Microprocessors and Microsystems xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A scalable, non-interfering, synthesizable Network-on-Chip monitor – Extended version

Antti Alhonen *, Erno Salminen, Lasse Lehtonen, Timo D. Hämäläinen

Tampere University of Technology, Department of Computer Systems, P.O. Box 553, FIN-33101 Tampere, Finland

A R T I C L E I N F O

Article history:
Available online xxxx

Keywords:
Network-on-Chip
Hardware monitoring
System-on-Chip
Performance analysis
Real-time trace

A B S T R A C T

Today's Multi-Processor System-on-Chips incorporate Network-on-Chips to interconnect multiple processors, memories, and accelerators. We present a freely available toolset to monitor and analyze these networks. Internal signals are pre-analyzed on FPGA without interfering the system. Host PC carries out further analysis with post-processing algorithms and an intuitive graphical interface. Traces of end-to-end communication can be approximated from mere link statistics, average error being 10%. In a case study of MPEG-4 encoder ran at 25 MHz, we compared link utilizations and stall cycles by any time window from 500 clock cycles to the whole running time. Area overhead for monitoring was 5%.

1. Introduction

System-on-Chip (SoC) integrates processing elements (PEs), memories, and external interfaces into a single chip. Network-on-Chip (NoC) is a paradigm of transferring data between these resources inside a chip [1–3]. As a SoC grows larger, selecting and further shaping the appropriate communication network between the elements becomes more and more crucial [4]. NoCs typically consist of *routers* which are connected to each other with *links*. This way, long wires and combinational paths can be split to smaller segments compared to traditional bus approaches [1]. Different types of NoCs are portrayed and compared in [5,6].

The design of a SoC involves a large amount of simulations to verify the correct functionality and to approximate the performance. However, simulation is typically on an order of 1000 times or more slower than the actual HW or emulation [7]. Furthermore, in most practical cases, creating simulation models for user input, external chips, etc. will require much additional work and may be hard to simulate perfectly. Hence, it is beneficial that different communication architectures can be evaluated using FPGA prototyping. However, the prototype acts as a black box with only I/O available to the developer, whereas simulator allows designer to pick any internal signal for evaluation. *Hardware monitoring* solves the problem of the limited visibility in FPGA compared to simulation. Monitoring the links in the NoC helps to recognize the bottlenecks and develop the NoC or the system mapping further. Fig. 1 shows an example of NoC monitoring using our approach.

Measuring the relevant figures from the NoC can be done on many different levels, all of which have their own trade-offs. For example, cycle-accurate traces need lots of hardware resources and bandwidth whereas performance counters on SW provide only very coarse measurements. Another key issue is to process and visualize the performance statistics so that designer can understand the system behavior and optimize it. Sometimes, Transaction Level Models (TLMs) are created for simulation to increase system abstraction from logical levels to complete units of communication such as packets of data and their acknowledgements. If the abstraction level of HW measurements can be increased accordingly, one can avoid the work of creating these separate models which are difficult to match with the actual HW.

The goals of this work is (1) to find a monitoring solution that works at the level of NoC, instead of single PEs, (2) to find a new, intermediate level of abstraction between the bit-accurate logic analyzer and highly-specialized custom function monitoring, and (3) to find ways to post-process, interpret and visualize the collected data that, by its nature, is vast in size.

In this paper we propose a scalable non-interfering low-level approach, called Trace Monitor, to monitor the behavior of NoCs in real-time with high precision. By "trace" we mean a set of data captured from the system under evaluation. Trace collection can be done in real-time and the results can be shown simultaneously on the host computer screen while running the prototype on FPGA. On the other hand, the trace collected can later be analyzed further, which is where we have set our main emphasis. The Trace Monitor toolset is freely available under the LGPL license.

This paper is an extended version of our conference paper in Norchip 2010 [8]. We have added a few additional references, and some implementation details of both HW and SW part. Then,

* Corresponding author. Tel.: +358 40 5916233.
E-mail addresses: antti.alhonen@tut.fi (A. Alhonen), erno.salminen@tut.fi (E. Salminen).

0141-9331/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2012.04.008

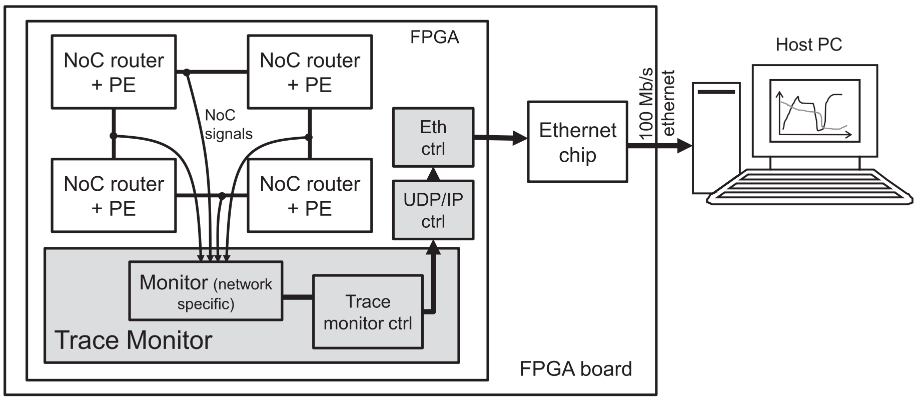


Fig. 1. An example instantiation of Trace Monitor (shaded blocks) in a 2 × 2 mesh Network-on-Chip on FPGA.

we introduce a new, sophisticated data analysis technique to generate approximations of traffic between two endpoints from the bare data of network links without data tagging or routing or address logging in HW. Finally, we present monitoring results from a multiprocessor MPEG-4 video encoder [9].

The rest of the paper is organized as follows: Section 2 presents related work on NoC monitoring. Section 3 discusses our objectives and major design choices. Section 4 summarizes the synthesizable HW part of the Trace Monitor in detail, whereas Section 5 describes the SW running on the host PC. Sections 6 and 7 present the post-processing algorithm and evaluate its performance. Section 8 presents the monitoring case study with MPEG-4, and finally, Section 9 concludes the paper.

2. Related work

The act of extracting information about the internal state or actions in an integrated circuit is hereafter called "monitoring". This definition is purposely vague, as there are many approaches for different needs.

2.1. Traffic generator based monitoring

The importance of evaluating NoCs on FPGA in addition to simulation is well portrayed in [7,10,11], where traffic generator (TG) and traffic receptor (TR) suites are presented. Traffic generators enable the integration of end-to-end monitoring mechanisms, but we take a more generalized monitoring approach with a possibility to monitor any real, synthesizable system on FPGA. This includes our Traffic Generator [12], covered very briefly in this paper, or any other synthesizable Traffic Generator, or any real system that can be synthesized on FPGA, as shown in the case study section.

2.2. Graphical presentation of monitored system

In [13], a graphical tool for NoC data flow evaluation similar to ours is presented. This tool, however, is limited to simulation. Simulation provides quick system set-up time for design space exploration, and allows unlimited system size without area restrictions, but is very limited in speed and thus simulated time as shown in [7,10,11]. Furthermore, creating simulation models for peripheral and I/O devices present in most real systems may turn out to be cumbersome or nearly impossible in some cases.

2.3. CPU and PE-oriented monitoring

General purpose processors used in an MP-SoC for the application can collect statistics by using counters or data tagging [14]. This approach is quick to harness and offers a graphical user interface. However, it interferes with the system as it affects the timing of data transactions and causes extra processing for the processors. Usage of monitor SW is naturally limited to general purpose processors only, thus it cannot be used for HW accelerators or other HW types, and it cannot log the inner state of the network but just the communication the processors see. The time resolution may be thousands of clock cycles, especially if a multi-tasking operating system is involved.

One option is to implement custom monitoring functions in the processing elements. Such a monitor is aware of the functionality of the PE, thus working at a higher level to log only relevant data. For example, it can be connected to an internal state machine of an HW accelerator. Nonetheless, these kinds of monitors are very useful for evaluating PEs' internal performances, and to some extent PE-to-PE communication, but not very effective to give understanding how the NoC performs. Furthermore, a generic type of a monitor for many different IP types may be hard to create.

2.4. Interfering monitoring

[15] presents a monitoring scheme with the aim of run-time optimization and resource allocation. The Run-Time Manager (RTM) collects statistical data from network routers. The monitor uses the NoC under evaluation to transfer monitoring data. Furthermore, monitoring data is transferred with higher priority than the actual data in system, which also demands for network that supports prioritized transfers. This is a good example of an interfering monitor that affects system timing and performance by using the studied NoC for the monitoring purposes.

2.5. Cycle-accurate trace collection

A low-level monitor can capture cycle-accurate signal values from a set of wires, resulting in *waveforms* when plotted as a function of time. For example, major FPGA vendors offer synthesizable *logic analyzers* in their FPGA synthesis tools for design debugging in a real environment, including actual user input and communication with external circuitry. This type of monitoring is straightforward to implement in HW; wires (signals) of the actual application are connected to (read by) the monitoring entity. This is completely transparent to the original application, unless the increased fan-out requirement affects timing in some special cases.

The major challenge is the huge amount of data generated; the data needs to be stored somehow locally and then transferred to a computer for off-line evaluation. Altera SignalTap [16] uses the FPGA's internal memory for storage and allows a relatively limited window (up to a few thousand cycles) and number of signals (a few dozen).

[17] presents an advanced monitor designed for debugging SoCs using the NoC paradigm. It allows real-time trace data collection but is focused on extracting the inner state of a system around specified triggers but not on making broad traces for performance evaluation. Another NoC monitor is presented in our earlier work [9]. In this paper, however we emphasize a more general monitoring approach that can collect and post-process unlimited amounts of data, the primary aim being performance evaluation instead of debugging.

2.6. NoC monitors close to ours

In [18], a method close to ours is presented, where some of the links of the NoC are monitored. However, it appears that only the links connecting IPs to routers are monitored, not the links connecting routers with each other, and only one link can be monitored at a time. Furthermore, trace length and monitor data bandwidth to PC are very restricted and amount of required on-chip memory is high. Whereas we selected to use windowing to reduce monitoring data, [18] detects events and timestamp them. Generality, especially in larger systems than a 2×2 mesh, or non-deterministic applications, is not shown.

3. Objective of this research

Our monitoring scheme results in understanding of the system studied in the following fields: how much any of the NoC links carry data and how many clock cycles are wasted in stalling. Additionally, approximations of how much the PEs communicate with each other are obtained. All this information is available as a function of time with good temporal accuracy.

The foremost objective is to study the performance bottlenecks in a NoC, such as stalling links. This information guides the designer on how to place PEs in the NoC, or to configure the NoC routers by changing the routing algorithm, FIFO depths, switching type, virtual channels, etc. In addition, we can get a first-order approximate of the power usage by observing total link activity, which is necessary in battery-powered large systems.

With respect to NoC benchmarking, the second objective is to construct a good approximation of all PE-to-PE intercommunication as a function of time. This information can be used to generate a *traffic model*, partly or fully automatically, to evaluate different network models efficiently [6]. Furthermore, the traffic models can be easily modified to evaluate what-if-scenarios, before applying such modifications in a real system.

The third objective is a highly practical and intuitive visualization. Since the trace is captured as a function of time, we can identify correlating actions happening in the SoC by looking at data transactions. For example, we can identify transactions that always precede others. Graphical visualization is a great help in this as it provides features like zooming, stepping, and coloring. It is also a suitable tool for teaching, because it offers a quick way to get a grasp of how a real large system works internally.

We have taken the following approaches when designing Trace Monitor:

Scalability:

– *Accuracy scalability:* The same system can be used to monitor the state of the NoC from the resolution of one clock cycle to tens of thousands of clock cycles, to adapt to different needs.
– *NoC scalability:* Different kinds of NoCs can be monitored with minor changes in the monitor structure. We have currently implemented monitoring for a generic mesh NoC and our HIBI hierarchical multi-hop bus [6].

Uninterference: The monitor itself shall not affect data transactions in NoC in any way. The only allowed exceptions are fan-out changes due to reading of NoC signals, and optional decrease of the NoC clock rate. The latter happens when a very high-resolution measurement is used and trace data transfer to the host PC becomes a bottleneck.

Affordable footprint: The whole monitor system, including peripheral control for data transfer to PC, should be small compared to network, so it is possible to harness in an existing FPGA design in most cases.

Free availability: Trace Monitor, including the synthesizable HW code and PC software for receiving and processing data, is freely available to download, use and further modification.

Fig. 2 shows the different design possibilities we evaluated during the design of Trace Monitor. Starting from left, there are many choices to make, regarding the accuracy and whether to stream the trace to PC or try to capture it completely before sending. The selected properties are shown in darker color.

4. Construction of Trace Monitor HW

The structure of the monitor HW is introduced here and the accompanying SW in the next section.

4.1. Structure

Fig. 1 depicts a top-level view of the system under evaluation, i.e. a NoC, the Trace Monitor, an external Ethernet chip and a host PC computer. Trace Monitor takes a logic analyzer-like approach. A monitoring block, written in synthesizable VHDL, connects to the relevant NoC signals. Captured NoC signals are processed to detect interesting states. Usually these states are.

- *data transfer* – one data word is transferred through the link,
- *data stall* – there is data available but it cannot be transferred yet.

Occurrences of these states are counted by monitoring few control signals. For example, a sender outputs a low level in its *empty* signal when data is available, and then the receiver acknowledges it by assigning *read_enable*. Fig. 3a shows an example where both control signals are active-high. The logic is very simple (few logic gates) and easily modified if the NoC uses some other flow control scheme. The handshake signals are very similar in all NoCs we have seen. Instead, the most laborous step in connecting the Trace Monitor to a new NoC is connecting the "probes" by writing the port mapping code, because it may be necessary to go deep down in hierarchy to bring some signals to upper level.

A control block, Fig. 3b, sends the collected counter data to the controller responsible for transferring the data to the host PC. We use an Ethernet connection as an example. The counter values are collected once in every *time window* (see Section 4.3). Our example has network (UDP/IP) header generation and Ethernet chip initialization logic between the control block and Ethernet chip. Table 1 shows a comparison of relative VHDL complexity of the entities, the network-specific monitoring block being the simplest one.

If necessary, the long "probe" wires can be segmented by inserting a constant number of flip-flops; then, the synthesis tool can place the flip-flops optimally and reach a high design frequency. "Globally Asynchronous Locally Synchronous" (GALS) systems can be supported by running the state counters on the local clock signals and adding a synchronization logic between the counters and Trace Monitor CTRL.

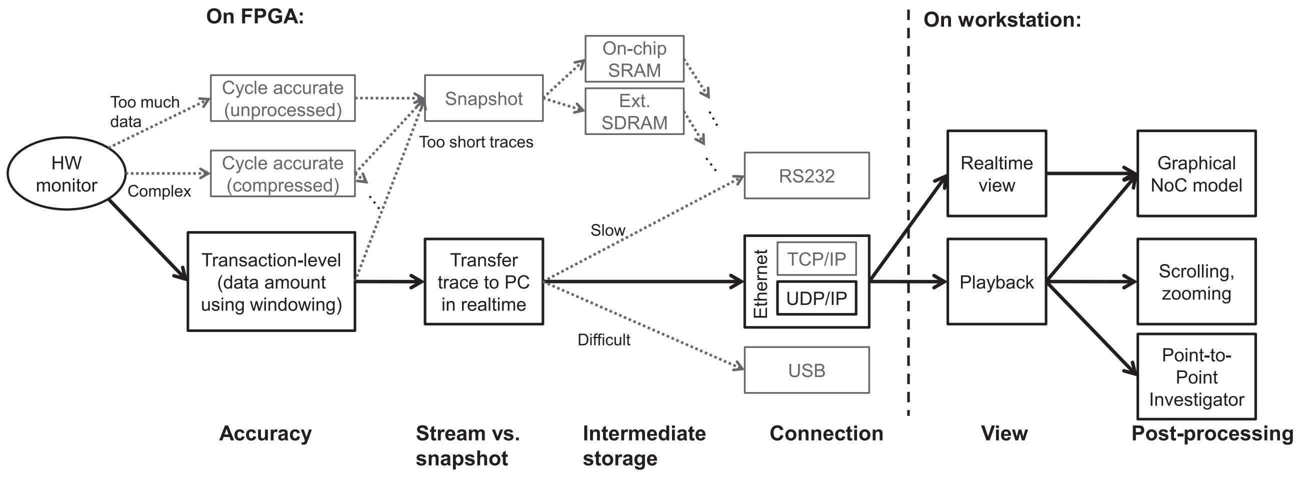


Fig. 2. Design possibilities explored.

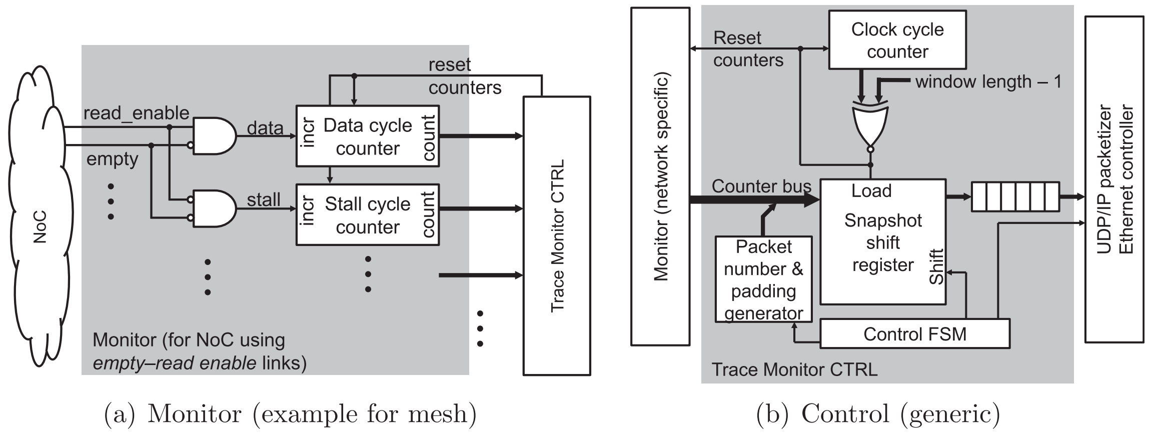


Fig. 3. Inner structures of the Trace Monitor blocks shown in Fig. 1.

Table 1
VHDL code lines of Trace Monitor entities.

Entity	VHDL code lines
Monitor for mesh	142
Monitor for HIBI	267
Trace Monitor ctrl	430
UDP/IP packetizer	1224
DM9000A eth chip ctrl	1914
LAN91C111 eth chip ctrl	1905

4.2. Connection to the host PC

We use the standard 100 Mbit/s Ethernet connection as a design example to transfer the data to the host PC. This allows quite large amount of data and is a very widely-used and standard way of transferring data. We have made a small-footprint hardware UDP/IP implementation in VHDL and controllers for Davicom DM9000A [19] and SMSC LAN91C111 [20] Ethernet controller chips. The DM9000A is featured e.g. in Altera DE2 FPGA development board [21], whereas SMSC LAN91C111 is featured e.g. in Altera Stratix II development boards [22]. Both implement the PHY and MAC protocols for 100 Mbit/s Ethernet.

Trace Monitor adds a serial number to every UDP packet to allow the detection of lost data. Other kind of data transfer devices can be used as well. Trace Monitor offers a general FIFO interface for data output.

4.3. Windowing

The first level of data processing is done at the HW level by the means of *windowing* as shown in Fig. 4. First, a two-word transfer (1A and 1B) goes through the link. The destination router waits one cycle before reading the data. Then, another two-word transfer (2A and 2B) shows the case of stalling for two clock cycles before the data can be transferred. When the window ends, counter values (amount = 4 and stall = 3) are copied to the snapshot register and are then reset back to zero. The two counter values here require just 4 + 4 bits.

The window length can be varied from one up to hundreds of thousands of clock cycles. The shortest window lengths allow cycle-accurate or almost cycle-accurate evaluation and debugging of the system. The downside is the vast amount of data which causes the data transfer to the PC to become the system bottleneck. This is easily addressed by frequency scaling. When the NoC frequency is lowered, cycle counts remain unchanged if the clocks of the PEs are scaled accordingly. The downside to frequency

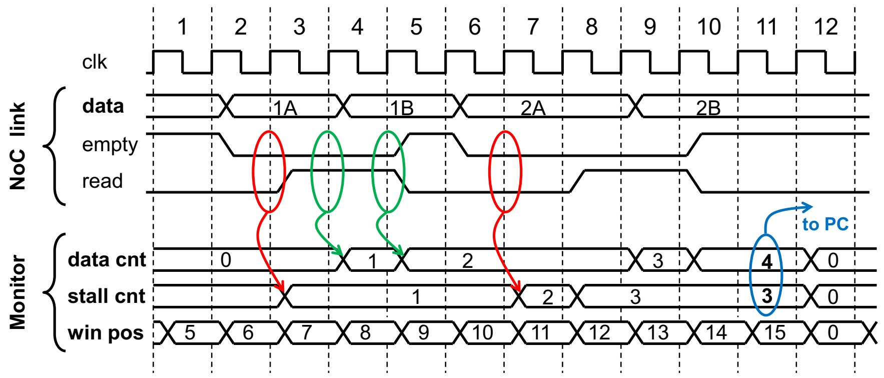


Fig. 4. An example with *empty—read enable* type of link, showing the counters and the effect of windowing. Four words are transferred but the receiver stalls occasionally. Counter values are sent and reset at the end of window.

scaling is that the real-time ability is lost. If this is unacceptable, temporal accuracy can be compromised by using longer window, quickly lowering the amount of data.

4.4. Bandwidth requirements

Total payload bandwidth can be calculated as

$$B_{payload} = (f_{clk}/w) \cdot (D_{snapshot} + P_{snapshot}) \tag{1}$$

where $B_{payload}$ is the total payload bandwidth in bits per second, f_{clk} is the clock frequency of the monitored network in Hz, w is the window length in clock cycles, $D_{snapshot}$ is data in bits per snapshot and $P_{snapshot}$ is padding data in bits per snapshot, so that $(D_{snapshot} + P_{snapshot})$ is a multiple of the width of monitor data bus (Ethernet controller data bus in our example).

$D_{snapshot}$ can be calculated as

$$D_{snapshot} = \lceil log_2(w + 1)\rceil \cdot (n_{counters} \cdot n_{links} + n_{global}) \tag{2}$$

where w is window length as before, $n_{counters}$ is the number of counters per link, e.g. 2 (data-read and data-stall), n_{links} is the number of monitored links in the NoC under evaluation, n_{global} (optional) is number of global counters, e.g. bus idle counter in a HIBI network segment.

For example, a 4 × 4 mesh has total of 80 unidirectional links (the links connecting routers and PEs also included). Assuming a window length of 500 cycles, 25 MHz operating frequency, and two counters (data amount and stall), the payload bandwidth is

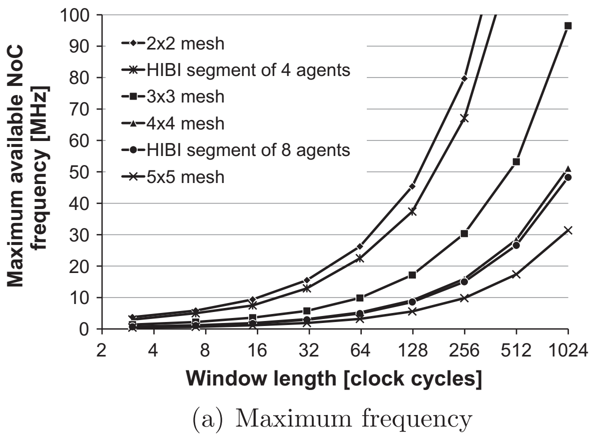
$$\begin{aligned} B_{example} &= (25\text{ MHz}/500) \cdot (\lceil log_2(500 + 1)\rceil \cdot 2 \cdot 80 + P_{snapshot})bit \\ &= (50\text{ kHz}) \cdot (9 \cdot 160 + 0)bit = 50\text{ kHz} \cdot 1440bit \\ &\approx 72\text{ Mbit/s} \end{aligned}$$

Fig. 5a shows the relation between window length and maximum available clock frequency when the Ethernet connection determines the maximum payload bitrate of about 80 Mbit/s. Assuming 127-cycle windows, small networks obtain operation frequency around 40 MHz and larger ones around 10 MHz. Hence, the operation is nearly real-time and orders of magnitude faster than simulation.

Increasing window length makes state counters and the snapshot register wider, hence increasing the area on chip somewhat. This relation is shown in Fig. 5b. Even with very long windows the size of Trace Monitor is reasonable: a few thousand adaptive look-up-tables (ALUTs) – comparable to 1–3 typical NoC routers, depending on the network.

5. Construction of Trace Monitor PC software

The collected data amount is so high that intuitive ways are needed for showing and analyzing it for trends, anomalies and correlations [14]. Our major objective is to be able to both view real-time data on screen while the FPGA prototype is running; and to be able to easily play back the recorded trace and seek for interesting occurrences. In addition, data parsing to a standard format for further processing is possible.



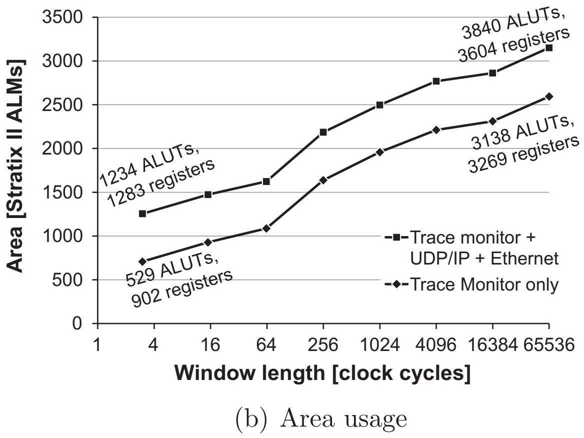


Fig. 5. (a) The maximum available NoC frequency on some networks as a function of window length, when using a 100 Mb/s Ethernet connection. (b) Logic usage of Trace Monitor on Altera Stratix II FPGA, with 3 × 3 mesh as an example.

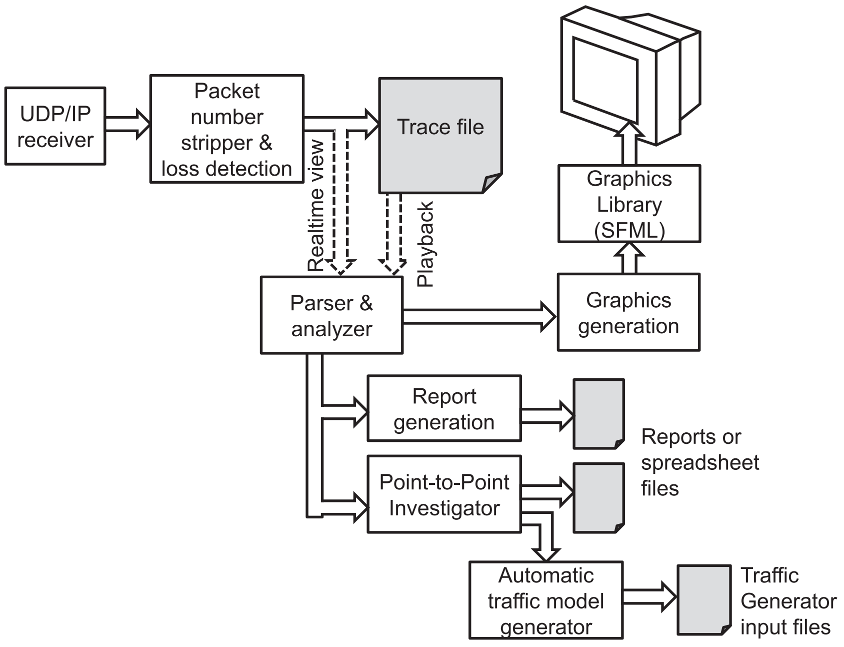

UDP/IP receiver
Packet number stripper & loss detection
Trace file
Realtime view
Playback
Graphics Library (SFML)
Parser & analyzer
Graphics generation
Report generation
Point-to-Point Investigator
Reports or spreadsheet files
Automatic traffic model generator
Traffic Generator input files

Fig. 6. Software data flow block diagram.

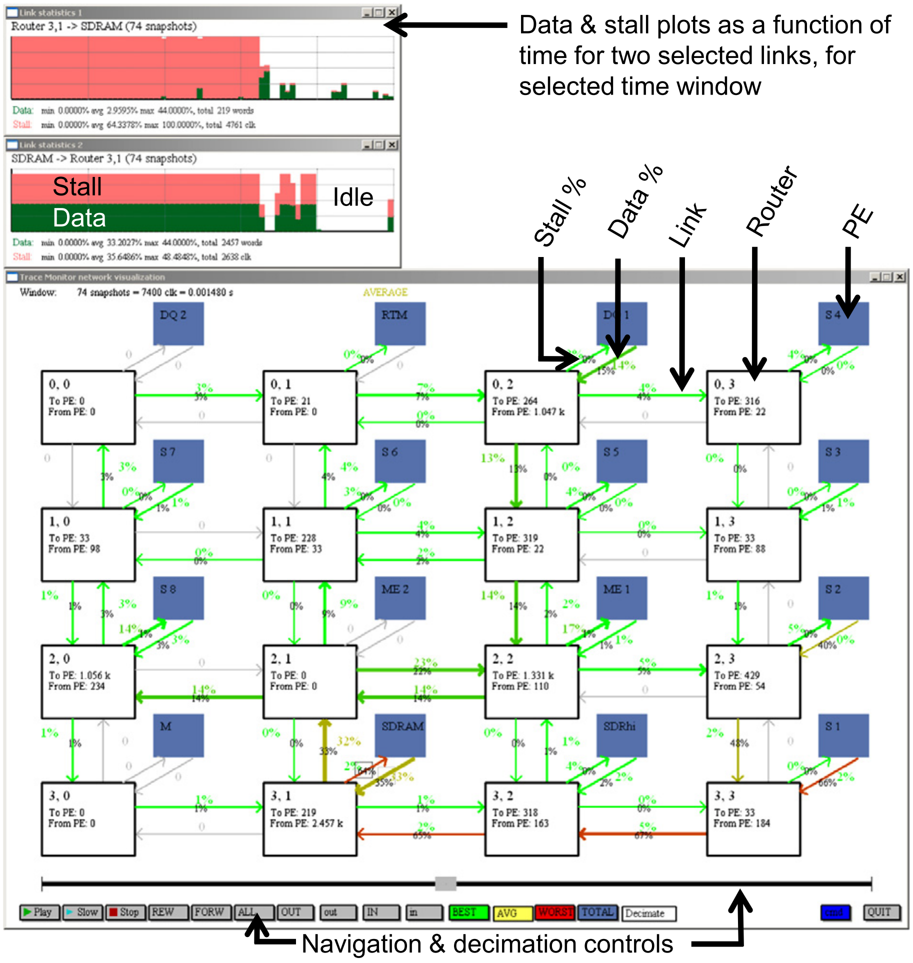

Data & stall plots as a function of time for two selected links, for selected time window
Link statistics 1
Router 3,1 -> SDRAM (74 snapshots)
Link statistics 2
SDRAM -> Router 3,1 (74 snapshots)
Stall
Data
Idle
Stall %
Data %
Link
Router
PE
Trace Monitor network visualization
Window: 74 snapshots = 7400 clk = 0.001480 s
AVERAGE
Navigation & decimation controls
Play
Slow
Stop
REW
FORW
ALL
OUT
out
IN
in
BEST
AVG
WORST
TOTAL
Decimate
QUIT

Fig. 7. Screen capture of our Trace Monitor PC software. A 4 × 4 mesh network is evaluated after trace capture. The GUI shows the NoC structure and two detailed link statistic windows. The user can select the links by clicking them.

```
TRACE MONITOR REPORT

Window length = 500 clk
Clock rate    = 25000000 Hz
Window range start = 126838 w = 63419000 clk = 2.53676s
Window range end   = 135302 w = 67651000 clk = 2.70604s
Region size        = 8464 w = 4232000 clk = 0.16928s

Link statistics:
Link 0: Router 0,0 -> Router 0,1
  DATA
    MIN 0 %   AVG 1.0935 %   MAX 44 %
  STALL
    MIN 0 %   AVG 1.0065 %   MAX 44.6 %
. . .
```

Fig. 8. Excerpt of a generated plain-text report.

Fig. 6 shows the structure of the analysis program. Once the data arrives to the PC (top-left corner), it is in a binary format where the counter values can be extracted by a simple parsing algorithm. Our program is written in C++ and uses the free, cross-platform SFML library [23] to receive UDP packets and generate graphical output via a user interface.

Trace Monitor PC Software currently consists of 10 600 lines of C++ code. Data receiver and parser being the simplest, 90% of the code is for visualization, reporting and further post-processing of the data.

5.1. Visualization

The program saves the binary trace file on the hard disk for later examination. The same program can be used to review the generated file. Fig. 7 shows a screen capture of the PC program, showing an evaluation of a 4×4 mesh network. The navigation bar on the bottom can be used to explore through the recorded trace. Real-time evaluation looks identical, but without the navigation function.

Links are visualized as arrows that are width-encoded depending on data amount and color-encoded depending on stalling. Color-encoded percentages near the links depict the amount of data transferred. Stall percentages are added over the links with a small font. NoC links can be clicked to pick any link to the "Link statistics" window (two of them shown on top). It shows the data transfers and stalls as a function of time during the zoom region (1.48 ms in Fig. 7).

As remeasuring with longer window is not always reasonable, the PC software enables further down-sampling or decimation of the data. For the user, this looks intuitively like zooming. The user can move on the time scale and adjust zoom level to view both long-scale statistics and precise timing information down to the window length. When zooming out, three different decimating functions can be used; worst case, average, and best case. For example, if there is only one transfer that occupies the link at 100% for one and only one window, and the user zooms out to see the average of 100 windows, worst case mode shows 100% usage, average mode shows 1% usage and best case mode shows 0% usage. This display mode can be changed any time while viewing.

When the zoom level shows a longer time than one window, the separate "Link statistics" windows become very useful. The user can see a plot of data transfer and stalling for any link as a function of time. This way, it is easy to find correspondences with how the system functions; such as specific accelerators communicating at a certain point of time, and memory accessed at another point of time. The user can click on the link statistic plots at any two points of time to set the zoom level between them.

5.2. Reporting

The PC software also writes a standard CSV (Comma Separated Values) file that can be viewed with any spreadsheet program. Further analysis and figures can be done with these programs.

Fig. 8 shows a snippet of an additional plain-text report that can be automatically generated from any zoom region. The report shows the same values that are displayed graphically but in textual form and in higher precision. In this example, a region of 169 ms or 8464 windows of 500 clock cycles is reported.

6. Point-to-Point Investigator

So far, the emphasis has been on monitoring and visualizing the individual NoC links. Now we aim to understand the connections between PEs. In practice, this means a 2D matrix of $n_{PE} \cdot n_{PE}$ in size, showing how much data was sent from each PE to each PE during one window. An example is depicted in Fig. 9. This is created for every time window, leading to a 3D matrix that describes the actual communication among the PEs at any point of time. This matrix is very useful for modeling the time behavior of each IP. From this information, we can automatically create a traffic model for every IP; these models can then be simulated at Transaction or Register Transfer levels as well as ran on FPGA using synthesizable Traffic Generators. This enables the designer to see how modifications in the NoC configuration, or selecting another NoC altogether, change the system performance, without altering the original system, which might be more time-consuming.

We will use term "point-to-point path" hereafter to describe a higher level of communication from any PE to another. Note that this communication graph is independent of the NoC topology.

6.1. Design options

The presented monitoring scheme could be modified in a few different ways to identify the sender and receiver, such as

1. monitoring the actual data in NoC for addresses
2. increasing the word width to insert tags into data
3. creating custom monitors for different HW types; with a general purpose processor, software could be used.

The first two require quite a bit of HW area, and would need substantially more network-specific code than the handshake sig-

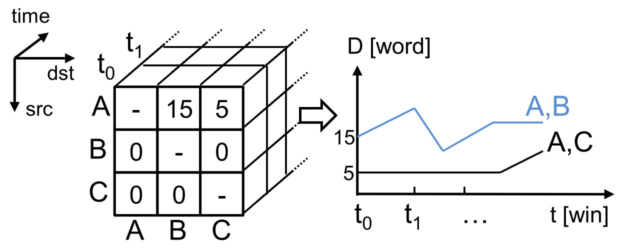


Fig. 9. A three-dimensional matrix showing the data amounts transferred between PEs A–C as a function of time. The graph can be used for visualizing the data.

nal monitoring. In the second case, the support for arbitrary word width is required from the network as well. The third option takes a lot of non-generic extra work, and in the case of processors, the timing is changed by the addition of monitor code.

In all cases, the monitoring of every point-to-point path in hardware causes a huge increase in the monitor data. For example, monitoring a 8×8 mesh for every link needs $n_{countersperlink} \cdot (n_{meshlinks} + 2 \cdot n_{pe}) = 704$ counters whereas counting every point-to-point path requires $n_{countersperlink} \cdot n_{pe} \cdot (n_{pe} - 1) = 8064$ counters, i.e. 11x increase in monitor data transferred to the PC. This is directly comparable to the fact that the mesh NoC is considered scalable but point-to-point connections are not. Similarly, monitoring the mesh links is scalable, but monitoring underlying point-to-point connections is not.

Table 2 compares the presented method (top) to address snooping and their combination. It lists the data amounts and minimum window lengths assuming 25 MHz frequency and a 100 Mbit/s connection to the PC.

Fortunately, the point-to-point transfers can be deduced by post-processing the links statistics. The resulting communication matrix will inevitably be an approximation, but it must be noted that the additional HW cost for this operation is zero.

6.2. Post-processing algorithms

We developed three experimental algorithms, two very simple ones and one somewhat more complex; namely the min–min, the min–min–min and the Elimination–Deductive algorithms.

The foremost assumption is that during one short time window, there is usually a relatively small number of transactions occurring. Otherwise, it may be impossible to perfectly distinguish them from each other.

The min–min algorithm works by taking one point-to-point path $src \rightarrow dst$ at a time. Of course, the traffic on the path cannot exceed the amount sent by source or that received by destination. Hence, the estimated data amount is

$$min{-}min : D_{src \rightarrow dst} = min(Snd(src), Rcv(dst)), \tag{3}$$

where $Snd()$ denotes the outgoing data, $Rcv()$ the incoming data. For example, if A sends 20 words, B receives 15 words and C receives 5 words during a window, this algorithm deduces that

$$A \rightarrow B = min(20, 15) = 15$$
$$A \rightarrow C = min(20, 5) = 5. \tag{4}$$

In this simple case, this is correct, but if we add a transfer B → C, it will be counted as an extra to A → C due to increased RX amount at C.

The min–min–min algorithm is identical to min–min, but all the in-between links are used to assist the process. This necessitates that the routing algorithm is deterministic and known. Then the resulting data amount becomes

$$\text{min—min—min} : D_{src \rightarrow dst} = min(Snd(src), Rcv(dst), D_{link_1}, D_{link_2}, \ldots, D_{link_n}), \tag{5}$$

where $D_{link_1} \ldots D_{link_n}$ are the data amounts in NoC links between the source and destination PE.

Now, if one of the links between A and C would be 0, the algorithm would know that the data coming to C originates from somewhere else than from A, thus counting the data only once. This simple algorithm works adequately when the transactions during the same time window do not utilize the same links. Otherwise, the same data may still be counted multiple times.

The third algorithm tries to deduce a complete network situation (still during a short time window) as a whole. It eliminates infallible P2P paths and assigns the data amounts with a set of sudoku-like *rules*. The algorithm starts out like the previous ones but uses a recursive function to try different possibilities to find the most probable description of the network situation. For example, it understands that data cannot make a U-turn in a router, in order to decide how the multiple simultaneous connections are mapped to each other. As an another example, it uses the information whether some paths are already solved to reach a situation where some rows or columns in the communication matrix have only one unknown that can be solved from the row or column sum.

Once a path is deduced by using the rules, its data amount is subtracted from the associated Snd () and Rcv () amounts and all in-between links. This ensures that all data is counted only once. If the calculated data amounts do not match, the algorithm backs off to try different routings, until the best possible match is found. In unclear situations, an educated guess can be made automatically: this means connecting the most sending sender to the most receiving receiver if all other rules allow this, and then continuing the algorithm from the start.

To improve the results from any algorithm, the matrix can be easily *equalized* since the totals are known sender-wise and receiver-wise. Equalization calculates the sum of each row and compares it to the known sent amount of that PE, and each element is multiplied by their ratio. Then the same is repeated to every column. This action proved to be surprisingly effective with the simple algorithms.

Table 2
Different HW monitoring approaches.

Monitoring approach	Data amount (approx.)	Shortest window in a 4×4 mesh at 25 MHz	Shortest window in a 8×8 mesh at 25 MHz	Main purposes
Links and PE inputs and outputs	Manageable, $\propto N$	450 clk	2600 clk	Hot spot detection, network configuration, power analysis
Point-to-point by address snooping	High in large networks, $\propto N^2$	1700 clk	40,000 clk	PE intercommunication modeling
Both of the above	High in large networks, $\propto N^2 + N$	2400 clk	44,000 clk	Both of the above

7. Evaluating Point-to-Point Investigator results

Before the actual usage in a case study, we created known test cases for the Point-to-Point Investigator as a proof of the concept. Thus, we created exactly correct point-to-point matrices for the test cases and compared them to the results obtained from our algorithms.

7.1. Defining the criteria

We evaluated many different quality metrics, some of them trying to portray the overall total error, whereas some of them emphasize the small connections that were completely lost, and some of them emphasize the “false positives”, or transactions that were not happening in reality.

We found that the sum-of-absolute-differences ($SAD_{allpaths}$) in proportion to total data amount is the most useful universal metric in comparing algorithm performance. Absolute values of differences between the correct and deduced values in matrices are summed, and the sum is divided by the total correct data amount. Thus, 0% in total SAD means the perfect analysis. For example, let us assume that the first row in Fig. 9 had {0, 16, 4} instead of the correct {0, 15, 5}. The SAD would be 0 + 1 + 1 = 2 although the total data amount is correct. Relative SAD is 2/20 = 10%. We use such SAD-% instead of the absolute SAD so that we can compare different test cases with each other.

7.2. Traffic generators as test cases

We used our synthesizable traffic generator (HWTG) [12] to generate a known amount of traffic in the test NoC of 4 × 4 mesh. One traffic generator (TG) is instantiated per modeled PE. Each TG consists of a *sender*, the traffic generation part that creates and sends packets by using *events* and *triggers* as rules, and a *reader*, the receiver part that monitors and counts the data coming from the network, and causes the *sender* to trigger to these packets if a relevant trigger rule exists.

Events cause a packet to be sent at a defined time either once, or multiple times at a specified time interval. The size of the packet and recipient TG can be defined explicitly or randomized. *Triggers*, on the other hand, are activated when a matching type of packet is received by the TG in question, causing an answer transaction after a specified time.

The reader part in TG counts the exact amount of data coming from every other TG by looking at the sender field in the data. This information is reported to host computer after the measurement run is over.

The HWTG system is written in synthesizable VHDL and consists of ca. 6000 lines of code. It is freely available under LGPL license on our website.

The word counts in the TG reports form a 2D matrix of every sender-receiver pair. These numbers are word-accurate and thus can be directly used in the evaluation of the Point-to-Point Investigator.

7.3. Test cases

We created three different TG models for testing. Their properties are summarized in Table 3. Also, the data flow diagram of the first test is shown in Fig. 10. None of the tests nor the TG mapping to mesh were designed particularly P2P Investigator in mind; they were created before.

All test cases used 4 × 4 mesh network with wormhole switching and fixed *Y–X* routing. Tests ran at 5 MHz for four seconds.

Table 3
Test case properties.

	#Nodes	#Edges	Avg dest per node	Max dest per node	RX max (MB/s)	TX max (MB/s)	TX avg (MB/s)	Total TX data rate (MB/s)
Test 1	16	22	1.38	7	43.95	43.95	16.52	264.38
Test 2	15	26	1.73	6	18.40	18.31	2.39	35.86
Test 3	14	20	1.43	4	3.09	2.88	0.75	10.52

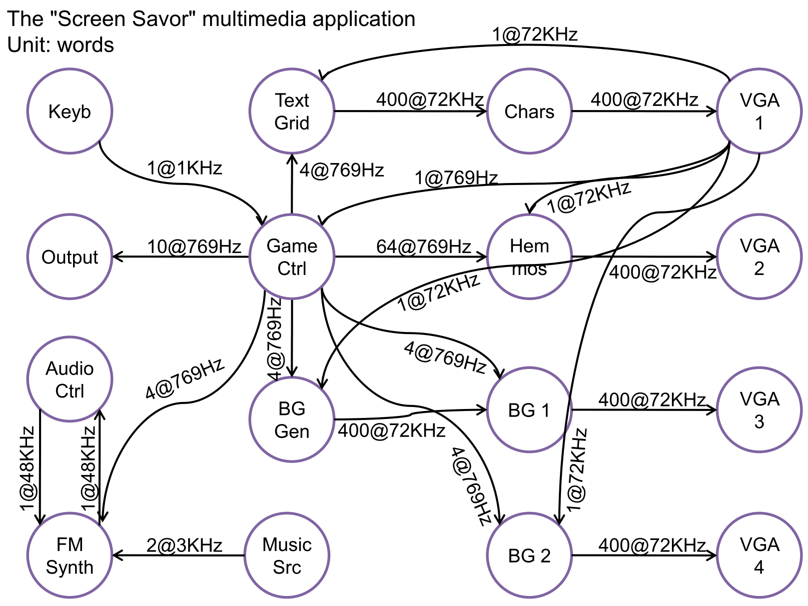


Fig. 10. The data flow graph of test case 1. Nodes denote tasks. Throughputs are marked with each communication arc in terms of data words and how often they are sent.

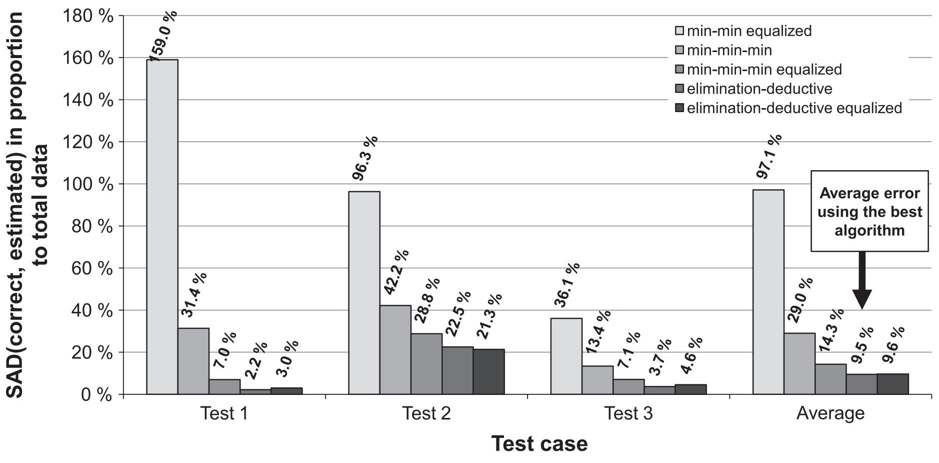


Fig. 11. The error percentages in P2P Investigator's traces with different processing algorithms. The min-min algorithm without equalization benchmarked 414%, 249% and 49% in the three tests, and is not shown. The Elimination–Deductive algorithm was the most accurate.

Table 4
Properties of the case study application.

Property	Value
Application	Multiprocessor MPEG-4 simple profile encoder
Resolution	CIF (352 × 288) in the case study example. 720p HDTV also studied
Input bitstream	150 kB/ frame
Output bitstream	Approx. 5 kB/ frame
FPS @ 50 MHz	Approx. 25 FPS
HW platform	Altera Stratix II S180 dev board
NoC	4 × 4 mesh with fixed *Y–X* routing
NoC switching	Store-and-forward or wormhole
HW accelerators	DCT-Quantization-IDCT, motion estimation
Other IPs	Resource manager, shared SDRAM controller
SW platform	Altera Nios II softcore processor
Parallelism	Same program multiple data
Memory	Local scratch-pad data memory & shared SDRAM w/ DMA
Language	Ansi C
Trace Monitor window length	100, 500 and 1000 cycles
System clock rate	5 MHz, 25 MHz or 50 MHz
Encoded sequence	foreman.cif [24]

Window length for Trace Monitor was 100 clock cycles. For every test case, all three algorithms presented were tested.

7.4. Evaluating the results

Fig. 11 summarizes the error (SAD-%) in the models produced by the P2P Investigator when compared to Traffic Generator's original models.

As expected, the simplest min-min algorithm shows heavy levels of error (bars omitted for clarity in the figure), which is acceptable only in a limited number of cases. Adding equalization improves the result considerably by reducing the overshoot (exaggeration of data amount), but as a trade-off, it causes undershoot (too low data amounts). For example in Test 1, equalization decreased the overshoot from 411% to 79.1%, but increased the undershoot from 3.9% to 79.9%. As a result, the total SAD-%, the sum of undershoot and overshoot percentages dropped to 159%.

The min-min-min algorithm performed clearly better. For example, the total SAD in Test 1 was 31.4% (with 27.5% overshoot and 3.9% undershoot) and only 7.0% (with 3.0% overshoot and 4.1% undershoot) after equalization.

The Elimination–Deductive algorithm indeed scored best in all tests. At this point, equalization improved the result only in the most demanding Test 2; otherwise, it only added a new source of error. For example, the total SAD in Test 1 was 2.2%, and 3.0% after equalization.

The margin to equalized min-min-min algorithm was quite small, but on the other hand, Elimination–Deductive algorithm still has potential for further development, whereas the two others are quite fixed by definition.

With total arithmetic mean error of 9.5% of total data amount in all three tests, we found that the Point-to-Point Investigator indeed can be used to create very realistic, although not perfect, approximations of end-to-end traffic from mere interconnection network link usage statistics. No information about the original data flow model was used in the software. In most cases, this accuracy is enough to create models that can be used for NoC exploration.

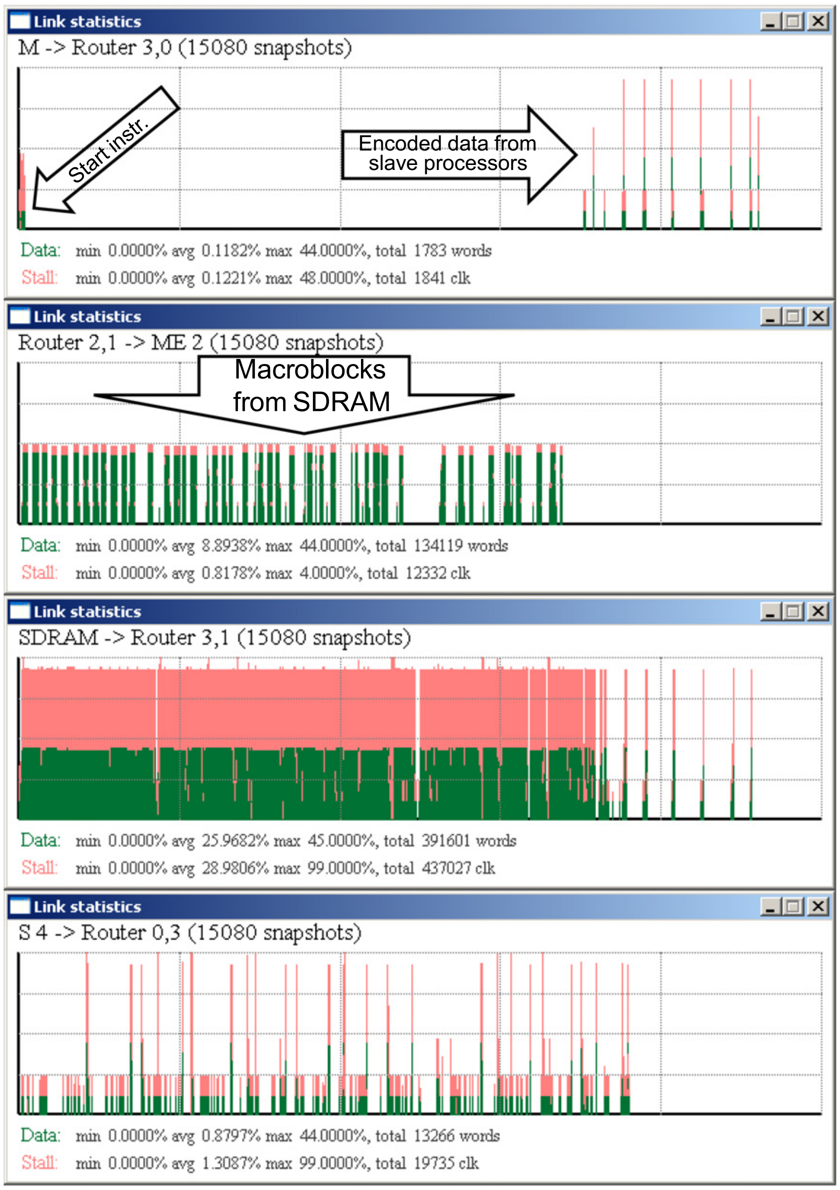
It is well possible that a considerable part of the error comes from the windowing. As all the transfers over the NoC have latency because of the routers, some of the transfers start during a different window than they end in. Hence, some data is left unaccounted for, causing a small negative bias to the direction of error. Also, the packet header overhead in network links disturbs the precise evaluation of end-to-end payload traffic. Solving these problems to further improve the approximations is our future work.

The actual processing core of the algorithms consist of 40, 80 and 1500 lines of C++ code, respectively for min-min, min-min-min and Elimination–Deductive algorithms.

8. A case study of an MPEG-4 encoder on FPGA

Finally, we show an example of harnessing the Trace Monitor into an existing Multi-Processor Network-on-Chip design, a data-parallel MPEG-4 encoder using Altera NIOS II soft-core processors, several HW accelerators, shared SDRAM memory and an interconnection network. This particular system has been earlier used to demonstrate the NoC concept [9]. Here, our goal is to simply demonstrate how Trace Monitor can be attached to such an existing system and how measurements can be made. Further research about the video system is beyond the scope of this paper.

The system consists of a Master Processor (M in Fig. 7), eight Slave Processors (S1–S8), two Motion Estimation (ME) and two DCT–Quantization–IDCT (DQ) HW accelerators, HW Resource Manager (RTM) and an SDRAM Controller with two interfaces with different priorities (SDRAM and SDRhi). These IP blocks are



M -> Router 3,0 (15080 snapshots)

```
Data:  min 0.0000% avg 0.1182% max 44.0000%, total 1783 words
Stall: min 0.0000% avg 0.1221% max 48.0000%, total 1841 clk
```

Router 2,1 -> ME 2 (15080 snapshots)

```
Data:  min 0.0000% avg 8.8938% max 44.0000%, total 134119 words
Stall: min 0.0000% avg 0.8178% max 4.0000%, total 12332 clk
```

SDRAM -> Router 3,1 (15080 snapshots)

```
Data:  min 0.0000% avg 25.9682% max 45.0000%, total 391601 words
Stall: min 0.0000% avg 28.9806% max 99.0000%, total 437027 clk
```

S 4 -> Router 0,3 (15080 snapshots)

```
Data:  min 0.0000% avg 0.8797% max 44.0000%, total 13266 words
Stall: min 0.0000% avg 1.3087% max 99.0000%, total 19735 clk
```

Fig. 12. Screen capture examples from Trace Monitor PC software, evaluating the trace from MPEG-4 video encoder MPSoC prototype on FPGA.

connected by a 4 × 4 mesh interconnection network. First, store-and-forward switching with fixed *Y–X* routing was used.

The Master Processor receives video from Ethernet connection, splits it to slaves for encoding, and transmits the compressed data back to the host computer for decompression and displaying.

Altera Stratix II S180 FPGA development board was used. Because the board has only one Ethernet connection, used by the video application, we built an Ethernet expansion card around the Davicom DM9000 Ethernet chip [19]. This connection is used to transfer Trace Monitor data to PC unintrusively.

The properties of the case study application are summarized in Table 4.

8.1. Traffic analysis

First, we have placed the PEs as shown in Fig. 7. The system is ran at different clock rates and different window lengths, as shown in Table 4.

Fig. 7 is a screen capture of the Trace Monitor PC Software, showing a trace review zoomed at a level which highlights a clogging transaction with the SDRAM controller. Of course, read and write operations cannot occur simultaneously. The writing operation has to wait, which is seen as 100% stalling in the Link Statistics 1 window. After the reading stops, all of the stalled data can finally be written. The main window shows how the links between routers (3, 1), (3, 2) and (3, 3) also become stalled, as they are displayed in red. The amount of data transferred in these links is low.

Fig. 12 shows four IP link statistics when the trace is zoomed out so that the processing of one complete video frame is seen. These same actions happen very similarly for every frame. The role of different IPs from the NoC viewpoint can be easily seen; for example, the master processor uses the NoC only occasionally, and even then the amount of data is not high. Also, the motion estimators can read the data instantly with little stalling even though they transfer the highest amounts of data: each spike corresponds to processing of one 16 × 16 pixel macroblock.

The total data amount transferred during one frame is approximately 2.7 megabytes. About 15% of that goes to the SDRAM and 50% comes from there.

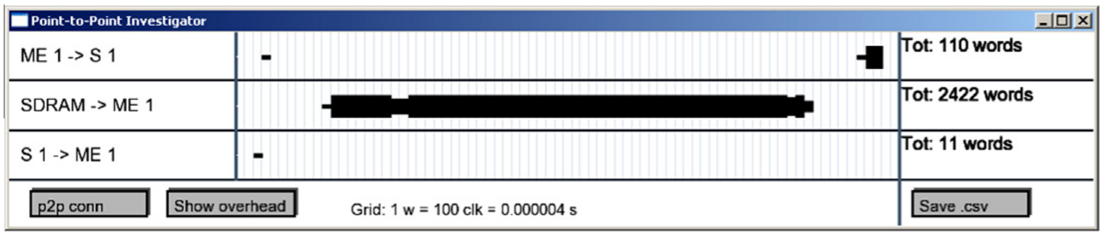


Fig. 13. An example screen capture from the P2P Investigator result visualization, zoomed to show one transaction between a slave processor and a Motion Estimator. Slave 1 sends a processing instruction to ME 1. ME acknowledges the instruction, reads the necessary macroblocks from the SDRAM, and finally reports the results (motion vectors) to Slave 1. This way, a higher level of communication between any two endpoints can be seen without actually monitoring those end-to-end connections.

It is also revealed that the application never utilizes one link in the network. Hence, this link might be removed in favor of area utilization.

8.2. Remapping the application

We changed the IP mapping so that Motion Estimators, using SDRAM extensively as can be seen from the trace, are far away from the SDRAM. Then we ran the system again to capture a new trace and used the "create detailed report" function. Hence, we easily obtained the sums of the total data and stall cycles in all links.

In the system where MEs are far away from the SDRAM, the total average link utilization during video encoding is 2.05% of the maximum bandwidth of all links. Originally, the utilization was 1.42% or one third smaller. This is due to shorter paths between the most active processing elements. Minimizing this reduces power consumption of the NoC as well as gives higher amount of free bandwidth for other communication. As an average of all links, these numbers are not very high, but are usable when a single figure is needed to portray the system. Actually, the links between Motion Estimators and SDRAM are most heavily used at 35% utilization during processing.

Original stall percentage was 3.73% which is about one tenth smaller than 4.19% acquired from the remapped system. This difference in wasted clock cycles becomes important when the NoC is the system bottleneck in timing. Again, links between MEs and SDRAM stalled at 37% on average.

8.3. Communication overhead in store-and-forward switching

Originally, the system used store-and-forward switching. The downsides of this scheme became very apparent quickly after analyzing with Trace Monitor. We used additional monitoring points so that we measured data amounts between IP and NoC router from two points for both RX and TX; before and after the Packet Codec. This way, we got exact measurements of the network packet overhead.

It was revealed that during the processing of one video frame in the store-and-forward switched system, 80% of the data coming from the SDRAM was network overhead. By changing the network swithing to wormhole, this dropped to 45%, also reducing the percentage of stall cycles of the total time from 29% to 8.4%.

The reason for this is that the communication is scattered to many small transactions, and the store-and-forward switched packet encoder fills small packets to match the packet size.

This information cannot be found with SW measurements as the packet decoders strip the overhead; still, it is an important factor for the application developer who needs to know the *real* bandwidth requirement of the application.

8.4. Debugging the system

The system under study has always had a problem of occasional freezing. Former debugging attempts have not led to any results.

Even though Trace Monitor is not primarily meant for debugging, we very quickly found a valuable hint for the freezing problem by finding that there still is traffic in the network after the system freezes, generated by processors polling the Resource Manager. Clearly, the Resource Manager answers that all HW accelerators are busy when in reality they are not and thus the polling continues forever.

Former (intrusive) attempts to add debug prints to the processor SW changed the timing of the system so that it became very hard to get it freeze.

8.5. Point-to-point traffic evaluation

Finally, we ran Point-to-Point Investigator, using a trace region of processing four video frames of a CIF video sequence. The post-processing using the Elimination–Deductive algorithm took less than a minute on a typical workstation and outputted a zoomable

Table 5
The MPEG-4 encoder SoC evaluated with P2P Investigator. Data amounts are per one CIF interframe. Direct, exact measurements are shown in bold.

Connection	Metric	P2P Investigator result	Software measurement	Custom HW monitor	P2P Investigator error
Slaves → SDRAM (sum of 16 paths)	Payload	275 kB	**289 kB**	**293 kB**	−6.1%
	Pay + oaddr	357 kB	⩾372 kB	**377 kB**	−5.4%
	All data	517 kB	⩾541 kB	**546 kB**	−5.3%
Slaves → DQs (sum of 16 paths)	Payload	305 kB	**306 kB**	**310 kB**	−1.7%
	Pay + oaddr	352 kB	⩾331 kB	**355 kB**	−1.1%
	All data	443 kB	⩾384 kB	**450 kB**	−1.7%
SDRAM → MEs (sum of 4 paths)	Payload	939 kB	N/A	**953 kB**	−1.5%
	Pay + oaddr	1051 kB	N/A	**1064 kB**	−1.2%
	All data	1282 kB	N/A	**1284 kB**	−0.1%
DQs → Slaves (sum of 16 paths)	Payload	607 kB	**605 kB**	**610 kB**	−0.4%
	Pay + oaddr	879 kB	⩾654 kB	**895 kB**	−1.8%
	All data	1463 kB	⩾765 kB	**1464 kB**	−0.1%

Table 6
Area comparison of several design entities of our case study.

Entity	Area (ALMs)	(%)
Processors total	22,112	40
One Nios II sub-system	2 442	
Other IPs total	13,114	24
4×4 mesh total	17,038	31
One router with packet codec	1386	
Trace Monitor with Ethernet ctrl	2506	5
Trace Monitor only	1950	
Total	54,770	100

graphical representation of all point-to-point communication (Fig. 13 inset) and a CSV file of the totals.

To obtain correct communication control metrics for comparison, we created a custom end-to-end monitoring HW module. It selectively counts data of a specific end-to-end path by the means of address snooping (see Section 6.1). This special module is connected to the existing Trace Monitor infrastructure. To minimize the bandwidth, we monitored all source network routers for the addresses of only one or two destination network routers at a time, rerunning the test several times to obtain a number of interesting end-to-end communication volumes. These numbers were then compared to some former software approximations, and finally, to the results obtained with the P2P Investigator, the entity we wanted to verify.

The foremost problem in this comparison was deciding what to measure; total data is interesting for the NoC traffic analysis whereas the payload is more interesting for analysing the application. Both can be measured with HW, but SW can only measure the pure payload accurately, as the headers are generated at the NoC HW. Furthermore, there was one header flit that was not used by the NoC but by the application—but which was still inserted by the NoC HW: namely the Original Address flit (oaddr). As it was hard to decide whether to treat this as header or payload depending on the viewpoint, we created three different metrics; pure payload, payload with the original address flit, and all data, consisting of payload, original address flit, packet length flit and mesh address flit.

Trace Monitor can measure all three metrics from TX and RX links and utilize this information in P2P Investigator. The custom end-to-end monitoring HW was also created in such way. Hence, only the former software measurement were lacking the ability to measure anything but payload. We modified the software to give an approximation of the header traffic in the optimum case.

The results are summarized in Table 5. P2P Investigator numbers are compared with the control data from the custom HW monitor and show an average error of 1.8% in "All data" for the four connection groups evaluated. Software and Custom HW monitor results for payload are within 1.4% from each other. It is also worth noting that the software cannot measure all paths such as SDRAM → MEs.

We can conclude that the P2P Investigator can clearly be used to give realistic approximations of the behavior of a real system, error being close to the error obtained in Section 7.

8.6. Area overhead

The total HW area of Trace Monitor on FPGA was 2506 Stratix II ALMs. The system in whole used 54,770 ALMs. Thus, only 4.8% monitoring area overhead was observed. Comparison of the areas of some design entities is shown in Table 6.

In comparison to proposal [18], we notice a somewhat similar HW area overhead. Their monitoring approach consumed from 1672 to 2628 LUTs on a Xilinx FPGA and from 6 to 25 RAM blocks, whereas our proposal consumed 2506 Altera ALMs and zero RAM blocks. On the other hand, we monitor a 4×4 mesh instead of a 2×2 mesh, and we monitor all of the links at the same time. The Ethernet controller and UDP/IP packetizer used by us are also probably at least an order of magnitude larger than the UART controller used in [18] due to higher complexity.

9. Conclusions

In this paper we presented Trace Monitor, a tool for evaluating, measuring and analyzing Network-on-Chips at a large scale of different accuracy levels at the same time. In the case study, we found that Trace Monitor is a very capable and versatile tool for analyzing an existing system on a real platform.

A trace of 5 s in length was recorded, occupying 43 MB hard disk drive space from the measurement computer, however, much longer traces can be recorded as easily. The record had accuracy of 500 clock cycles, allowing us to play back what happened inside the chip in slow-motion. The high time-domain accuracy allowed us to identify short command transactions from the bulk of image data. The system interfaced with external IO devices hard to simulate (Ethernet connection) and was run in real time at 25 MHz, even though it was monitored with Trace Monitor.

Accuracy can be further greatly increased, if needed, by slowing down the system clock more. We demonstrated this by slowing the

Table 7
Central facts about Trace Monitor.

Property	Value
Purpose	MPSoC & NoC monitoring on FPGA
Outcome	Hot spot detection, traffic model, etc.
Principle	Signal snooping
Interference	Non-interfering
Monitor data transfer	100 Mbps Ethernet
Major parts	HW monitor on FPGA, data collection and analysis software on PC
Software GUI features	Real-time view, playback, zoom, min/avg/max values, graphs, end-to-end data flow analysis
Temporal accuracy	Typically 1–50 k clock cycle (s)
Clock frequency	Typically 0.1–100 MHz
Area usage	Typically 1–4 k ALUTs (5%)
P2P error %	Typically 1–30%
Language	4 k LoC VHDL, 11 k LoC C++
Libraries	SFML [23]
Licence	LGPL
Available	http://www.tkt.cs.tut.fi/research/nocbench

clock down to 5 MHz, which still allows the usage of the video system as intended (with reduced framerate), but allowed us to increase the temporal accuracy of our monitor to 100 clock cycles. In this case, we recorded 20 s of trace, creating a 186 MB trace file.

We also presented a feature to automatically approximate the end-to-end IP interconnect traffic based on measurements at just link level in NoC, without monitoring addresses, to keep the hardware cost and monitor data amount low. By using three known test cases, we could show that our approximation showed average error of about 10%. Hence, the tool is suitable for creating traffic profiles for NoC benchmarking even without having any information of the application. We successfully demonstrated this feature in a real system.

Table 7 summarizes our proposal. Trace Monitor suite is available under the LGPL licence to freely use and modify on our website at http://www.tkt.cs.tut.fi/research/nocbench/.

Acknowledgement

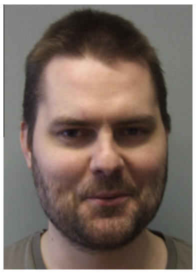
This work has been financially supported by the Academy of Finland. Authors wish to thank the anonymous reviewers for their valuable comments.

References

[1] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, Computer 35 (2002) 70–78.
[2] P. Guerrier, A. Greiner, A generic architecture for on-chip packet-switched interconnections, in: Design, Automation & Test in Europe, pp. 250–256.
[3] W. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: DAC, pp. 684–689.
[4] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, Surviving the SoC Revolution, Kluwer Academic Publishers, 1999.
[5] T. Bjerregaard, S. Mahadevan, A survey of research and practices of network-on-chip, ACM Computing Surveys 38 (2006).
[6] E. Salminen, On Design and Comparison of On-Chip Networks, Ph.D. thesis, Tampere University of Technology, 2010.
[7] N. Genko, D. Atienza, G. De Micheli, L. Benini, Feature – NoC emulation: a tool and design flow for MPSoC, Circuits and Systems Magazine IEEE 7 (2007) 42–51.
[8] A. Alhonen, E. Salminen, J. Nieminen, T. Hämäläinen, A scalable, non-interfering, synthesizable network-on-chip monitor, in: NORCHIP, 2010, pp. 1–6.
[9] A. Kulmala, E. Salminen, M. Hännikäinen, T.D. Hämäläinen, Evaluating SoC network performance in MPEG-4 encoder, The Journal of Signal Processing Systems for Signal, Image, and Video Technology (2008) 19.
[10] Z. Duoli, H. Ning, D. Gaoming, G. Luofeng, J. Jinghua, Design and evaluation of shared buffer based NoC, Computational Intelligence and Industrial Applications (2009) 321–325.
[11] S. Lotlikar, V. Pai, P. Gratz, AcENoCs: A configurable HW/SW platform for FPGA accelerated NoC emulation, in: VLSI Design (VLSI Design), 2011 24th International Conference, pp. 147–152.
[12] J. Nieminen, Traffic Generator Usage, User manual, Tampere University of Technology, 2010. <http://http://www.tkt.cs.tut.fi/research/nocbench>.
[13] L. Möller, L.S. Indrusiak, M. Glesner, NoCScope: a graphical interface to improve networks-on-chip monitoring and design space exploration, in: Design and Test Workshop (IDT), pp. 1–6.
[14] K. Holma, T. Arpinen, E. Salminen, M. Hännikäinen, T.D. Hämäläinen, Real-Time execution monitoring on Multi-Processor System-on-Chip, in: International Symposium on System-on-Chip, pp. 23–28.
[15] L. Fiorin, G. Palermo, C. Silvano, MPSoCs run-time monitoring through networks-on-chip, in: Design, Automation & Test in Europe 2009, pp. 558–561.
[16] Chapter 15: Design Debugging Using the SignalTap II Logic Analyzer in Quartus II Handbook Version 10.1, vol. 3, Altera Corporation, 2010.
[17] S. Tang, Q. Xu, A multi-core debug platform for NoC-based systems, in: Design, Automation & Test in Europe, 2007, pp. 1–6.
[18] R.B. Mouhoub, O. Hammami, Noc monitoring feedback for parallel programmers, in: Circuits and Systems, 2006 IEEE North-East Workshop on, pp. 141–144.
[19] DM9000A Ethernet Controller with General Processor Interface, datasheet, Davicom Semiconductor Inc., 2006.
[20] 10/100 Non-PCI Ethernet Single Chip MAC + PHY, datasheet, SMSC, 2006.
[21] DE2 Development and Education Board, User manual, Altera Corporation, 2007.
[22] Stratix EP1S80 DSP Development Board, Data sheet, Altera Corporation, 2004.
[23] SFML home page, 2011. <http://www.sfml-dev.org/>.
[24] Xiph.org Test Media, 2010. <http://media.xiph.org/video/derf/>.

Antti Alhonen works as research assistant in the Department of Computer Systems (DCS) at Tampere University of Technology (TUT). His research interests include NoC, FPGA design, and electronics.

Erno Salminen received his MSc and PhD degrees in 2001 and 2010 from TUT. Currently he works as research fellow at DCS and acts as a chairman of OCP-IP Network-on-Chip benchmarking workgroup. His research interests include design of NoC, MP-SoC, and digital logic in general. He has (co-) authored over 60 scientific articles and received the OCP-IP Contributor of the year award in 2009.



Lasse Lehtonen works as research assistant in the Department of Computer Systems (DCS) at Tampere University of Technology (TUT). His research interests include NoC, FPGA design, and SystemC.

Timo D. Hämäläinen received the M.S. degree in electrical engineering from Tampere University of Technology (TUT), Finland, in 1993, and Ph.D. degree in electrical engineering from TUT, Finland, in 1997. He is Professor at Department of Computer Systems (DCS) at TUT from 2001. He is author of over 50 journal and 180 conference publications and holds several patents on wireless systems. He heads the DACI research group that focuses on wireless sensor networks as well as multi-processor System-on-Chip architectures, modeling and design tools.