Microprocessors and Microsystems 37 (2013) 446–459

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 12 June 2012

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

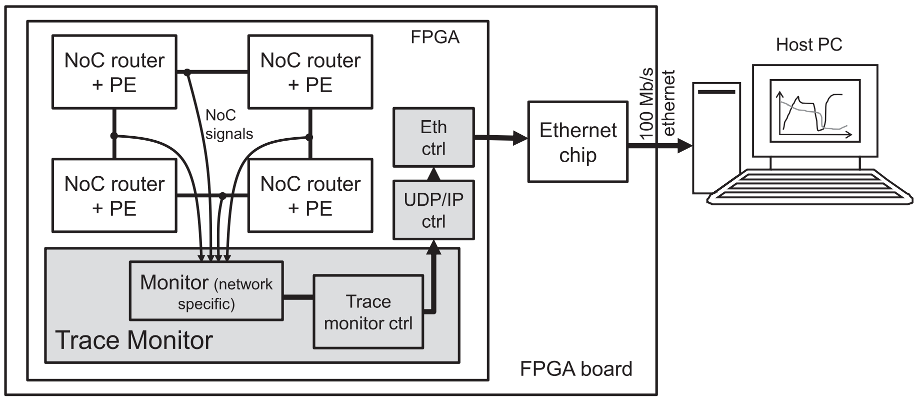

Fig. 1. An example instantiation of Trace Monitor (shaded blocks) in a 2 × 2 mesh Network-on-Chip on FPGA.

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