MPSoCs Run-Time Monitoring through Networks-on-Chip

Leandro Fiorin
ALaRI, Faculty of Informatics
University of Lugano
Lugano, Switzerland
Email: fiorin@alari.ch

Gianluca Palermo, Cristina Silvano
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Milano, Italy
Email: {gpalermo, silvano}@elet.polimi.it

***Abstract*—Networks-on-Chip (NoCs) have appeared as design strategy to overcome the limitations, in terms of scalability, efficiency, and power consumption of current buses. In this paper, we discuss the idea of using NoCs to monitor system behaviour at run-time by tracing activities at initiators and targets. Main goal of the monitoring system is to retrieve information useful for run-time optimization and resources allocation in adaptive systems. Information detected by probes embedded within NIs is sent to a central unit, in charge of collecting and elaborating the data. We detail the design of the basic blocks and analyse the overhead associated with the ASIC implementation of the monitoring system, as well as discussing implications in terms of the additional traffic generated in the NoC[1].**

I. Introduction

Next generation MPSoC platforms will integrate in the same embedded device a large number of processing cores, storage elements and I/O peripherals, interconnected by Networks-on-Chips (NoCs) [1], [2], foreseen as the design paradigm to overcome efficiency problems of traditional solutions for inter-core communication based on shared bus. In order to optimize the utilization of available resources in such complex platforms, the use of adaptive and reconfigurable platforms implementing a run-time management of components and resources has been proposed [3], [4]. Fundamental components of run-time management systems are the monitoring subsystems, that must be able to detect variations in the run-time conditions of users and applications requirements, and to allow run-time managers to perform an appropriate and optimized allocation of resources.

In this work, we propose the idea of monitoring run-time system activities in adaptive NoC-based MPSoC platforms through the observation of transactions performed on the communication subsystem. As central element of architectures based on the communication-centric paradigm [1], NoCs are the ideal mean to collect information about cores, and more general system behaviour. In particular, probes collecting information about system activity are implemented within OCP/IP [5] compliant Network Interfaces (NIs). A central unit is in charge of collecting run-time information and adopt the suitable strategy for optimizing the use of system resources. Probes, working in parallel with protocol translation, do not influence core operations, and a relatively small overhead is paid in term of area consumed by the monitoring system, as well as in the amount of data transmitted by probes to the central unit.

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents an overview of the proposed NoC monitoring architecture, while Section IV provides implementation details of the probes deployed in the system. Section V presents synthesis results, and discusses overhead in NoC traffic due to the monitoring system. Finally, Section VI presents conclusions.

II. Related Work

This Section overviews related work on monitoring of NoCs. Methodologies for real-time debugging of NoC-based SoCs are presented in [6], [7] . Authors propose a NoC monitoring systems composed of configurable monitoring probes attached to NIs or to routers and NIs. Associated programming models are discussed, as well as monitoring traffic management strategies. In [8], several design alternatives for NoC monitoring systems are discussed, in particular focusing on the interconnection of the monitoring resources.

In [9], a debugging approach based on probes inserted between the cores and their network interfaces is proposed. A system-level debug agent controlled by an off-chip multi-core debug controller collects information about system activities, providing in-depth analysis features such as NoC transaction analysis, multi-core cross-triggering and global synchronized timestamping.

While some of the concepts developed for testing and debugging NoCs can be used in our case, monitoring for run-time management of system information presents unique challenges related to the implementation of intelligent probes that should be able to minimize the impact of monitoring on the overall system behaviour. Our work can be considered similar in some concepts to [10], in which link utilization is monitored to implement a strategy for controlling congestion in on-chip networks, or [11] and [12], where monitoring of NoCs to detect security violations in suggested. However, we differentiate from [10], [11], [12] in the fact that we propose

[1]This work has been partially funded by FP7-ICT-2007-1-216693-MULTICUBE project.

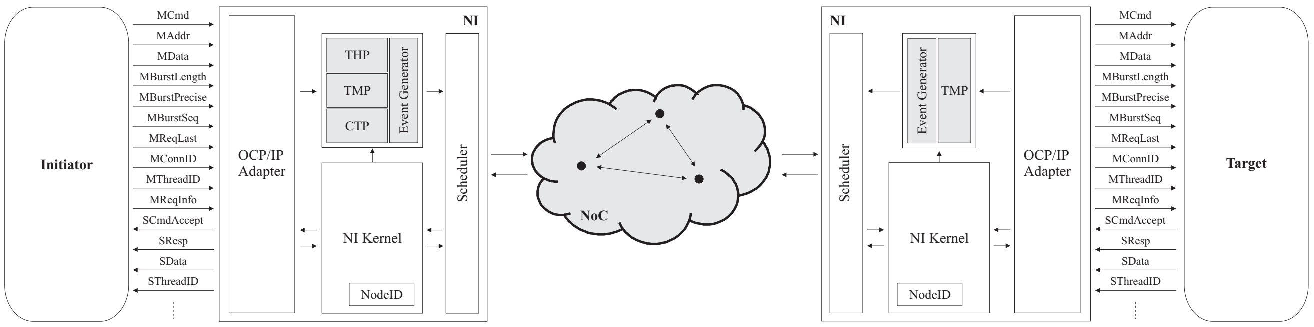

Fig. 1. Monitoring architecture including the proposed probes

a more general approach that can be considered valid for all type of NoC based adaptive systems.

III. Overview of Monitoring Architecture

A. General Architecture

In our work, we refer to a NoC-based MPSoCs with memory-mapped cores. The NoC implements a transaction-level protocol [13], with cores acting as *initiators* or *targets* of transactions. We assume a source-based routing, and a wormhole control flow strategy. The monitoring system is mainly composed of three elements: *probes* (P), the *communication infrastructure* (NIs and Routers (R)), and a *Run-time Manager* (RTM).

In probes, transactions are analysed when requested by the core (in case of probes at initiators' NI), or when it receive them (probes at targets' NI), in parallel with operations performed by the NI kernel for the protocol translation from the OCP/IP interface.

Probes generate *Events* (definition is given Subsection III-C) to communicate to the central unit (RTM) the information collected. Without loss of generality, we assume in our architecture a *Master/Slave configuration* [3], where a master processor is in charge of acting as RTM and collecting and elaborating information from the several probes distributed in the system.

B. Types of Profiling

From the analysis of possible transactions requested by a core acting as initiator (or target), we identified three general categories of profiling that can be extracted by our monitoring system:

- *Throughtput*: amount of traffic generated and received by cores is monitored in order to measure communication bandwidth required by applications;
- *Timing/latency*: time can be monitored at different levels and different purposes. We can distinguish between time monitoring involving *communication* and *computational* aspects of applications;
- *Occurrences*: counters for measuring the occurrence of a particular event can be placed within the NI to monitor the number of requests of access to a specific address range and its related memory-mapped element (e.g. core, memory, I/O, etc.).

C. Event messages

As definition of event message, we comply with what discussed in [6]: an *Event* can be represented as a tuple composed of an *Identifier*, a *Timestamp*, a *Producer*, and several *Attributes*.The *Identifier* identifies events of a certain class of events, and it is unique for each class. The *Timestamp* defines the time at which the event was generated by the producer, identified by the field *Producer*. Attributes are represented in the form $Attribute = (Attribute\_Identifier, Value)$, and the type of attribute and its value depend on the type of the event generated.

In our case, *Identifier* specifies the type of information detected by the probe, while *Producer* specifies the identifier of the NoC node. While not using *Timestamp* (we assume service packets transmitted in order and with prioritized or predictable latencies - statistics about global timing are therefore generated inside the RTM), *Attributes* depend on type of probe and information collected.

IV. Implementation

In this Section, we present implementation details of the probes embedded in NIs, able to provide measurements described Section III. Fig. 1 shows NIs (at initiator and target) embedding the probes we propose (in grey in the figure). The *Throughput Probe* (THP) provides measurements about the amount of traffic generated by cores. *Timing Probes* (TMP) provide indication about time measurements, while the *Counter Probe* (CNP) about occurrences of a specific transaction. The *Event Generator* is triggered by the probes and generates the event packets sent to the RTM to communicate values and information measured.

A. Throughput Probe

Fig. 2 shows architectural details of the Throughput Probe. This probe keeps track of the amount of traffic to/from a selected range of addresses generated by an initiator (thread running on a processing element). The bandwidth is calculated by considering the length of data loaded/stored per each connection by every initiator during a defined time window. The length of the time window can be set by the RTM at run-time or by the designer at design time. The time window is implemented by using a programmable counter. In general, the

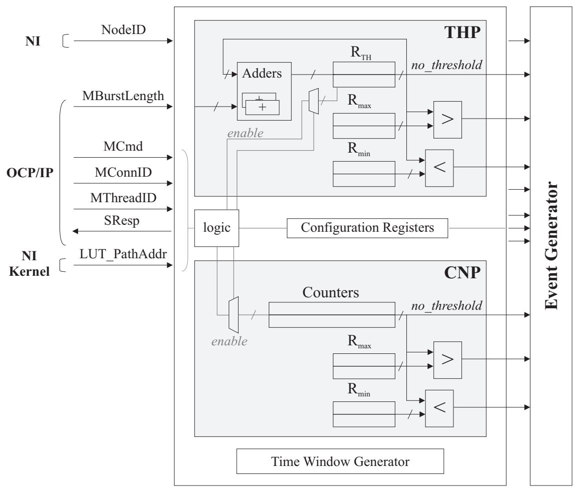

Fig. 2. Implementation details of the Throughput Probe (THP) and the Counter Probe (CNP)

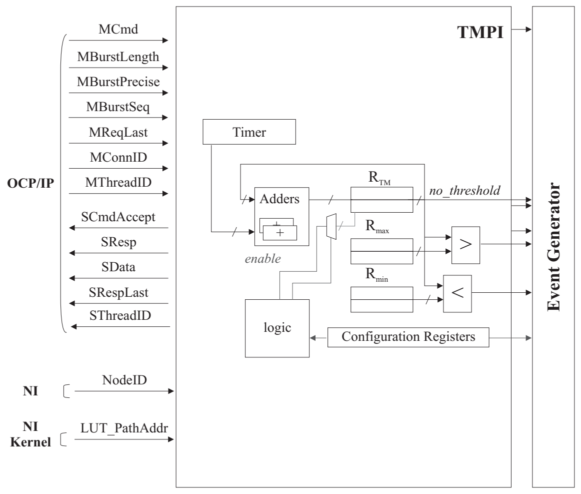

Fig. 3. Implementation details of the Timing Probe at initiator (TMPI)

probe is realized by using a combination of registers ($R_{TH,i}$) where the collected run-time information is stored, and circuits to trigger the *Event generator*. Through the RTM, each register is associated to a specific connection and is selected by the combination of the OCP/IP signals which identify the initiator (*MCmd*, *MConnID*, and *MthreadID*, respectively indicating the type of operation (load/store), the connection identifier, and the thread identifier) and the *LUT_PathAddr* signal, coming from the look up table (LUT) of the NI Kernel. *LUT_PathAddr* drives the RAM of the LUT looking up the memory address present as input in the OCP/IP *MAddr* signal to retrieve the path to be inserted in the header of the packet.

When a core requests a transmission by driving the *MCmd* signal of the OCP/IP interface of the NI, the length of the data transmitted or received (given by the OCP/IP signal *MBurstLength*) is added to the value already stored in the selected $R_{TH,i}$ register. At the end of the time window, the *Event Generator* is triggered and creates a packet to communicate the collected information to the RTM.

In order to potentially reduce the amount of data transmitted by the *Event Generator*, we implemented the possibility to transmit only information about data traffic exceeding or under a certain thresholds set by the RTM, and stored in registers $R_{max,i}$ and $R_{min,i}$.

B. Timing Probes

Fig. 3 shows architectural details of probes monitoring timing related aspects of applications. We distinguish probes implemented at initiators (TMPI) and targets (TMPT). Only timing probe at initiator is shown in Fig. 3. TMPIs monitor time needed by transactions to be completed. We assume a transaction being composed of a request sent by the initiator to the target, and of an acknowledgement signal sent back by the target to the initiator. The transaction can be considered complete when the initiator receives the last part of the packet containing the acknowledgement message. When the initiator begins a transaction, the probe detects a change in the OCP/IP signal *MCmd*, recording a timestamps in the associated register $R_{TM,i}$. Therefore, the probe waits for the transaction to be finished, i.e., when the acknowledgement message is received by the NI of the initiator and, in case of *load* requests, the last information received is passed to the initiator (corresponding to a change of state of the OCP/IP signal *SResp*). When the transaction ends, the value previously stored in $R_{TM,i}$ is subtracted from the current time, and the *Event Generator* is triggered to communicate the event and the value collected to the RTM.

The second type of timing monitoring probe we propose is in charge of measuring time employed by targets to execute their tasks. Differently from TMPs, these probes are implemented at targets' NIs. When receiving the requests of a new transaction, and on the change of the OCP/IP signal *MCmd* (*Master* signal driven is this case by the target's NI), the timestamp is recorder in register $R_{TM,i}$, as well as the information about the initiator of the transaction taken from the header of the packet. Once the execution is completed, the time recorded is subtracted from the current time, and the *Event Generator* is triggered to generate the event packet.

As in the case of THPs, we implemented the possibility to communicate the event and the related information only if the time measured is outside boundary values stored in two registers ($R_{max,i}$ and $R_{min,i}$).

C. Counter Probe

Counter Probes (CNPs) (located at the initiator's NI and shown in Fig. 3), monitor the number of transactions directed towards a specific target. They are mainly composed of counters, incremented when a new transaction starts. Similarly to THPs, they collect data during a time window and are assigned by the RTM to initiator - target tuples. They are selected by the combination of *MCmd*, *MConnID*, *MthreadID*, and *LUT_PathAddr* signals. At the end of the time window, the *Event Generator* is triggered and creates a packet to communicate the collected information to the RTM. A system with threshold has been implemented as well, to reduce the amount of traffic transmitted to the RTM.

TABLE I
AREA COST OF PROBES

	Area Percentage (%)
Probes at Initiator	100
THP	26.38
TMP	26.73
CNP	20.62
Event Gen.	26.27
Probes at target	100
TMP	75.13
Event Gen.	24.87

V. SYNTHESIS AND EXPERIMENTAL RESULTS

In this Section, we present synthesis results for the implementations of the probes presented in Section IV, obtained by using the 0.13μm HCMOS9GPHS STMicroelectronics technology library. The synthesis was optimized for a clock frequency of 500 MHz. In Table I, we show the area percentage values of the proposed components, in the case of the implementation of a monitoring system able to monitor 4 connections with THP, TMP and CNP probes. The value shown for the THP includes also the programmable counter used to generated the time window, employed also by CNPs to time the generation of their data. Area occupied by components increases linearly with the maximum number of connections that it is possible to monitor at the same time (the graph is not shown for space limitation). This results is expected, being in fact the area proportional to the number of registers recording the monitored information, directly proportional to the number of possible connections. Considering as reference [13], the total area overhead of the system is around 27% of the area occupied by the NI for the probes implemented at the initiator, while around 10% for those implemented at the target.

In order to avoid the delay of monitored information due to possible network congestion, the transmission of packets from probes must be guaranteed through the use of priority communication or guaranteed throughput services [13]. Therefore, the communication system should be over-designed in order to allocate the monitoring traffic. The bandwidth required by the proposed probes can be evaluated considering that in case of THPs and CNPs an event packet will be produced every time the time window expires, while in case of TMPs at the end of a transaction or a task execution. Therefore, for THPs and CNPs the bandwidth can be expressed as:

$$BW_{THP} = \frac{n\_bit_{header} + (n\_bit_{info} * n\_conn)}{tw_{clk}} * f_{NoC} \quad (1)$$

where n_bit_{header} is the number of bits needed by the packet header, n_bit_{info} the length of the information transmitted for each connection, and n_conn the number of connections monitored by the probe. tw_{clk} is the length of the time window expressed in number of clock cycles, while f_{NoC} the frequency at which the NoC transmits. For TMPs, the bandwidth is equal to:

$$BW_{TMP} = \frac{n\_bit_{header} + n\_bit_{info}}{tran_{clk}} * f_{NoC} \quad (2)$$

where $tran_{clk}$ is the time needed for executing the transaction or the task. While in the first case the value of the bandwidth can be calculated once that the length of the minimum time window to be used has been decided, the value of BW_{TMP} should be estimated considering the minimum transaction time foreseen for the applications.

VI. CONCLUSIONS

In this paper, we presented the basic blocks of a run-time monitoring system for NoC architectures, whose goal it to monitor run-time system activities through the Network Interface. Information collected is sent to a central unit (RTM), in charge of elaborating data received. The monitoring system is targeted to be used with adaptive and reconfigurable platforms, in order to provide useful information in the optimization of the use of system resources in environments where user requirements are not fully known at design time, or in which multiple applications are competing for same shared resources. We proposed the use of "intelligent" probes to monitor throughput, time and occurrence of events in applications, and we presented architectural details of their implementation. We analysed the overhead associated with an ASIC implementation of the monitoring system, as well as providing an estimation of the traffic generated by probes.

REFERENCES

[1] L. Benini and G. De Micheli, "Networks on Chips: A New SOC Paradigm," *IEEE Computer*, 2002.

[2] W. J. Dally and B. Towles, "Route packets, not wires: on-chip inteconnection networks," in *Proc.of DAC'01*, 2001.

[3] V. Nollet, D. Verkest, and H. Corporall, "A Quick Safari Through the MPSoC Run-Time Management Jungle," in *Proc. of ESTIMedia'07*, 2007.

[4] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-Time Management of a MPSoC Containing FPGA Fabric Tiles," *IEEE Trans. on VLSI Systems*, vol. 16, no. 1, pp. 24–33, January 2008.

[5] *Open Core Protocol Specification 2.2*.

[6] C. Ciordas, T. Basten, R. Radulescu, K. Goossens, and J. Van Meerbergen, "An Event-Based Monitoring Service for Networks on Chip," *ACM Trans. on Design Automation of Electronic Systems*, vol. 10, no. 4, pp. 702–723, Oct. 2005.

[7] B. Vermeulen, K. Goossens, and S. Umrani, "Debugging Distributed-Shared-Memory Communication at Multiple Granularities in Networks on Chip," in *Proc. of NOCS'08*, 2008.

[8] C. Ciordas, K. Goossens, R. Radulescu, and T. Basten, "NoC Monitoring: Impact on the Desing Flow," in *Proc. of ISCAS '06*, 2006.

[9] S. Tang and Q. Xu, "A multi-core debug platform for noc-based systems," in *Proc. of DATE'07*, 2007.

[10] J. van Den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," in *Proc. of DATE '07*, 2007.

[11] L. Fiorin, C. Silvano, and M. Sami, "Security Aspects in Networks-on-Chips: Overview and Proposals for Secure Implementations," in *Proc. of DSD'07*, 2007.

[12] J. P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "NoC-centric security of reconfigurable soc," in *Proc. of NOCS'07*, 2007.

[13] A. Radulescu, J. Dielissen, S. G. Pestana, O. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, January 2005.