


T-CREST

TIME-PREDICTABLE MULTI-CORE ARCHITECTURE
FOR EMBEDDED SYSTEMS

**Project Number 288008**

# D 3.5 Report on Impact of Asynchronicity on Predictability of the NOC

**Version 1.0**
**23 September 2013**
**Final**

**Public Distribution**

**Technical University of Denmark**

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

# Contents

# Document Control

| Version | Status | Date |
|---|---|---|
| 0.1 | First draft | 27 August 2013 |
| 0.2 | Self-contained vrt. D 3.4 | 30 August 2013 |
| 0.3 | Draft sent to T-CREST partners | 9 September 2013 |
| 1.0 | Final version | 23 September 2013 |

# Executive Summary

This document is deliverable *D 3.5 Report on Impact of Asynchronicity on Predictability of the NOC* of work package 3 of the T-CREST project, due 24 months after project start as stated in the Description of Work.

The hardware implementation of the message-passing NOC is described in deliverable D 3.4 that is also due at month 24. The scope of this deliverable, D 3.5, is the timing behavior of the NOC as seen from an application programmer's point of view.

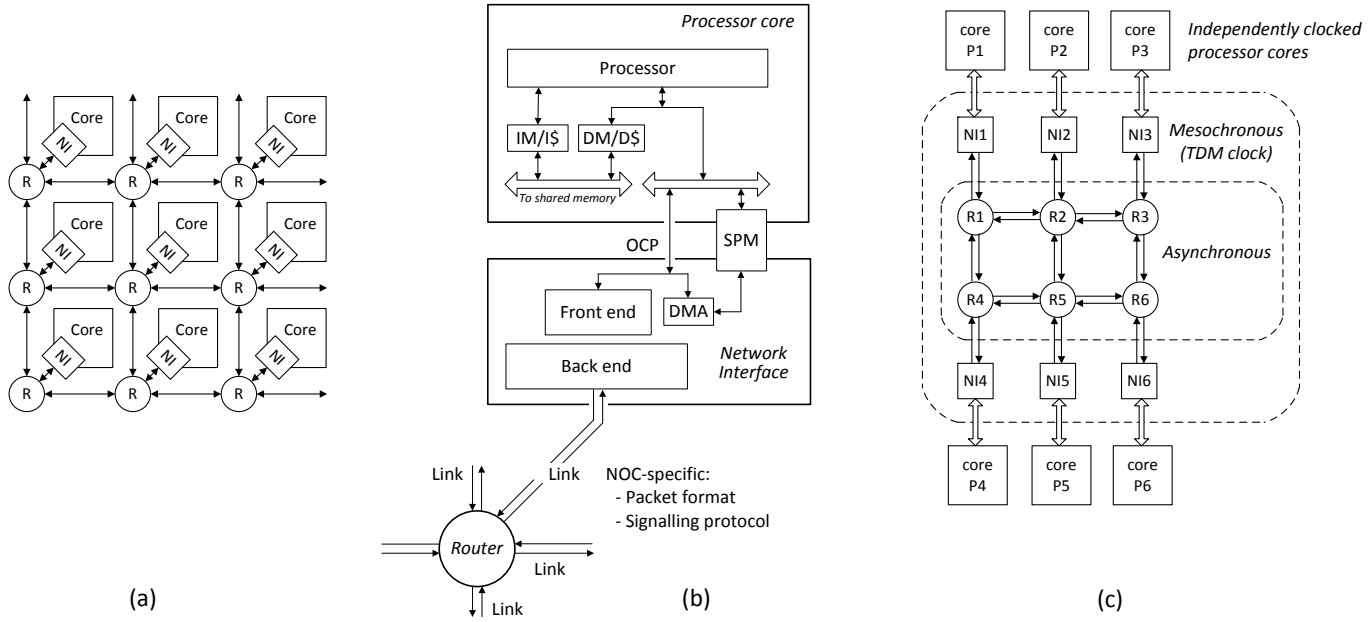


Figure 1: (a) An example T-CREST multi-processor platform using a 2 D mesh NOC topology. (b) Details of a processor core and a network interface (NI). (c) Timing architecture of the T-CREST NOC: example with 6 processor cores connected to a NOC consisting of 6 routers and 6 NIs.

# 1 Introduction

The architecture and implementation of the message passing NOC is addressed in project deliverable D 3.4 [2] and scientific publications [5, 3].

Figure 1 shows the architecture of the T-CREST platform. The figure focusses on the message passing NOC and shows its most important features. The figure omits the memory controller and the processor to memory controller communication that is covered in WP4 of the T-CREST project.

Figure 1(a) shows an example platform where a set of processor cores are connected by the NOC that forms a 2D-mesh topology. The NOC uses packet switched routers and links. Processor cores, each with some amount of private memory, are connected to the structure of routers and links using network interfaces (NIs). In general, the role of a NI is to bridge between the packet switching interfaces of the routers and links and the read/write transaction interfaces of the processors cores.

The T-CREST NI use a micro-architecture where the DMA controllers, that are normally part of the processor cores, are integrated with the TDM-scheduling mechanism in the NOC. This novel micro-architecture results in a small and efficient implementation, that avoids resources for buffering and flow control [5]. The routers and links in the NOC are implemented as self-timed asynchronous circuits [2, 3]. The NIs and the processor cores are conventional clocked synchronous circuits. Figures 1(b) and 1(c) will be explained in the following sections 2 and 3.

From an application programmers point of view the NOC offers asynchronous message passing between processor cores. The communication channels that are used for this message passing are implemented as (virtual) end-to-end circuits, and the mechanism used to implement these virtual

end-to-end channels is *statically-scheduled time division multiplexing (TDM)* [4]. This is a straightforward and easy to analyze approach to obtain time predictability. In addition, it avoids run-time arbitration and data buffering in the routers in the NOC and this makes the routers very simple and efficient.

Intuitively the use of asynchronous routers could potentially complicate the time-predictability of the NOC, and thereby also complicate worst-case execution time (WCET) analysis of an application executing on the platform. For this reason the T-CREST Document of Work includes this deliverable, D 3.5.

As will be clear from the following, the fact that the routers are asynchronous is *completely invisible* at the interface between a processor core and its NI. And as all NIs are synchronous circuits operating at the same clock frequency (with only some bounded amount of clock skew) all timing analysis relate to the same clock signal; a property that greatly simplifies WCET analysis.

The rest of this document is organized as follows: Section 2 presents the architecture and implementation of the NOC – as seen in Figure 1(b) – in more detail. Section 3 elaborate on the timing aspects of the design, as illustrated in Figure 1(c). Section 4 provides the information necessary for an application programmer to assess the NOC's contributions to the worst-case execution time (WCET) of an application. Finally Section 6 concludes the report. For a review of the requirements that are relevant for the NOC developed in WP3, the reader is referred to deliverable D 3.4.

# 2 Architecture and Implementation of the NOC

Figure 1(b) shows details of a NI and a processor core. A processor core consists of the processor itself and some (private) instruction and data caches as well as a private scratch-pad memory (SPM). The transfer of a message across a communication channel (virtual circuit) is driven by a DMA controller associated with the specific channel. The DMA-controller is in the source end, and it is capable of transferring a block of data from the local SPM and into the SPM of the processor at the destination end of the communication channel. The virtual end-to-end circuits that implement the channels are defined during an initialization phase where the NOC is configured.

A unique feature of the T-CREST NOC design is that the SPMs are dual ported and that the DMA controllers are included in the NIs as shown in Figure 1(b). This allowed us to merge the DMA controllers and the TDM scheduling such that data is transferred from the source SPM, across the NOC, and into the target SPM, without any buffering or flow control at all. Compared to other NOC designs that provide similar functionality, this more than halves the size if the NI implementation, and it also greatly simplifies timing analysis. The use of dual ported SPMs allows program execution on the processor cores to overlap with the DMA transfers. It is also an efficient way to implement clock domain crossing.

The NOC uses source routing and packet-switching. A packet consists of three phits; a header phit and two payload phits. A phit is 32 bits of data and a 3 bit tag (as explained in [2, 3]). The header phit consists of a bit pattern representing the route from source to destination and the address in the target SPM where the payload data is to be written. In every clock cycle a NI transmits and receives one phit from the packet switched structure of routers and links. In time-slots where no communication is

scheduled, the NI transmits and receives so-called void-phits. Further details are explained in D 3.4 [2]. The essence of this is that a block of data is transferred in atoms of two 32 bit words, and that it takes 3 clock cycles for a NI to send and/or receive a packet with this payload.

Before any message passing can be done it is necessary to initialize the slot tables and the DMA tables in the NIs. This is part of the initialization of the platform that is performed before the execution of the application starts, and it is not included in the analysis presented below.

# 3 Timing Perspective on the Implementation of the NOC

The requirement *N-3-043 Timing Organisation* listed in deliverable D 1.1 states that the NOC shall support a globally-asynchronous locally-synchronous (GALS) style design. This requirement is indeed satisfied by the NOC architecture described above and illustrated in Figure 1. As seen in Figure 1(a) the packet switched structure of routers and links that spans the whole platform is implemented using self-timed asynchronous routers and links. This avoids clock-distribution and clock skew problems at the global level, and clocked synchronous operation is confined to within a single NI and/or a single processor core. Figure 1(c) emphasizes this view: an asynchronous packet-switched network to which a number of NI-processor pairs are connected.

The NIs are conventional clocked synchronous circuits. The clock signals used in the different NIs originate from the same source/oscillator, but the architecture allow some clock skew (possibly varying but bounded) among the NIs. The terms *mesochronous*[1] or *multi-synchronous*[2] denote this situation. As a result of this, all NIs operate at the same rate. This allows a straightforward implementation of the statically-scheduled time-division-multiplexed transmission of data across the NOC. The clock-skew between the NIs is absorbed by the asynchronous routers and links.

The processor cores are also conventional clocked synchronous circuits. The processor cores may use independent clocks. This allows the use of frequency scaling to save power. It also allows the use of different processors with different speeds, should this be needed. To support this, a clock-domain crossing module, not shown in Figure 1(b) must be added at the interface marked “OCP” between a processor core and the NI. This clock domain crossing will add latency to the read/write-transactions used to set up the DMA transfers. For the actual data that is transmitted across the NOC the use of dual-ported SMPs provides clock-domain-crossing without any additional latency and without any additional hardware.

Altogether the architecture presented above is a so-called globally-synchronous locally-asynchronous (GALS) design that is robust to timing variability at the global level.

The timing organization described here and illustrated in Figure 1(c) is the most general and flexible instance of the T-CTRET platform that one can think of. A range of simpler timing organizations are possible. Below we list and describe the most obvious implementations. For completeness the list repeats the timing-organization just described. In all four implementations the clock signals used in the NIs originate from the same source.

[1] same frequency, fixed skew
[2] same frequency, varying but bounded skew

Two implementations use a self-timed asynchronous NOC and offer some tolerance towards clock skew at the global level, and these are the target of the T-CREST project, i.e., the design delivered in D 3.3 and documented in D 3.4 [2]:

1. A globally-asynchronous locally-synchronous (GALS) implementation where each processor core operate using its own independent clock, where all the NIs are clocked by the same clock signal, possibly with some bounded but unknown skew (i.e., the NIs are multi-synchronous), and where the routers are implemented as self-timed asynchronous circuits. This implementation, illustrated in figure 1(b), allows independent frequency scaling in the processor cores and/or the use of different processors with different clock frequencies. In addition it offers tolerance towards clock skew at the global level.

2. A globally multi-synchronous implementation where each processor core use the same clock as the NI to which it is attached and where the routers are implemented as self-timed asynchronous circuits. This implementation tolerates a bounded amount of clock skew between processor-NI pairs.

A third implementation that could be relevant for testing purposes, but not for a practical implementation, in which all processor cores and NIs are clocked synchronously:

3. An implementation where all the processors and NIs operate synchronously and where the routers are implemented as self-timed asynchronous circuits.

Finally we mention that for FPGA prototyping, for software development, and perhaps for implementation of small systems, the following is relevant:

4. A globally-synchronous implementation where the same global clock is used in all the processors as well as in all the routers and NIs that constitute the NOC. It is not a deliverable of the T-CREST project, but for completeness we mention that the simulation model delivered in D 3.2 [1] was described as RTL-level VHDL-code. Consequently it is possible to synthesize a globally-synchronous implementation of the T-CREST platform, should this be desired.

From the point of view of WCET analysis there is very little difference between the above mentioned four implementations – their timing characteristics in this context relate to the clock used in the NIs and in all four cases the clock signals used in the NIs originate from a single clock oscillator.

# 4 Quantifying the NOC's Contribution to the Execution Time of an Application

By the term "application" we understand a set of tasks that execute on a set of processors, and that communicate using asynchronous message passing as supported by the DMA-driven block-writes implemented by the NOC.

In the context of a single task executing on a single processor the situation is simple: some load and store instructions target the NI where they initiate or monitor progress of the DMA-driven block transfers. To estimate the WCET of the task, all that needs to be known about the NOC, is the number of processor clock cycles it takes to execute a load instruction and a store instruction that targets the NI.

In the context of the whole application the situation is more challenging. In this report we provide the information necessary to estimate the latency of a block transfer, from the transfer is initiated by the processor core setting up a DMA-controller in its NI, and until all data has been transferred across the NOC and is available in the SPM in the target processor core. This can be precisely estimated from characteristics of the implementation of the NOC and from the application-specific TDM schedule that drives the transmission of data across the NOC. To what extent this latency affects the overall execution time of an application, consisting of a set of communicating tasks, is more difficult to assess. It depends on how the application is programmed and to what extent the application programmer has been able to hide the communication latency, and it is therefore outside the scope of this document where focus is on the NOC. As the clock signals used in all NIs originate from the same clock oscillator, it follows that all DMA-transfers in the entire platform take place at the same rate. This simplifies the WCET analysis and it is an additional benefit of the micro-architecture of the NI.

Below we elaborate on these two perspectives on the NOC's contribution to WCET-analysis:

**WCET-analysis of a single task executing on a single processor core:** The application code executing on a processor includes some load and store instructions that set up DMA-transfers and monitor the status of ongoing DMA-transfers. These load and store instructions cause single-word read and single-word write transactions across the interface marked OCP between the processor core and the NI, see figure 1(c). With the specific NI-design described in D 3.4, each read or write transaction has a latency of 1 or 2 NI clock cycles, i.e., a worst case of 2 cycles. This assumes that the processor core and the NI are synchronous, and corresponds to the second timing organization described in the previous section. If the processor cores use independent clock signals – the first timing organization described in the previous section – the worst case latency of a clock domain crossing has to be added to each read or write transaction. Assuming the frequency of the clock signals used in a processor core and a NI differ by at most a factor of two, a safe worst-case estimate is that a single-word write-transaction and a clock domain crossing has a latency of 10 processor clock cycles and that a single-word read-transaction and a clock-domain crossing has a latency of 15 processor clock cycles. Precise figures depend on the ratio between the frequency of the NI-clock and the processor-clock as well as the exact design of the clock-domain crossing module.

**Latency of a DMA-driven block write:** This is the time from a DMA-transfer is set up and until the block of data has been transferred from the local SPM, across the NOC and into the SPM of the target processor core. This accounts for most of the traffic between the processor and the NOC. As the DMA-driven block-transfer happens in parallel with program execution on the processor core, it is a property of the way the application is programmed, to what extent this latency is visible and influences the WCET of the application. If the application programmer implements double-buffering, that overlaps computation and communication, it may even be possible to hide the latency of the block transfers. The degree to which this is possible depends

on the whether or not it is possible to reserve the necessary extra slots in the TDM-schedule without affecting the timing of the entire system.

Once the DMA-transfer has been set up, the latency of transferring a block of data from the local SPM, across the NOC and into the SPM of a remote processor core can be calculated as follows:

$$L = T_{Wait} + T_{Send} + T_{NOC}$$

$T_{Wait}$ is the worst-case wait for the first available slot in the TDM-schedule reserved for the DMA-transfer. A worst-case estimate of this is the maximum time separation between two slots reserved for the same channel within a sequence of TDM-schedule periods.

$T_{Send}$ is the time it takes to send the entire block. Assuming a block of some size, this can be estimated as the block size (measured in 32 bit words) multiplied by the period of the TDM schedule and divided by the number of 32 bit words transmitted during one period of the schedule. This number is 2 (words per packet) times the number of slots reserved in a schedule period.

$T_{NOC}$ is the time it takes the last data-phit of the last packet to traverse the NOC. In the current implementation of the asynchronous NOC the links are not pipelined and the pipeline depth of the routers is two phits per router. A worst-case estimate of $T_{NOC}$ is therefore two times the number of routers in the longest path between any two processor cores. In an $N \times N$ 2D bi-torus this is $2 \cdot \lceil \frac{N-1}{2} \rceil \cdot \lceil \frac{N-1}{2} \rceil$ NI-clock-cycles.

# 5 Requirements

The requirements listed in deliverable D1.1 that are relevant for the NOC are listed and commented on in deliverable D 3.4 and are not repeated here.

# 6 Conclusion

This report made several contributions: It showed that the use of asynchronous routers has no impact on the time-predictability of the NOC, and it provided the information necessary for an application programmer to assess the NOC's contributions to the worst-case execution time (WCET) of an application.

One perspective on the latter considers a single task that execute on a single processor core. Here the time base is the clock used in the processor core and the contribution of the NOC is the latency of the read and write transactions caused by the load and store instructions that target the NI and that set up the DMA transfers. This latency may include a clock domain crossing.

Another perspective considers an application consisting of a set of tasks executing on a set of processors that communicate using asynchronous message passing as provided by the DMA-driven block-transfers implemented by the NOC. Here the time base is the clock used in the NIs, and the report provided the information necessary to calculate the latency of a block transfer. To what extent this affects the WCET of an application depends on how the application is programmed.

# References

[1] T-CREST deliverable D3.2. Simulation model of the self-timed NOC. http://www.t-crest.org/page/results.

[2] T-CREST deliverable D3.4. Report documenting hardware implementation of the NOC. http://www.t-crest.org/page/results.

[3] E. Kasapaki, J. Sparsø, R.B. Sørensen, and K.W.G. Goossens. Router Designs for an Asynchronous Time-Division-Multiplexed Network-on-Chip. In *Proc. of Euromicro Conference on Digital System Design (DSD)*, pages ??–??, September 2013.

[4] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems. In *Proc. IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 152–160. IEEE Computer Society Press, May 2012.

[5] J. Sparsø, E. Kasapaki, and M. Schoeberl. An Area-efficient Network Interface for a TDM-based Network-on-Chip. In *Proc. Design Automation and Test in Europe (DATE)*, pages 1044–1047, 2013.