every other core and the bandwidth is identical for each communication channel. For a general-purpose solution we need to find a single static schedule, which can then be implemented in hardware.

We look at several different NoC topologies and evaluate how well they support this general-purpose static schedule. We consider mesh, torus, torus with bidirectional links (bi-torus), tree, fat-tree, ring, ring with bidirectional links (bi-ring), and bus topologies. Figure 2 shows these topologies. Except for the tree, the fat-tree and the bus, we assume that each topology is composed of n nodes each consisting of an IP-core and a router. With tree structures the IP cores and router do not have a one-to-one mapping. The routers range from 2-ported routers (2 in and 2 out) to 5-ported routers: the mesh (inner nodes) and the bi-torus use 5-ported routers; the fat-tree uses 4-ported routes; the torus, the bi-ring, and the tree use 3-ported routers; and the ring use 2-ported routers.

In this paper we concentrate on the network itself and consider it as a structure that supports communicating streams of flits. Designing the network interface and the flow control are out of the scope of this paper.

IV. Network Design

A static schedule guarantees latency and bandwidth for sending data over the NoC, which itself enables WCET analysis of tasks. Furthermore, this static schedule allows optimization of the routers. With our design we avoid transmitting the packet route via the network, but keep the network schedule in the routers. With a predefined schedule there are no collisions possible.

The simple router, as shown in Figure 3, consists of multiplexers and registers. This structure fits very well to the structure of a logic cell (LC) in an FPGA. A LC usually contains a lookup table (LUT) and a register. The LUT is used to build combinational logic (e.g., the multiplexer). Although we do not want to restrict our NoC to FPGAs, we consider FPGAs as an important platform and aim for an FPGA-friendly design.

A. Packet Organization

As the static schedule is contained in the router, the flits traveling through the network contain only data and no routing information. The time slot the flit is injected to the network implicitly gives the destination address. Without the address header we are less constrained on packet lengths – we do not need to amortize for the address overhead. Therefore, we define that a packet is a single flit long and use schedules for single clock/flit granularity. This short packet length minimizes the latency for short data transportation.

The link width is usually determined by the length of the address field (the packet header) and therefore depends on the network size. Free of these constraints, we can use any link width that is needed for the bandwidth requirements. The resource consumption of a router is directly related to the link width. Therefore, we can trade bandwidth with resource consumption.

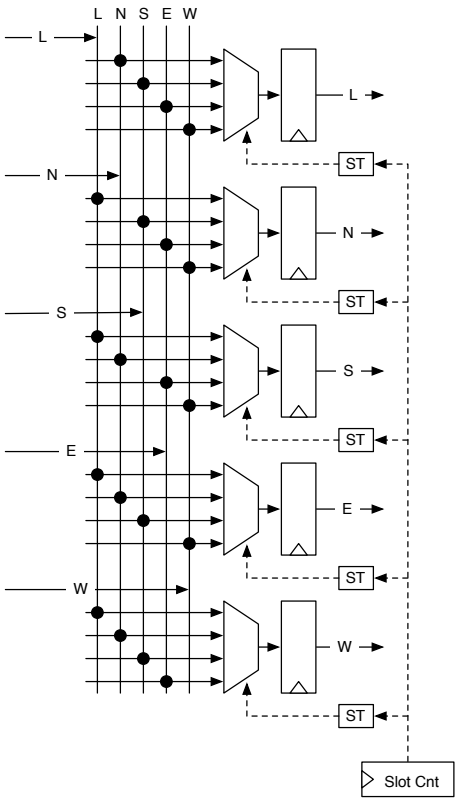

Figure 3. Connections of the multiplexer based router

At the lowest level the individual flits are considered to be individual words of a data stream – similar to a serial line connection. The only control structure is a valid bit in the flit. The framing of the data, the forming of longer packets (e.g., cache line fills), and the meaning of the stream is defined in the network interface. Or in other words: the NoC just represents a transport layer with end-to-end channels.

Using single flit packets and a static schedule results in a deadlock free design.

B. Router

One of the benefits of a static scheduled NoC is the simplification of the network routers. Figure 3 shows the router for our NoC. A router consists of registers for a single flit at the output ports and a multiplexer feeding this register from each input port. Due to the static schedule there is no need for dynamic arbitration or additional buffers. Furthermore, there are no flit collisions or deadlocks possible. Flits move each clock cycle one hop. They cannot stay within the routers' registers.

Figure 3 shows a router for a mesh or bi-torus NoC with 5 input and 5 output ports. Input and output L stands for the local connection to the IP-core; N, S, E, and W represent the four directions in a mesh-style network. Each multiplexer is connected to four inputs. The black dots represent a connection point. The box with ST represents the schedule table for each multiplexer. A slot counter drives the schedule table. This counter can be a global one. To avoid long wires from this global schedule counter the distribution can be pipelined or the counter can simply be replicated locally in every router.

C. Schedule Tables

In [4] schedule tables in the routers have been avoided due to high resource consumptions. However, in our design two properties make schedule tables in routers attractive: (a) a fixed schedule (compared to a configurable) is cheaper to implement (in a ROM); and (b) the size of the table (the schedule period) is relatively small for the symmetric bandwidth configuration. Each router has a table for the static schedule, which basically includes just the multiplexer selection. The schedule period is quite short. Therefore, the resource consumption for this table is low. Let us consider a small network, a 3x3 mesh network, as an example. For this network the ideal static schedule is 10 entries long – in each clock cycle each IP-core injects a flit into the network and receives one flit. Let us assume that we cannot find this ideal schedule, but one that fits into a maximum of 16 entries. With FPGA technology, 16 entries of a schedule table fit perfectly into the available 4-bit LUT. Therefore, for each multiplexer, which has 2-bit select lines, 2 LUTs are needed to represent the schedule table.

D. Router Size Estimation

To set the size of the schedule table in relation to the complete router resource consumption we assume for now a 16-bit link. A 4:1 multiplexer can be built out of two 4-bit LUTs. Each LC contains one register. The LUT used for the multiplexers can share the same LC for the registers. Therefore, a 16-bit multiplexer plus register consumes 32 LCs. The overhead for the schedule table for up to 16 schedule entries is 2 LCs, in other words 6.25%. A 4-port router with 16-bit links consumes overall 136 LCs, which is way smaller than other designs that need additional buffering, dynamic arbitration, and/or pipeline stages.

V. Finding the Schedule

A corner stone of our NoC is the assumption of a periodic communication schedule that allows every node in the network to communicate with any other node. This schedule is fixed, does not change during the operation of the network, and can thus be pre-computed off-line. Computing optimal schedules is rather time consuming, however, it needs to be done only once for every network configuration.

We formulate the problem of finding an optimal schedule as a generic, linear optimization problem (ILP) that is solved using CPLEX. The input to our formulation is an instance of an arbitrary network topology, specifying the number of nodes and routers, and the interconnect structure. Such an instance is described by the graph $G = (R, I, L)$, where R represents the set of routers available in the NoC, I denotes the set of IP-cores connected to the network, and $L \subseteq (R \cup I) \times (R \cup I)$ represents the set of directed network links. We assume that the network operates synchronously and transmits flits between routers and/or IP-cores on every clock tick. Links may only be used to transfer one flit at a time, while we assume that routers and IP-cores do not impose any constraints on the schedule, e.g., routers and IP-cores are capable of processing all incoming and outgoing flits at every time instant. Furthermore, flits cannot be stalled at a link, which implies that a flit has to be routed to another link on the next clock tick.

Given a network instance $G = (R, I, L)$ we generate a set of linear equations that model the flow of flits in the network over a time period $[0, T]$, where T represents an upper bound of the schedule length. In a first step we generate a time-expanded network $G^T = (R^T, I^T, L^T)$ which is defined as follows: (1) $R^T = \{r^t | t \in [0, T], r \in R\}$, (2) $I^T = \{c^t | t \in [0, T], c \in I\}$, (3) $L^T = \{l^t = (u^t, v^{t+1}) | t \in [0, T-1], l = (u, v) \in L\}$. This approach is very similar to time-expanded networks of dynamic multi-commodity flow problems [17].

A. Variables

We then introduce ILP variables for every link in the time-expanded network G^T in order to express the flow of flits within the network. A binary variable $\ell^t_{l,c}$, where $l^t \in L^T$ and $c \in I$, represents the use of the network link l at time instant t in order to send a flit to IP-core c. Note that the variable does not cover the source IP-core sending the flit. More formally we introduce the following variables:

$$\mathcal{V}(G^T) = \{\ell^t_{l,c} | l^t \in L^T, c \in I\} \quad (1)$$

B. Constraints

Next, we define constraints on the variables in $\mathcal{V}(G^T)$ expressing legal flows of flits through the NoC. These constraints ensure three properties: (1) every flit on a network link leading to a router at one time instant will be forwarded to another network link leading from the router on the next time instant, (2) every link is used to transmit at most one flit at any moment in time, and (3) every IP-core sends and receives a flit to/from every other IP-core on the network. In the following we will make use of the functions $In(n^t) = \{l^t | l^t = (u^{t-1}, n^t) \in L^T\}$ and $Out(n^t)$, which is defined analogously, that return the incoming and outgoing links of a router or IP-core in the time-expanded network.

The first constraint ensures that flits are never lost within the network. For every router $r^t \in R^t$, $t \in [0, T-1]$ the

following equation has to be fulfilled:

$$\sum_{i^t \in In(r^t)} \sum_{c \in I} \ell^t_{i,c} - \sum_{o^{t+1} \in Out(r^{t+1})} \sum_{c \in I} \ell^{t+1}_{o,c} = 0 \qquad (2)$$

The second constraint ensures that every network link $l^t \in L^T$ is used to transmit at most one flit on every time instant $t \in [0, T]$:

$$\sum_{c \in I} \ell^t_{l,c} = 1 \qquad (3)$$

Finally it remains to ensure that flits are actually sent from every IP-core on the network to every other IP-core, we thus add the following equations to the optimization problem for every IP-core $c \in I$ and every destination IP-core $s \in I, s \neq c$:

$$\sum_{l^t \in In(c), t \in [0,T]} \ell^t_{l,c} = |I| - 1 \qquad (4)$$

$$\sum_{l^t \in Out(c), t \in [0,T]} \ell^t_{l,s} = 1 \qquad (5)$$

C. Solving

We then ask the CPLEX optimizer to find a schedule under the given constraints that minimizes the schedule length, which can easily be derived from the time index of the ILP variables in $\mathcal{V}(G^T)$ representing the usage of network links from above. In addition to minimizing the schedule length, we also assign a small weight to every use of a link to transmit a flit. This minimizes the overall length of the paths within the schedule and, among others, helps to avoid useless cycles in the schedule.

Note that, due to the intrinsic complexity of solving ILP problems, the size of the network, and the use of a time-expanded network to model the network behavior over time, solving the equation system from above is time consuming. However, this approach is simple to realize and very powerful. It allows to model arbitrary network topologies, to model constraints on the usage of the network and its links, and to express constraints on time instants when flits are sent and/or received, et cetera.

VI. Analytical Bounds on the Period

By considering different bandwidth limits of the various network topologies, we can derive a set of analytical lower bounds on the schedule's period P.

We assume a system with n IP-cores that are connected to the network using bidirectional links, such that an IP-core can send and receive one flit per clock period. During a period P each node is assumed to send a flit to all the other IP-cores in the system. This means that the total number of flits injected into the network during a period P is $n(n-1)$. Based on these simple assumptions we can derive three fundamental lower bounds on the period:

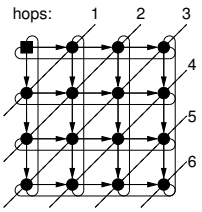

Figure 4. 4×4 Torus: No. of hops to reach from one node to all other nodes.

- *IO-bound, B_{IO}*: The total amount of flits communicated during a period has to be communicated across the n links connecting the IP-cores to the network. For a given number of IP-cores, n, this figure is the same for all the different topologies:

$$B_{IO} = \frac{n(n-1)}{n} = n - 1 \qquad (6)$$

- *Link capacity bound, B_{Cap}*: The total number of link-hops required to communicate the total amount of flits through the network (assuming the shortest paths), divided by the total number of links in the network.
- *Bisection capacity bound, B_{Bisect}*: The total amount of flits communicated across a bisection boundary ($(n/2)^2$), divided by the number of links crossing the bisection boundary.

In the following we will derive the above three bounds for a range of common NoC topologies, as shown in Figure 2. These bounds, and actual minimal periods for specific schedules are shown in Table I which is introduced and discussed later in Section VII. For some of the topologies we derive analytical expressions, for others we calculate actual figures. When counting the number of links in the mesh, the torus, the bi-torus, the ring, and the bi-ring, we do not consider the links that connect IP-blocks to routers or routers to IP-cores. This is because the full amount of traffic is also carried across the links in the "core network", i.e., by the links that connect pairs of routers. In this way we get a tighter bound than if the IP-router and router-IP links were also included in the count. Assuming $n = m^2$ the number of links in the core network is $2 \cdot 2 \cdot m \cdot (m-1)$ for a mesh, $2 \cdot m^2$ for a torus and $4 \cdot m^2$ for a bi-torus.

Torus: Due to its symmetry, every node in a torus sees the same distances to the $n-1$ nodes to which it transmits flits. With the help of Figure 4 we can determine the capacity bound $B_{Cap,Torus}$ for a 16-node, 4x4 torus. For simplicity we consider the upper left node that does not make use of the links that wrap around the edges. We first compute the total number of hops made by all the flits send from the node considered, to all the other $n-1$ nodes. From the figure we see that in 1 hop 2 nodes are reachable, in 2 hops 3 nodes are reachable, etc. resulting in a total hop count of:

$$1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + 4 \cdot 3 + 5 \cdot 2 + 6 \cdot 1 = 48 \qquad (7)$$

Multiplying this figure by the number of nodes and dividing by the total number of uni-directional links (2 per node in a torus) we get:

$$B_{Cap,Torus,4\times4} = \frac{48 \times 16}{32} = 24 \tag{8}$$

Assuming quadratic structures with a side-length of m, i.e. $n = m^2$, it is possible to derive a general expression:

$$B_{Cap,Torus} = \frac{m^3 - m^2}{2} \tag{9}$$

Finally, assuming that m is even (resulting in equal-sized bisection partitions), we derive a general expression for the minimum period determined by the bisection capacity. It is calculated as the number of flits transmitted from one partition to the other($m^2/2 \cdot m^2/2$), divided by the number of links crossing the bisection boundary in that direction (m):

$$B_{Bisect,Torus} = \frac{m^3}{4} \qquad \text{(for } m \text{ even)} \tag{10}$$

Mesh: Due to the many special cases, which have to be considered in the non-symmetric mesh topology, it is more involved to derive an analytical bound. The basic idea is to sum the minimal distances between all pairs of IP-cores in the NoC, which gives after simplification:

$$B_{Cap,Mesh} = \frac{m^2\ (m+1)}{6} \tag{11}$$

The bisection bound is the same as for the torus (again assuming that m is even):

$$B_{Bisect,Mesh} = \frac{m^3}{4} \qquad \text{(for } m \text{ even)} \tag{12}$$

Bi-torus: For the bi-torus, the capacity bound is determined in the same way as for the torus, but as the links are bidirectional the analysis considers a center node transmitting to all other nodes. When m is odd, the situation is fully symmetrical, and for this we have the following general expression:

$$B_{Cap,Bi-torus} = \frac{m^3 - m}{8} \qquad \text{(for } m \text{ odd)} \tag{13}$$

The bi-torus has double the bisection bandwidth of a torus, and therefore the period-bound is half:

$$B_{Bisect,Bi-torus} = \frac{m^3}{8} \qquad \text{(for } m \text{ even)} \tag{14}$$

Trees: In the binary tree there is one link to support the traffic that crosses the bisection boundary. When the number of IP-cores increases this obviously becomes the bottleneck and the period will be limited by the (constant) bisection capacity.

In the fat-tree the bisection bandwidth increases with the number of IP-cores. In the fat tree with 4 IP-cores there are 2 links crossing the bisection boundary, in a system with 8 IP cores there are 4 links, and in a system with 16 IP-cores there are 8 links, etc.

Ring: It is possible to devise an optimal schedule for a ring, and thereby determine the minimum period, analytically. In a unidirectional ring each router either forwards a flit from its neighbor router or it injects a flit from its attached IP-core. As the ring is symmetric we can easily derive a static schedule, where each node performs the same communication pattern. Within the first cycle the flit is sent to the next neighbor, which can be performed in a single hop. During the next two cycles a flit is sent to the node two hops away. This is continued until the last flit, which travels $n-1$ hops. The resulting period P is:

$$P_{Min,Ring} = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \tag{15}$$

A unidirectional ring results in the minimum hardware cost, but flits that travel almost the whole ring consume a lot of those communication resources. Adding a second ring in the other direction reduces the maximum travel length of a flit to the half. The period for a double ring is:

$$P_{Min,Bi-ring} = \sum_{i=1}^{\lceil \frac{n-1}{2} \rceil} i = \frac{(\lceil \frac{n-1}{2} \rceil + 1)\lceil \frac{n-1}{2} \rceil}{2} \tag{16}$$

With the above given schedule, and period of that schedule, all links are used. Therefore, the capacity bound is equal to the derived period. Furthermore, as we have the optimal period there is no incentive to show a bisectional bound of the period.

VII. Evaluation Results

For all the topologies shown in Figure 2, and for a range of network sizes, Table I lists: the amount of resources (number of routers and links), the 3 analytical lower bounds on the period: B_{IO}, B_{Cap} and B_{Bisect}, and the smallest period, P_{Min}, for which we have been able to find an actual schedule using the ILP formulation. It should be noted that B_{Bisect} can only be calculated when $n = m^2$ is even using one of the equations from Section VI. The figures for odd values of n have been calculated using the same equations, rounded up, and as a reminder that the values are not exact; they have been put in parenthesis. For easy comparison with P_{min}, we have highlighted the largest of the 3 bounds.

A. Discussion

We see that mesh and torus topologies become very fast bisection and capacity bounded when the number of nodes increases to a medium sized 4x4 network. The practical schedules that we found also reflect this. With larger structures we see a considerable growth of the period.

For large m the total number of hops is 4 times lower in a bi-torus than in a torus. Comparing $B_{Cap,Torus}$ and

Table I
RESOURCES, PERIOD BOUNDS, AND MINIMAL PERIODS FOR STATIC SCHEDULED NETWORKS.

		NoC Resources		Period Bounds			
Topology	IPs	Routers	Links	IO	Cap.	Bisect.	Min.
Mesh	2x2	4	8+8	**3**	2	2	5
	3x3	9	24+18	**8**	6	7	10
	4x4	16	48+32	15	14	**16**	18
	5x5	25	80+50	24	25	**(32)**	34
Torus	2x2	4	8+8	**3**	2	2	5
	3x3	9	18+18	8	**9**	(7)	11
	4x4	16	32+32	15	**24**	16	26
	5x5	25	50+50	24	**50**	(32)	52
	6x6	36	72+72	35	**90**	54	n.a.
Bi-torus	2x2	4	16+8	**3**	1	1	4
	3x3	9	36+18	**8**	3	(4)	10
	4x4	16	64+32	**15**	8	8	18
	5x5	25	100+50	**24**	15	(16)	28
	6x6	36	144+72	**35**	27	27	n.a.
	7x7	49	196+98	**48**	42	(43)	n.a.
	8x8	64	256+128	63	**64**	**64**	n.a.
	9x9	81	324+162	80	90	**(92)**	n.a.
Tree	4	2	2x5	3		**4**	
	8	6	2x13	7		**16**	
	16	14	2x29	15		**64**	
Fat tree	4	4	2x8	**3**		2	3
	8	8	2x24	**7**		4	9
Ring	4	4	4+8	3	6		6
	9	9	9+18	8	36		36
	16	16	16+32	15	120		120
	25	25	25+50	24	300		300
Bi-ring	4	4	2x4	3	3		3
	9	9	2x9	8	10		10
	16	16	2x16	15	36		36
	25	25	2x25	24	78		78
Bus	4	4		3	12		12
	9	9		8	72		72
	16	16		15	240		240
	25	25		24	600		600

$B_{Bisect,Torus}$ one can see that for increasing values of m, the torus is always limited by the capacity bound, and never the bisection bound. For all other structures, the bisection bound is always the larger one (for large enough values of m). If links are pipelined, the network capacity increases and topologies that are not limited by IO or bisection bounds may benefit from pipelining. The bidirectional torus and the fat tree provide enough link capacity to build reasonable sized networks where the schedule period is still bounded by the IO capacity.

Finding an optimal solution for torus and bi-torus networks for sizes larger than 5x5 were not yet possible, due to the computational complexity. We thus plan to investigate fast optimal, or near-optimal, algorithms to derive schedules for larger problem instances. For instance, symmetric topologies such as the bi-torus permit schedules where every router in the NoC performs the same action. The problem of finding a good schedule thus could be reduced to finding a spanning tree that maximizes the number of flits in-flight at the same time. This idea could also be adapted for other topologies that are somewhat symmetric, e.g., mesh or torus, where *almost all* routers could take identical routing decisions.

It is important to note that application-specific optimizations, as found in other real-time NoCs, can also be realized in our framework, e.g., by adding constraints to the ILP formulation or providing a tailored network topology.

Table II
RESOURCES AND FREQUENCIES FOR DIFFERENT SIZES OF MESH NOCS WITH PROCESSOR CORES

	Logic cells			Frequency (MHz)	
Size	Total	NoC	Processor	Processor	NoC
2x2	1132	48–105	217–222	127	371
3x3	2726	48–112	217–223	130	327
4x4	5166	48–145	218–222	130	280
5x5	8287	48–146	218–226	134	264
6x6	12095	48–150	217–223	128	254
7x7	16662	48–160	217–224	125	223
8x8	22006	48–160	217–225	125	230
9x9	28113	48–160	219–224	127	227

B. FPGA Implementation

We have implemented the mesh network topology in an FPGA and used a small processor (Leros [19]) as IP core. The number of nodes is configurable and we explored networks of different size. We used a flit/link width of 16 bits.

We evaluated the maximum attainable clock frequency and its dependency on the size of the design. To explore the NoC limits we use a PLL to generate two clocks: a 100 MHz clock for the processor and a 400 MHz clock for the NoC. For our experiments we use a Cyclone III device from Altera. The Cyclone device belongs to the low-cost FPGA family from Altera. We left the synthesis tool Quartus select the smallest FPGA for each configuration. To report the maximum frequency of the NoC we use the slow timing model at 85° C and 1200 mV core voltage.

The results of the experiments of different sizes of mesh NoCs are shown in Table II. The processor node consumes about 220 LCs, as it is a processor optimized for low resource consumption. Simple 32-bit RISC pipelines in an FPGA consume about 2000 LCs. The router, as it appears on the table, occupies a small number of LCs (48-160), for all implementation sizes, as it is a simple design. A router in the middle of the network consumes, as expected, about 150 LCs. Routers at the sides are considerable smaller. The operation frequency is also high (223-371 MHz) compared to the processor frequency (125-134 MHz). It is double the frequency of the processor even for larger NoCs. Therefore, one design option is to clock the NoC at double the clock frequency of the IP-cores and reduce the link width to half.

VIII. Conclusions

Hard real-time systems need an architecture with processors and communication channels where upper bounds on the worst-case execution time (WCET) can be statically derived. To provide such bounds, all resources have to be arbitrated using a static TDM schedule. This schedule has to be considered during the WCET analysis.

In this paper we explored statically scheduled, time-division-multiplexed network-on-chips for such real-time systems. We investigated different topologies and derived optimal static schedule tables by solving an integer linear programming problem. Static schedules that allow communication between all processing nodes result in a considerable pressure on the network bandwidth. For networks above 16 nodes, only bi-torus and fat trees have enough link capacity to enable a schedule period that is in the same range as the IO capacity of the IP cores.

The presented NoC design and the program for the schedule generation are provided in open-source under the BSD license. The source can be found at `https://github.com/t-crest/s4noc`.

Acknowledgments

This work received funding from the FP7-ICT Project 288008 Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

References

[1] T. Bjerregaard and J. Sparsø, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE Computer Society Press, 2005, pp. 1226–1231.

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

[3] K. Goossens and A. Hansson, "The AEthereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC 2010)*, 2010, pp. 306 –311.

[4] A. Hansson, M. Subburaman, and K. Goossens, "aelite: a flit-synchronous network on chip with composable and predictable services," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2009)*, Leuven, Belgium, 2009, pp. 250–255.

[5] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The nostrum backbone-a communication protocol stack for networks on chip," in *Proceedings of the 17th International Conference on VLSI Design*, 2004, pp. 693 – 696.

[6] P. T. Wolkotte, G. J. Smit, G. K. Rauwerda, and L. T. Smit, "An energy-efficient reconfigurable circuit switched network-on-chip," in *Proc. Int'l Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

[7] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens, "A TDM NoC supporting QoS, multicast, and fast connection set-up," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE 2012)*, 2012.

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

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

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

[11] S. Ward, K. Abdalla, R. Dujari, M. Fetterman, F. Honoré, R. Jenez, P. Laffont, K. Mackenzie, C. Metcalf, M. Minsky, J. Nguyen, J. Pezaris, G. Pratt, and R. Tessier, "The numesh: a modular, scalable communications substrate," in *Proceedings of the 7th international conference on Supercomputing*, ser. ICS '93. New York, NY, USA: ACM, 1993, pp. 230–239.

[12] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[13] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, pp. 1021–1034, August 2008.

[14] A. Hansson, K. Goossens, and A. Radulescu, "A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic," *VLSI Design*, vol. 2007, p. 16, 2007.

[15] R. Stefan and K. Goossens, "An improved algorithm for slot selection in the æthereal network-on-chip," in *Proceedings of the International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, ser. INA-OCMC '11. ACM, 2011, pp. 7–10.

[16] ——, "A TDM slot allocation flow based on multipath routing in NoCs," *Microprocess. Microsyst.*, vol. 35, pp. 130–138, March 2011.

[17] L. R. Ford and D. R. Fulkerson, "Constructing maximal dynamic flows from static flows," *Operations Research*, vol. 6, pp. 419–433, 1958.

[18] L. Fleischer and M. Skutella, "The quickest multicommodity flow problem," in *Proceedings of the International Conference on Integer Programming and Combinatorial Optimization*, ser. IPCO '02. Springer, 2002, pp. 36–53.

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