T-CREST

TIME-PREDICTABLE MULTI-CORE ARCHITECTURE
FOR EMBEDDED SYSTEMS

Project Number 288008

D 3.1 Survey of time-predictable NOCs and their WCET analysis

Version 1.0
31 May 2012
Final

Public Distribution

Technical University of Denmark, Eindhoven University of Technology

Project Partners: **AbsInt Angewandte Informatik**, **Eindhoven University of Technology**, **GMVIS Skysoft**, **Intecs**, **Technical University of Denmark**, **The Open Group**, **University of York**, **Vienna University of Technology**

Project Partner Contact Information

AbsInt Angewandte Informatik Christian Ferdinand Science Park 1 66123 Saarbrücken, Germany Tel: +49 681 383600 Fax: +49 681 3836020 E-mail: ferdinand@absint.com	**Eindhoven University of Technology** Kees Goossens Potentiaal PT 9.34 Den Dolech 2 5612 AZ Eindhoven, The Netherlands E-mail: k.g.w.goossens@tue.nl
GMVIS Skysoft João Baptista Av. D. Joao II, Torre Fernao Magalhaes, 7 1998-025 Lisbon, Portugal Tel: +351 21 382 9366 E-mail: joao.baptista@gmv.com	**Intecs** Silvia Mazzini Via Forti trav. A5 Ospedaletto 56121 Pisa, Italy Tel: +39 050 965 7513 E-mail: silvia.mazzini@intecs.it
Technical University of Denmark Martin Schoeberl Richard Petersens Plads 2800 Lyngby, Denmark Tel: +45 45 25 37 43 Fax: +45 45 93 00 74 E-mail: masca@imm.dtu.dk	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 Fax: +32 2 675 7721 E-mail: s.hansen@opengroup.org
University of York Neil Audsley Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 325 500 E-mail: Neil.Audsley@cs.york.ac.uk	**Vienna University of Technology** Peter Puschner Treitlstrasse 3 1040 Vienna, Austria Tel: +43 1 58801 18227 Fax: +43 1 58801 918227 E-mail: peter@vmars.tuwien.ac.at

Contents

Document Control

Version	Status	Date
0.1	First draft	30 April 2012
1.0	Final version	31 May 2012

Executive Summary

This survey addresses the topic of networks-on-chip (NOCs) for real-time multi-processor systems and their worst-case execution time (WCET) properties. Real-time systems are required to provide service guarantees on their execution. In order to do that, each component of the system needs to be time-predictable. A network is a shared resource, that facilitates different communication needs with varying requirements. Real-time guarantees and WCET analysis of a shared resource are not easily estimated. A WCET analysis model is suggested in this survey, which considers latency and throughput as the fundamental timing parameters. A number of state-of-the-art time-predictable NOCs are reviewed. The focus is to present the basis of their functionality and describe their timing behavior. An effort to formally describe the performance parameters affecting the execution timing is made. An immediate comparison between the NOCs mentioned is not feasible but the analysis reveals their timing characteristics and gives useful insight in their performance and WCET analysis.

1 Introduction

Embedded systems today follow a modular design principle and they are usually used in real-time systems. In such a system timing behavior and specifically the WCET (worst-case execution time) is of great importance. These systems have hard requirements on processing and communication services. This survey addresses the topic of NOC design, but focuses on the aspects of NOCs that are suitable for real-time systems and existing designs of such networks.

The survey is organized as follows. In chapter 2 the basic concepts of NOCs are being presented as well as some general aspects in the field. Chapter 3 examines specific aspects of real-time NOCs and discusses how to quantify the influence of the NOC on the WCET of a process executing on a processing node. Chapter 4 reviews examples of real-time NOC designs and presents a discussion of their WCET-related properties. Finally, chapter 5 contains some discussion over the presented issues on this survey.

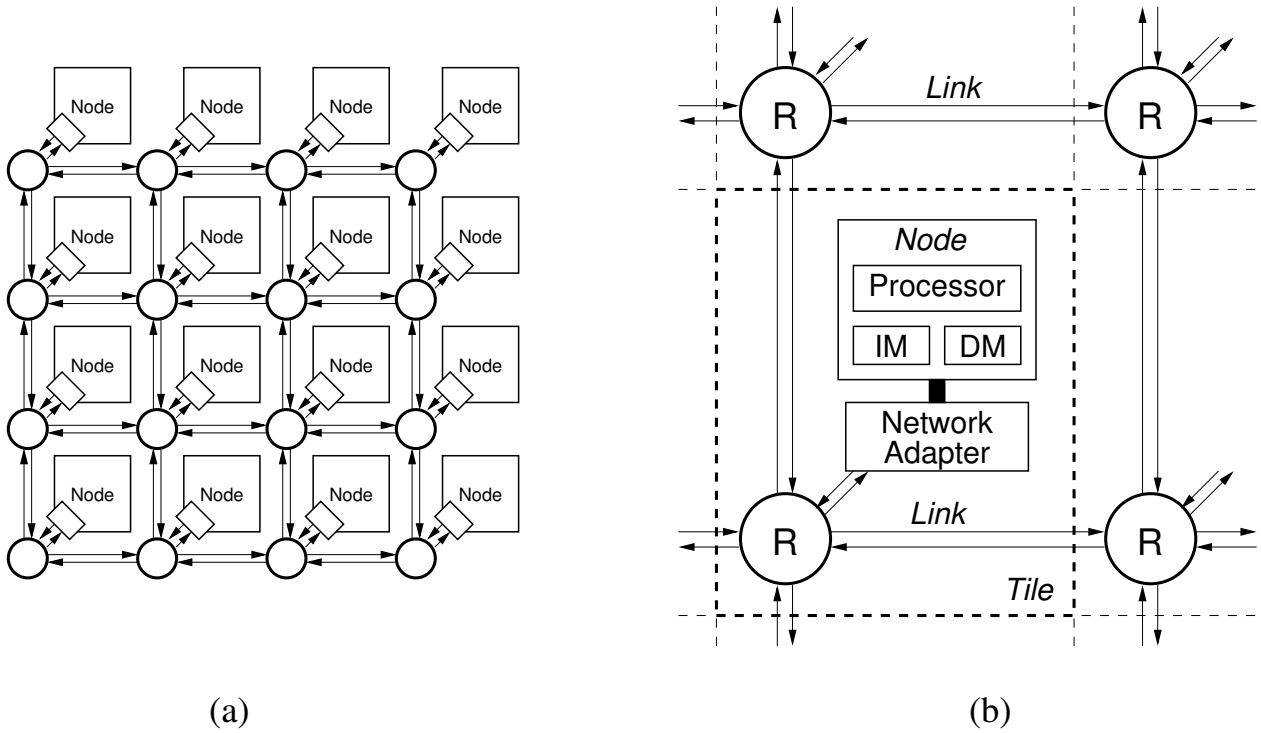


Figure 1: An example NOC based MPSoC: (a) 2D mesh NOC topology. (b) Details of a node/tile.

2 NOCs Basics

This chapter presents the basic aspects of a network-on-chip, that are needed to understand the concepts in the following chapters.

Figure 1 shows an example of a NOC-based MPSoC platform. The NOC itself consists of a structure of *routers* and *links* implementing a packet-switched communication fabric, and a set of *network adapters* (NA) that bridges from the packet-switched interconnect to the memory-style read-write transaction interfaces offered to the *nodes* connected by the NOC. A *n*ode is typically a processor with some amount of local memory. Very often the physical implementation of the platform use a *tile-based* approach where tiles are simply replicated and where connections are established by abutment, Figure 1(b). The network adapter maps a read or a write transaction (to a remote slave device) into a *packet* and this typically includes generating a header that is used to route the packet through the NOC. A packet consists of a sequence of *flits* that are transmitted in immediate succession. A network adapter may divide a transaction that involves reading or writing a block of words into a number of packets. In the processor node shown in Figure 1(b) IM and DM represents some form of instruction and data memories which can be caches and/or explicitly managed scratchpad memories. A node can also be a slave device, for example a DRAM-controller providing access to some larger and shared off-chip memory or an IO-controller.

The processor nodes in the T-CREST platform will include both cache memories and scratch pad memories, and one of the nodes in the platform will be a DRAM memory controller providing access to a shared bank of DRAM. The local scratch-pad memories and the off-chip DRAM will be

mapped into one single address space and in principle a processor in one node may access the scratch memories in all nodes as well as the off-chip DRAM.

3 Real-time NOCs and WCET analysis

3.1 Introduction

Most of the NOC-designs published in the literature are designed to support best-effort traffic only, and the focus is typically on minimizing average-case latency and on maximizing bandwidth utilization. The fact that a NOC is a shared communication medium comprising multiple independently-arbitrated resources (routers or links) may severely complicate timing analysis. The seemingly simple question: "What is the latency that the NOC adds to a read or write transaction towards a memory in a remote node?" can be very difficult to answer. In a multi-processor platform (like the T-CREST platform) with caches, local memories, and a block of shared main memory, the traffic in the NOC depends on the execution history of all processor nodes. This makes it extremely hard to compute the worst-case execution time (WCET) of a given program executing on a given processor node, and thus guaranteeing the real-time behavior of this program.

A NOC for a real-time platform must allow guarantees on bandwidth and latency to be made for individual processor-to-memory transactions, and this calls for solution that provides some form of end-to-end connection. There are essentially two ways of achieving this: non-blocking routers with rate control and circuit switching, possibly with Time Division Multiplexing (TDM).

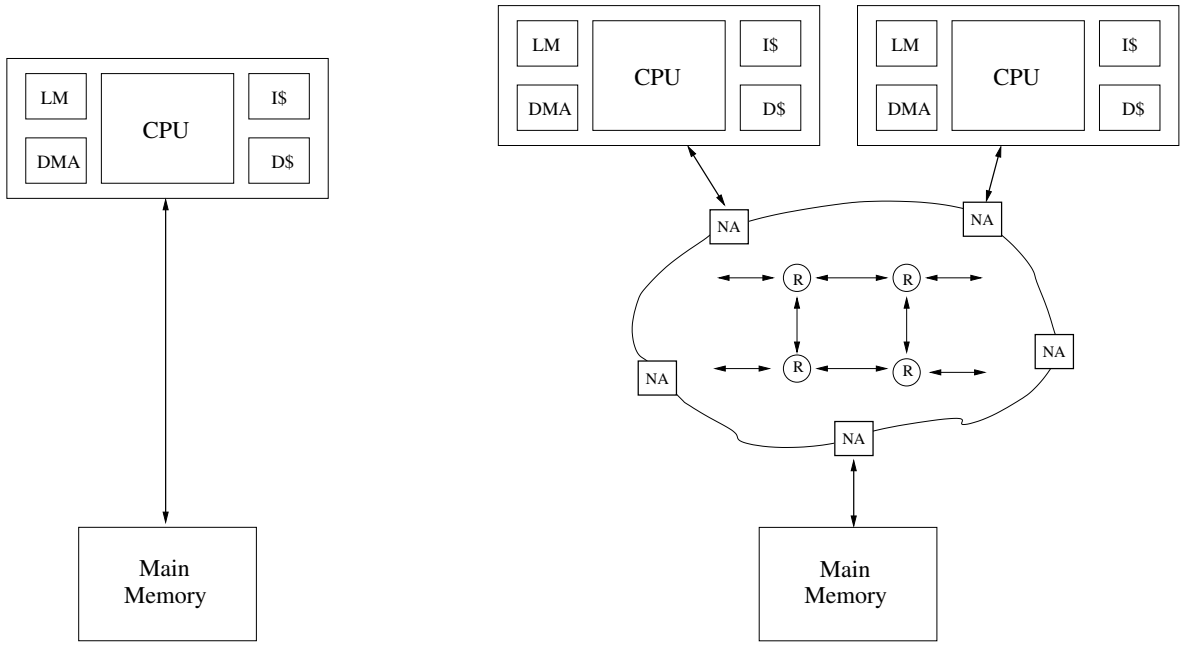
SoCBUS [12] and the network for the 4S project [13] are example NOCs that use pure circuit-switching. Connections, when established, don't share resources so real-time guarantees are easily achieved. However, this may result in low utilization of resources. TDM is another option of circuit-switching that has been applied in Æthereal [5] and Nostrum [10]. TDM implements virtual circuit-switching, which may result in better resource utilization. In order to implement TDM a common notion of time is necessary and this poses a challenge of preserving synchronization throughout the SoC design. Mesochronous links have been used to deal with this issue. Aelite [6], a lightweight version of Æthereal, employs the same mechanisms but at a lower hardware cost. The alternative to circuit-switching, non-blocking routers with rate control, can be seen in MANGO [1]. Packets are arbitrated locally at switches, where buffering and rate control is required in order to achieve guarantees.

The above mentioned NOCs will be further described and analyzed in the following chapter, and this will be based on the timing analysis model described below.

3.2 WCET analysis

WCET analysis techniques are well developed for single processor systems as the one shown in Figure 2(a). The analysis considers the specific trace of instructions that is executed, and the analysis takes into account the individual components of the platform, *i.e.* the processor, the cache and the main memory. For example, for deriving the timing of a load instruction, the analysis has to to consider the processor pipeline, the cache model (cache state, hit/miss), as well as the time to access main memory in case of a miss.

In a multi-processor platform, as shown in Figure 2(b), analyzing the WCET of a program executing on a processor node is in principle similar. The main difference is that accesses to the main memory



(a) Basic Processor-Memory platform.

(b) Multi-processor platform with network.

Figure 2: Platfrom

(or to resources in remote nodes) will experience some *additional latency resulting from packets traversing the NOC,* as well as some additional latency related to accessing the main memory (or the resource in the remote node). The latency involved in traversing the NOC depends on how the NOC itself is implemented and it may depend on interference from independent traffic in the NOC as well. The additional latency related to accessing the main memory consists of a possible wait before access to the shared memory is granted, followed by the time it takes to access the memory. These parameters depend on the design of the memory controller, as well as the timing characteristics of the memory device that is used.

This survey will attempt to analyze the additional latency that the NOC contributes to the execution time of the instructions of a program. In the direction of analyzing the timing of a network access, an effort will be made to identify the parameters affecting it. The nodes of a multi-processor platform are competing for network resources. The outcome of this competition depends on the execution history of all nodes involved. This makes the analysis very difficult and possibly practically intractable. However, the subject of this survey is time-predictable and analyzable systems, so, this behavior falls out of the scope of this survey and it will not be modeled in the analysis. Another interference to be noted is the local interference of instructions executed on the same processing node. This is a local issue that can be detected and handled by the processor in question, so it is expected to be incorporated in the execution analysis of the processor. Additionally, shared memory complicates the analysis of communication timing. It is a point where all traffic towards memory converges, so arbitration is needed. This may pose a bottleneck depending on the traffic and the arbitration scheme, severely affecting the execution time of a transaction. However, arbitration is part of the memory

controller, which is out of the scope of this survey and it will not be taken into consideration in following analysis.

The focus of this chapter is to formalize the way a NOC impacts the execution time of the instructions of a program, and the scope is limited to exactly this – estimating the latency of traversing the NOC when the processor accesses resources in remote nodes. The latency of actually accessing these shared resources is to be addressed in by documents specifically relating to memory work package in the T-CREST project.

3.3 Impact of NOC on the WCET

A program executed by a CPU may contain various sources of network accesses, each potentially affecting the execution time of the program. These can be cache misses originating from the instruction/data/stack caches accessing the main memory. Another possible source can be instructions bypassing the caches and accessing directly the main memory. The main memory can also be accessed by block transactions handled by DMAs, posing another source of network accesses. Finally, DMAs can initiate block transfers through the network to and from local memories of other processing nodes.

Based on the cases listed above, two types of network accesses can be distinguished in terms of the amount of data to be transferred. Single-word transactions, coming from load or store instructions bypassing caches, or block transactions, initiated either by DMAs or by cache misses. Concerning the type of transaction, *i.e.* read or write, four different cases can be identified:

1. Single-word read transactions.
2. Single-word write transactions.
3. Block-read transactions.
4. Block-write transactions.

The execution time of a transaction involving a network access, includes the time spent traversing the network (T_{NOC}) and the time spent accessing the remote memory (T_{mem}):

$$T_{transaction} = T_{NOC} + T_{mem} \tag{1}$$

As memory is a shared resource T_{mem} includes time for arbitration and waiting for access, followed by the memory access itself. As mentioned before this is out of the scope of this survey.

Depending on the case of transaction, the latency introduced by the NOC consists of two parts: (i) the time for the request to traverse the network, and (ii) the time for the response to traverse the network in the other direction. More specifically, all cases may experience some waiting time in order to get access to the network ($T_{wait,req}$), before, the transaction request is sent through the network (T_{req}). Finally, a reply should be send back, depending on the case, that would also require some waiting time for gaining access to the network ($T_{wait,reply}$) and some time to transfer the reply

through network (T_{reply}), back to the requesting node. In general, the network's contribution to the latency of a transaction is:

$$T_{NOC} = T_{wait,req} + T_{req} + T_{wait,reply} + T_{reply} \tag{2}$$

In the following, we take a closer look at each of the four cases, further explaining each timing parameter.

1. *Single-word read transactions* consist of a request for data and a reply. T_{req} represents the time of a read request, transferred from one processing node, through the network, to a local memory of a remote node or to the main memory. This is the transmission latency of one packet to be transmitted from one end-point of the network to another. T_{reply} corresponds to a single word of data as a response to a read request. It involves a point-to-point packet transmission latency, similar to the T_{req}. Parameters $T_{wait,req}$ and $T_{wait,reply}$ depend on the specific network design. Thus, the time interference from the network in this case, T_{sr}, is affected by the transmission latency of one packet and the waiting time to acquire access to the network.

$$T_{sr} = T_{wait,req} + T_{latency} + T_{wait,reply} + T_{latency} \tag{3}$$

2. *Single-word write transactions* consist of a write request along with the data to be written and optionally a completion acknowledgment. T_{req} is the time required for a write request, including the data, to be transferred from one processing node to a local memory of a remote node or to the main memory. As a single-word transaction, it corresponds to the point-to-point transmission latency of one packet. Depending on the write scheme, parameters $T_{wait,reply}$ and T_{reply} are considered or not. In the unacknowledged-write schemes, they are ignored. In the acknowledged-write schemes, parameter T_{reply} corresponds to a point-to-point packet transmission latency. Thus, the time interference from the network in the cases of acknowledged writes T_{swa} and unacknowledged writes T_{sw}, is affected by the transmission latency of one packet and the waiting time to acquire access to the network.

$$T_{swa} = T_{wait,req} + T_{latency} + T_{wait,reply} + T_{latency} = T_{sr} \tag{4}$$

$$T_{sw} = T_{wait,req} + T_{latency} \tag{5}$$

3. *Block read transactions* consist of a request for data and a reply. T_{req} is a read request and corresponds to a point-to-point packet transmission. T_{reply} represents the time for a block of data to be transferred. This depends on the number of packets in the block of data (n) and on the rate at which packets can be injected into the network (*throughput*), in the relation $n/throughput$. An additional point-to-point packet transmission latency is needed for the last packet of data to traverse the network. Thus, the time interference from the network in this case, T_{br}, is affected by the throughput, the size of the block of data and the waiting time to acquire access to the network.

$$T_{br} = T_{wait,req} + T_{latency} + T_{wait,reply} + n/throughput + T_{latency} \tag{6}$$

4. *Block write transactions* consist of a write request along with the data to be written and optionally a completion acknowledgment. T_{req} includes the time required to send the write request and the transmission of data to be written. This depends on the *throughput* and the size of the block of data (n packets) according to the relation $n/throughput$. An additional packet transmission should be considered for the last packet to reach the destination. The parameters $T_{wait,reply}$ and T_{reply} are considered depending on the write scheme. In the unacknowledged-write schemes, they are ignored. In the acknowledged-write schemes, parameter T_{reply} is one packet acknowledge signal that implies a packet transmission latency. Thus, the time interference from the network in the cases of acknowledged block writes T_{bwa} and unacknowledged block writes T_{bw}, is affected by the throughput, the size of block of data and the waiting time to acquire access to the network.

$$T_{bwa} = T_{wait,req} + n/throughput + T_{latency} + T_{wait,reply} + T_{latency} = T_{br} \tag{7}$$

$$T_{bw} = T_{wait,req} + n/throughput + T_{latency} \tag{8}$$

From the above analysis, it becomes clear that the timing contribution of a network to the execution time of a program, is reflected in three metrics: (i) The *latency* of one packet to travel through network from one point to another ($T_{latency}$), (ii) the *throughput* of the network, as the rate in which packets can be inserted in the network in one cycle, and (iii) the *waiting* time ($T_{wait,req}$/$T_{wait,reply}$) until access to the network can be acquired. This survey will attempt to provide analysis in terms of these metrics for some specific NOC designs and answer the question whether they are suitable for real-time systems.

Additionally, we will follow a common terminology for parameters affecting the timing of a NOC throughout the analysis. This might diverge from the one used in each NOC design but will lead to a better understanding of the NOCs described. We will use the terminology presented in the previous chapter, as well as some parameters listed below. The parameter list that follows provides a definition of each parameter but not a thorough description. Each parameter will be explained further in the corresponding design that it is used for. This list is intended to be used as a reference point to support the analysis in the following section.

The definitions that follow represent general design parameters of every NOC:

- N: number of router nodes in the network.
- h: number of hops (routers) in a path through the network.
- B: pipeline depth in routers.
- n: number of packets in a transaction.
- f: number of flits in a packet.
- b: number of bits in a packet.

The following list defines parameters used in specific NOC designs. More precise explanations will be given later:

l: lanes assigned to a virtual circuit.

L: lane width in bits.

t: containers assigned to a virtual circuit.

T: maximum length of a looped path.

p: time slots assigned to a virtual circuit.

P: period of time slots in a time schedule.

s: time slot duration in clock cycles.

C: number of virtual channels over a link.

4 Real-time NOCs

The real-time NOCs known from literature will be presented and their WCET will be analyzed following the approach described in the previous chapter.

4.1 SoCBUS

SoCBUS [12], [11], as mentioned earlier, is a pure circuit-switching network. The key features are presented in the following section.

4.1.1 Key Features

SoCBUS applies a packet-based procedure to set-up point-to-point connections, *i.e.* circuits, through the network. When the connection is established it owns all the resources along the path. For transferring data a 4-phase transaction is used. Initially, the connection is setup by a packet moving through the network and locking temporarily the requested resources, *i.e.* routers, links, (PCC - Packet Switched Circuit). Secondly, an acknowledge or negative acknowledge packet is sent back and the circuit is locked. At this point data can be transferred using the reserved resources. Finally, when data has arrived to the destination a packet is sent backwards, acknowledging the transfer and canceling the circuit at the same time, if needed.

SoCBUS is implemented as a two-dimensional mesh with five-ported routers, each connected to four neighboring routers and one processing core. Network Adapters are implemented as wrappers to the processing cores and they provide interfaces to the network. They are responsible for clock-domain crossings and for buffering packets for transmission. The routers are connected with bidirectional links and provide lines for data and control information. Mesochronous timing is used in the links to cover for clock skew and delay uncertainty in gates and wires.

During the setup procedure a minimum path length algorithm is used in the proposed design but other options are possible as well. At each hop, the setup packet is routed to one of the available routers based on the destination address in a round robin fashion. In case of no available routers, a failure of the setup procedure is caused. Resources are released and the setup procedure is repeated again in a later time. After a circuit setup is successfully achieved, data is forwarded from source to destination though the reserved routers and links without stalling and using full bandwidth of the connection.

A later version of SoCBUS, includes the sending of short messages in the setup process of packets. In this way short messages are accommodated by the transmission of the setup packet, avoiding the cost of setting up and tearing down of connections.

4.1.2 WCET properties

When a connection is established, it exclusively owns all the available resources it uses, (wires, routers, buffers), so real-time guarantees are trivially achieved. However, resources are not shared among connections, resulting in lower utilization. It is possible, that a required connection can not be setup due to lack of resources. Unsuccessful connection attempts lead to retrying and wasting

bandwidth. This dynamic behavior may cause fluctuations in the execution time of tasks in an unpredictable way, compromising the real-time properties. In the following analysis timing properties are evaluated assuming packets of one flit.

Wait time: For a packet to traverse the network a setup procedure has to be followed. For a packet to go though a router in order to setup a circuit, a minimum of four cycles are required per router. On the other hand, for the acknowledge signal to travel back one cycle per router is needed. Thus, for a path of h hops, the setup process requires $T_{wait,req} = 4 \cdot h + h$ cycles. For each transaction an extra h cycles need to be considered as overhead for tearing down the connection, when the transaction is done, but it can be done along with the reply message. The parameter $T_{wait,reply}$ is 0, since the circuit owns the resources and there is no waiting to gain access to the network for the reply message.

Latency: For a data packet to traverse a router, once a connection is established, one cycle per router is required. The total latency for one packet to travel through the end-to-end path is $T_{latency} = h$ cycles.

Throughput: The throughput of SoCBUS is 1 packet per cycle, *i.e.* b/c bps, as long as the connection is established. For a block transaction of n packets, the time required to transfer the block of data is n cycles.

Conclusion: Since SoCBUS has a high throughput it appears to be effective for block transactions, *i.e.* suitable for data streaming applications. For single packet transactions the overhead is large compared to the latency of the packet transmission. However, the success of the connection setup procedure is not guaranteed making SoCBUS unsuitable for real-time platforms.

4.2 4S project Network

The NOC developed by the University of Twente for the 4S project (Smart ChipS for Smart Surroundings) described in [13], [14], [8] is a pure circuit switching network. We present the key features below.

4.2.1 Key Features

The 4S project NOC is a reconfigurable circuit switching network developed to provide guaranteed services. It implements actual switching of wires in the routers, statically connecting input to output lines, similar to the switch boxes that are used in FPGA chips. Point-to-point circuits are formed in a static way by reserving portions of the link lines for this circuit. The circuit routes are predefined and the network is configured at start-up.

The key concept of this NOC is the organization of links in lanes. The links between routers are divided in fixed-width portions, called lanes. The number and the width of the lanes in a link is a design parameter and is fixed throughout the links of the system. Each point-to-point circuit is assigned a number of lanes and thereby claiming a portion of the available bandwidth.

A router in the system consists mainly of a crossbar that switches input lanes to output lanes. A configuration table is stored in the router to provide information about the connections between lanes. A crossbar configuration unit is connected to every router in order to configure the router with the

predefined connections. A data-converter component connects the router with a processing node, holding the role of the Network Adapter. It is responsible for converting the data packets of the processing node interface to lane-wide data blocks (*phits*) that fit to the width of the lanes and back again at the receiving end of the circuit. Additionally, it is responsible for adding a lane-width header to each packet of data. The outputs of the router are registered, so routers have a pipeline of 1 stage and width of lane-width.

For a packet to be inserted in the network, it is initially partitioned by the data-converter in phits of lane-width. In one cycle, a phit can go through an output of a router and through a lane to the next router. Lanes are wires without any pipelining. Therefore, for a packet to traverse a hop (router and link), as many cycles as phits are required.

End-to-end flow control is implemented using a credit-based scheme. To prevent buffer overflow among end-points of a circuit, an acknowledge signal is sent back for each lane in a separate, dedicated, line. Additionally, a window counter mechanism is used, which is reflecting the buffering capacity of each end-point of the circuit. The acknowledge signal is used as a form of credit to give information about the available space left in the destination point.

4.2.2 WCET properties

Circuit-switching networks bind resources to circuits providing guaranteed services. In this case, resources, as links, are not shared. Each connection uses specific lanes in a predefined route. So, few connections can be realized and utilization is potentially low. To support connections with additional lanes would require the addition of wires, increasing the hardware cost.

Wait time: After the configuration that is done at start-up, all the resources are dedicated to specified circuits. The result is no waiting time to get access to the network, *i.e.* parameters $T_{wait,req}$ and $T_{wait,reply}$ are 0.

Latency: As far as latency is concerned, the delay for a packet to traverse a router depends on the pipeline depth of the router (B). The time for a packet to go through a link depends on the packet size, the width of the lanes and the number of lanes that are assigned to the specific circuit. For a packet of size b bits, lane width L and l lanes owned by the circuit, $b/(L \cdot l)$ cycles are required to traverse a link. Due to serialization and immediate forwarding of phits in the routers, phits can proceed without waiting for the whole packet to arrive in the routers. The total end-to-end latency of a packet through a path of h hops is $T_{latency} = h \cdot B + b/(L \cdot l)$ cycles. The implementation proposed in [13] has $B = 1, b = 20$ and $L = 4$. Therefore the corresponding latency is $T_{latency} = h + 20/(4 \cdot l) = h + 5/l$ cycles.

Throughput: The number of packets that can be fed into the network per cycle depends on the phits a packet should be partitioned into, b/L phits. If l lanes are assigned to the circuit, throughput will be $(b \cdot l)/L$ bits per cycle. Thus, a block of n packets would require $(n \cdot L)/(b \cdot l)$ cycles. For the specific implementation proposed it would be $(n \cdot 4)/(20 \cdot l) = n/(5 \cdot l)$ cycles.

Conclusion: For the 4S project NOC, latency and throughput highly depend on the width of the lanes, which is a design property. Guarantees are trivially achieved and the network is effective for block transactions, *i.e.* streaming applications. However, resources are not shared between circuits.

4.3 Nostrum

Nostrum, [10], [9], is a virtual-circuit switching network. It employs Virtual Circuits (VCs) and mechanisms for time multiplexing. The key features are explained in the following section.

4.3.1 Key Features

Nostrum is a virtual-circuit switching network. It supports both GS and BE traffic. For BE, message packets are routed through the network and routing decisions are made dynamically in the routers. For GS, VCs are used with predefined routes that are stored in look-up tables in the routers. It uses the concepts of Temporally Disjointed Networks (TDN) and Looped Containers (LC) to facilitate both GS and BE.

The use of TDNs ensures the independence between traffic of different types (GS and BE) as well as between different VCs. TDNs can be seen as the different coloring of neighboring nodes. Two neighboring nodes cannot have the same color and packets travel from nodes of one color to nodes of different color. During a fixed duration time-slot packets move simultaneously from one router to another, passing from one color to another, such that packets residing in routers of different color in a point in time can never interfere. In this way, time division multiplexing is achieved. The maximum number of different TDNs that can exist in a network depends on the topology of the network and the number of pipeline stages of the router. That is because neighboring routers as well as neighboring pipeline stages of the same router should be of different color.

LC is a way of providing guaranteed bandwidth to a VC on behalf of the network. A VC in the network from a source point to a destination can be implemented as a looped path from source to destination and back. Looped Containers are traveling in this path, either loaded with a data packet or empty. The number of containers that are reserved for a VC reflects the bandwidth guarantees this VC has.

In combination TDNs and LC effectively implement a time division multiplexing mechanism. The maximum number of TDNs that exist in a network determine the maximum number of VCs that can share a link as well as the number of LC available for reservation. Consequently, containers are equivalent to a time-slot (in a TDM network) and TDNs as the repeated period of time-slots.

The number of maximum independent TDNs is a design feature, and corresponds to one period of time-slots in a TDM network. The VCs are statically decided at design time and are configured at start-up. However, the number of containers traveling through a VC can be changed at run-time, adjusting the offered bandwidth to the requirements. The maximum number of containers is limited by the number of TDNs.

Another feature of Nostrum is that the same looped path, can support multiple circuits by interleaving containers (assigned to these circuits), within the looped path. The VCs are still independent and provided with guarantees, similar to using different time-slots in a TDM schedule.

No control flow is implemented, as the source and destination points are considered able to handle any traffic, arriving within the predefined time multiplexing mechanism. However, buffers should be considered at the end points able to hold the data without overflowing, which is either expensive or requires some analysis to calculate the space needed.

The Network Adapter is divided in two components: A Network Interface (NI) component provides a set of services to the processing cores, like ordering of packets and (de)fragmentation of data to be transmitted. A Resource Network Interface (RNI), does the adaptation of the transaction protocol used by the cores to the set of services provided by the NI and vice-versa.

4.3.2 WCET properties

Resources are shared by VCs and for a VC that exists in the network, bandwidth is guaranteed. The number of VCs that exist in a network is decided statically at design time, depending on topology and other architectural characteristics. Bandwidth guarantees can be adjusted at run-time by issuing and removing packet containers in a VC. The round-trip time in a VC is a multiple of the maximum number of TDNs, T, going through this VC. The size of a packet is assumed to be one flit.

Wait time: The VCs are configured in the network at start-up and contain dedicated containers. The waiting time a packet experiences in a source end of a circuit is the time it waits in the source node until an empty packet container of the corresponding circuit arrives. In the worst-case, for a VC of T TDNs and one container assigned to the VC, a packet will wait for $T_{wait,req} = T$ cycles. If t containers are dedicated to this VC, the packet will wait in the worst-case for $T_{wait,req} = T - t$ cycles. t ranges in $[1, T]$, thus, the waiting time ranges from $T - 1$ to 0. The same holds for the waiting for the reply message, $T_{wait,reply}$. The proposed implementation suggests $T = 4$, therefore $T_{wait,req} = 4 - t$. The waiting time, also, depends on the allocation of containers to VCs. In the typical case, the containers allocated to a VC will not appear in immediate succession, but will be rather scattered around the schedule period. This indicates that $T_{wait,req} = T - t$ is a pessimistic but hard bound on the waiting time.

Latency: After a packet is loaded in a container, it travels through the path of this VC without further waiting. The latency it experiences in each router is B, considering B pipeline stages in every router. The point-to-point latency for a packet traveling through h hops is $T_{latency} = h \cdot B$ cycles. In the proposed implementation $B = 2$. Thus, $T_{latency} = 2 \cdot h$.

Throughput: The number of packets that can be inserted in the network per cycle depends on the number of containers owned by this VC. For each packet to be inserted, it has to wait for a container and then it uses the container to travel. For t containers in a VC, of T TDNs, it is implied that in a period of T time slots t packets can be injected in the network. Thus, this VC has a throughput of t/T packets per cycle. Since t is in the range $[1, T]$, throughput ranges in $[1/T, 1]$ packets per cycle. For a block transaction of n packets, $(n \cdot T)/t$ cycles are required. The proposed implementation has time $(4 \cdot n)/t$ cycles.

Conclusion: Nostrum is a TDM-based network and requires synchronization operation. The transfer latency of a single packet is proportional to the number of hops of the path and some waiting time in the beginning of transmission. This waiting time is a fraction of the TDM period. The throughput is inversely proportional to the TDM period. The bandwidth guarantees offered are a fraction of the total bandwidth. The timing is affected by topology factors of the entire network, and the entire traffic is inflicted in the timing behavior. This leads to a predefined and predictable timing behavior well suited for real-time systems.

4.4 Æthereal/Aelite

Æthereal and Aelite, as described in [5], [4], [7] and [6], are based on the TDM approach.

4.4.1 Key Features

Æthereal is a TDM circuit switching network that offers BE as well as GS. Aelite is a version of Æthereal that provides only GS with very lightweight routers.

Time slotting is a fundamental concept of Æthereal and Aelite. It is a fixed-duration slot of time in which a block of data can be forwarded though a router. It is the same duration for every router in the network and all routers are synchronized, *i.e.* every router in one time slot can forward one block of data to each output. The overall slot schedule per output per router is decided at design-time and the network is configured at run-time. A time-slot schedule is repeated periodically and this period is the same for the whole network.

Virtual circuits are defined as paths from one node to another through a number of hops. Guarantees are provided to virtual circuits by assigning them a number of time slots within a period. A block of data at a source node waits for a dedicated time slot and when that arrives the packet is forwarded from one hop to another in sequential time slots, without blocking or waiting.

Routers hold the slot tables with the routing information for each of their outputs. The NAs connect the routers with the processing cores implementing any protocol translation from read/write transactions to streams of packets and their segmentation in flits of data to be transferred. They are also responsible for buffering packets of data and for implementing flow control.

Æthereal employs a credit-based flow control. Buffers in the NAs hold packets of data in queues and credits are being sent from destination back to source when there is enough space to hold the packet and avoid buffer overflow.

The part of the router responsible for the BE traffic, implements wormhole-routing with source-routing and link level flow control.

Aelite, follows the same principle of time slots but provides only GS. In this version, the slot tables are held in the NAs instead of the routers and a source-routing scheme is applied. In this way the routers become very simple and low-cost. In addition, to avoid the need for global synchronization, mesochronous links are used, which are implemented as simple FIFOs. They cover for clock skew and phase shifting but they imply an additional hardware cost.

Both in Æthereal and Aelite the block of data that can be transmitted in a time-slot is equal to the number of flits in a packet, as well as the number of pipeline stages in a router. This indicates that in one time-slot a whole packet can be transmitted through a router.

4.4.2 WCET properties

Æthereal and Aelite are based on the concept of periods of time slots and schedules of those time slots. This implies that a period of a schedule is followed and repeated. For a virtual circuit, considering a period of P time slots, p number of slots can be assigned to the circuit. The execution

time properties, in addition, depend on design parameters like the pipeline depth of a router (B), the duration of a time slot in cycles (s) and the number of flits constituting a packet (f). In the actual implementations of both Æthereal and Aelite, the above parameters are chosen to be identical, with a value of 3, such that a packet can go through a router in one time slot.

Wait time: In order to transmit a packet of data through a virtual circuit, the packet needs to wait for the corresponding slot to arrive. If this virtual circuit owns p time slots, the waiting time in the worst-case would be $T_{wait,req} = (P - p) \cdot s$ cycles, where p ranges in $[1, P]$. The waiting time is affected by the position of the different time-slots assigned to a VC within a period. $T_{wait,req} = (P - p) \cdot s$ is a pessimistic bound, as the p time-slots, in the typical case, will be chosen to be spread throughout the period schedule. In the implemented design it is $T_{wait,req} = 3 \cdot (P - p)$ cycles. The same holds for $T_{wait,reply}$.

Latency: The latency of one packet in each router is the router pipeline depth, B, which is equal to duration of time slots in cycles, s, and to the number of flits of the packet, f. No waiting occurs in the routers, since each flit is forwarded immediately. Therefore, the latency from source to destination for h hops is $T_{latency} = h \cdot B + f$ cycles. For the implemented design it is $T_{latency} = 3 \cdot h + 3$ cycles.

Throughput: In terms of throughput, the schedule is organized in periods of P time slots and the throughput of a circuit depends on the time-slots it owns. With p time-slots assigned to a VC, p packets can be inserted in the network in one period. Thus, a circuit can reach the total throughput of $p/(P \cdot s)$ packets per cycle. In the specific implementation throughput is $p/(3 \cdot P)$ cycles. For a block transaction of n packets $(p \cdot n)/(P \cdot s)$ cycles are required. The implementation specific time would be $(p \cdot n)/(3 \cdot P)$ cycles.

Conclusion: Æthereal and Aelite offer transmission packet latency proportional to the hops of the traversed path. Some waiting time has to be considered in the beginning of transmission which depends on the TDM period and . As TDM networks, they are based greatly in the notion of time. Throughput is inversely proportional to the TDM period and depend on the reserved time-slots. A universal view of the network should be considered in all the timing parameters, for example TDM period and time-slot schedule is a decision affected by the whole network traffic. On the other hand, the universal view and the well-defined behavior offer great predictability making TDM networks highly suitable for real-time systems.

4.5 MANGO

MANGO, [1], [2], [3], is an asynchronous network, employing the concept of virtual channels (VCs) with rate control at routers and provides BE traffic support as well as GS.

4.5.1 Key Features

MANGO implements end-to-end circuits in a way which is radically different from the TDM networks described above. Another characteristic feature of MANGO is that it is implemented using asynchronous circuits. A MANGO router consists of a BE router and a GS router and the two types of traffic share the links in the NOC. In the following we focus on the GS part of the router.

A link in the NOC may be shared by multiple end-to-end circuits, and the sharing is controlled by local arbitration in the output ports of the individual routers. This effectively divides each link into a number of so-called virtual channels – one for each circuits using the link. Each output port in a router has a separate physical queue for each of the virtual channels sharing the link. This has implications on the size of the crossbar switch in a router as well as on the amount of buffer registers needed. In a 2D-mesh NOC with 8 virtual channels per link, the crossbar in a router has 5 input ports and 5x8 = 40 output ports, each with its own virtual channel FIFO-buffer.

The input-to-output switching performed by the crossbars in the routers is set up statically at initialization time, and a circuit is thus a sequence of output-port virtual-channel FIFO-buffers connected by crossbar switches. Credit based flow control between virtual channel FIFO's is used to prevent flits from stalling the shared links.

Admission control and arbitration modules in the output ports of the routers determines the bandwidth and latency of the connection (on a hop by hop basis). A fair-share scheme provides equal guarantees to all virtual channels sharing the link. An alternative scheme is the priority-based scheduling (ALG, [2]), which grants link access first to VCs with higher priority.

NA are used to connect the routers to the processing cores. They implement synchronization between the network and the cores and adaptation from protocol transactions in cores to packetization of data.

4.5.2 WCET properties

MANGO manages to facilitate modular design and GALS architecture resolving the timing synchronization and the clock distribution throughout the chip, while providing real-time guarantees. It doesn't require end-to-end flow control since more fine-grained arbitration is done in the routers leveraging the weight from NA. However, the routers become more costly in area since they need to support buffering for each VC. The timing properties depend on the number of VCs that share a link (C) and the link arbitration scheme. For a specific connection the timing is formed as follows, considering a fair-share arbitration scheme.

Wait time: There is no initial waiting time at the source end-point, since the virtual circuits are statically configured. There is some waiting for arbitration in each link which will be considered in the transmission latency in each hop. Thus, parameters $T_{wait,req}$ and $T_{wait,reply}$ are 0.

Latency: The packets in MANGO consist of a number of flits. The latency of a flit to traverse a hop (router and link) depends on the pipeline depth of the router(B) and the link access arbitration scheme. Assuming a fair-share arbitration scheme, a flit requires C cycles waiting time to gain access to the link in the worst-case. That is the number of VCs, *i.e.* output buffers, that share the link. Consequently, one-hop latency is $C + B$ cycles. Flits can be forwarded through the next hop without waiting for the whole packet to arrive. Thus, the latency of one packet of f flits traversing a path of h hops is $T_{latency} = (h + f) \cdot (C + B)$ cycles.

Throughput: Every flit has to go through link access arbitration. A packet of f flits will be successfully inserted in the network in $f \cdot C$ cycles. The packets that can be injected in the network in one cycle are $1/(f \cdot C)$ packets per cycle. For a block transfer of n packets $n \cdot f \cdot C$ cycles are required.

Conclusion: Transmission latency of a single packet in MANGO depends on the number of VCs that share this link. In addition, there is no waiting time in the beginning of transmission, as traffic is

controlled locally. Throughput is inversely proportional to the VCs using the link. The timing parameters in MANGO are affected by local factors of traffic and decisions are taken locally, as opposed to TDM networks that rely on a global schedule. This provides high flexibility and adjustment to the traffic needs. The cost of this comes in high area overhead, as each VC is buffered separately in every router.

5 Conclusion

A number of real-time NOC designs were described in this survey. Initially, the basic concepts of NOCs were introduced. An effort to identify the way a NOC affects the WCET of a task was attempted, suggesting a model for the WCET analysis for NOCs. This involves the evaluating of timing parameters for a NOC design, depending on the design features of each NOC. The key concepts and the timing properties characterizing a series of state-of-the-art NOC designs were presented, in order to give insight on their performance and their effectiveness, in terms of a time-predictable platform.

We presented and analyzed the timing properties of a number of real-time NOCs, giving insight on the performance of each network. The performance metrics described were the initial waiting time, the transport latency of a single packet and the throughput of the network. They appear to vary a lot, as they depend on design parameters of each specific network. Formal equations were formed for the above metrics based on the design parameters. However, they can not be used for comparison between the networks, as they include various design-specific parameters. Moreover, each network implementation operates on different frequencies, not allowing a direct comparison of metrics.

In spite of the variation among the designs, the performance metrics described show similarities based on their key concept of operation. They fall into three main categories. They are either pure-circuit switching networks(SoCBUS, S4 project NOC), TDM networks(Nostrum, Æthereal/Aelite) or rate-controlled(MANGO). The pure circuit-switching networks present an initial waiting time in the source point that either depends on the hop count of the path (SoCBUS) or is zero (4S project NOC). The latency of one hop depends on the pipeline stages of the hop and is a fraction of the link bandwidth. Throughput is a fraction of the total bandwidth. TDM networks have an initial waiting time until the arrival of the allocated time-slot, determined by the global schedule. The hop latency is fixed and depends on the hop count. Throughput is a fraction of the available bandwidth. Rate-controlled networks don't require initial waiting time but spend waiting time in each hop for arbitration, determined by the communication load sharing this link. Throughput is a fraction of the available bandwidth, similar as in the other cases.

In general, the presented networks based on their overall functionality, they exhibit some common characteristics. Pure circuit-switching NOCs present low utilization of resources. Resources are not shared, wasting available bandwidth. Low utilization can possibly lead to some connections failing to establish at run-time or not supported, while resources are not being effectively used. They are effective at supporting specific type of communication (streaming) and lack in flexibility.

On the other hand, TDM networks apply precise allocation of resources in every point in time, making them predictable by construction. Resources are effectively used. The allocation takes into account the overall communication needs and the traffic load of the whole network. That implies that the performance of TDM networks depends on the allocation scheme and the scheduling algorithms used, in addition to the network characteristics and traffic load. Moreover, TDM NOCs need a universal timing reference in order to synchronize time-slots throughout the system. Mesochronous links deal with synchronization but add to the area cost.

Rate controlled networks present great flexibility on the specific requirements and the current traffic load. However, the arbitration and control results in high area overhead.

In conclusion, all of the described NOCs have some commonalities but at the same time they also have some differences that makes it difficult to make meaningful quantitative comparisons of performance and cost. All the networks described provide virtual end-to-end connections and some form of sharing (in the temporal or spatial domains) of resources and bandwidth. The latency that the NOC adds to a memory read or write transaction targeting remote node can be calculated from the number of routers and links in the connection as well as relatively simple latency and throughput parameters which can be derived for the different types of networks.

In terms of simplicity, speed and area, a TDM approach as taken by Aelite seems to be attractive, and the work on preparing this survey has consolidated the original choice of basing the T-CREST NOC on these principles.

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)*, pages 1226–1231. IEEE Computer Society Press, 2005.

[2] T. Bjerregaard and J. Sparsø. A Scheduling Discipline for Latency and Bandwidth Guarantees in Asynchronous Network-on-chip. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 34–43. IEEE Computer Society Press, 2005. (Best paper award).

[3] T. Bjerregaard and J. Sparsø. Implementation of Guaranteed Services in the MANGO Clockless Network-on-Chip. *IEE Proceedings: Computing and Digital Techniques*, 153(4):217–229, 2006.

[4] Kees Goossens and Andreas Hansson. The AEthereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proc. Design Automation Conference (DAC)*, pages 306–311, June 2010.

[5] Kees Goossens *et al.* The Æthereal network on chip: Concepts, architectures, and implementations. 22(5), 2005.

[6] Andreas Hansson and Kees Goossens. *On-Chip Interconnect with aelite: Composable and Predictable Systems*. Embedded Systems Series. Springer, November 2010.

[7] Andreas Hansson *et al.* aelite: A flit-synchronous network on chip with composable and predictable services. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2009.

[8] N. Kavaldjiev, G.J.M. Smit, P.G. Jansen, and P.T. Wolkotte. A virtual channel network-on-chip for gt and be traffic. In *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, volume 00, page 6 pp., march 2006.

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

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

[11] S. Sathe, D. Wiklund, and D. Liu. Design of a guaranteed throughput router for on-chip networks. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, pages 25 – 28, nov. 2004.

[12] Daniel Wiklund and Dake Liu. SoCBUS: Switched network on chip for hard real time embedded systems. *Parallel and Distributed Processing Symposium, International*, 0:78a, 2003.

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

[14] P.T. Wolkotte, G.J.M. Smit, N. Kavaldjiev, J.E. Becker, and J. Becker. Energy model of networks-on-chip and a bus. In *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, pages 82 –85, nov. 2005.