Transaction Monitoring in Networks on Chip:
The On-Chip Run-Time Perspective

Calin Ciordas [†]   Kees Goossens [‡]   Twan Basten [†]   Andrei Radulescu [‡]   Andre Boon [†]

[†] Eindhoven University of Technology
Design Methodology for Electronic Systems
PO Box 513, NL-5600 MB Eindhoven
{c.ciordas,a.a.basten@tue.nl}
{a.g.boon@student.tue.nl}

[‡] Philips Research Laboratories
Embedded Systems Architectures on Silicon
Prof. Holstlaan 4, NL-5656 AA Eindhoven
{kees.goossens@philips.com}
{andrei.radulescu@philips.com}

Abstract

Networks-on-chip (NoC) are a scalable interconnect solution to multiprocessor systems on chip (MPSoC). NoCs transport data in packets which are fragments of transactions, such as read and write actions of IPs. For debug purposes, reconstructing transactions at run-time is essential. Run-time analysis of the NoC behavior at transaction level makes the complete MPSoC easier to understand. We present a NoC analyzer able to monitor NoC transactions at run-time. The proposed hardware transaction monitor is able to reconstruct on-chip, at run-time, NoC transactions from bit-level intercepted router link communication. Four NoC analyzer modes are detailed raising the abstraction level gradually from physical raw to logical connection-based, transaction-based and abstract transaction event-based. Each mode is analyzed for area and bandwidth in an experimental setup based on several Æthereal NoC designs. A transaction monitor has an area cost of $0.026mm^2$ in a $0.13\mu m$ CMOS technology, and for several MPEG/audio case studies, the entire monitoring system adds an average of 5% to the NoC area. We show the versatility of our NoC analyzer by run-time monitoring user connections and the Configuration Master IP in the NoC.

1 Introduction

1.1 Problem Statement

Due to ever increasing technological advances, very complex large scale system on a chip (SoC) designs are becoming possible. Each new SoC generation integrates more processing elements, more features, and offers increased functionality. With this ever increasing complex-ity, the step of getting the design working properly becomes increasingly difficult, leading to a need for better debug solutions. This points to the need for providing the necessary controllability, and in particular the observability of internal operations of a complex SoC. Observability of current SoCs is becoming a major bottleneck as the amount of embedded cores and critical internal signals per I/O pin ratio increases.

This has led to the addition of dedicated on-chip resources which support functional analysis in order to increase SoC observability. Existing techniques such as system simulations and state dumps using JTAG/boundary scan chains are complemented by this additional infrastructure. This debug instrumentation has become common at core level [2], and bus-level [21]. Increasingly, a system-centric debug infrastructure, supporting multi-core system-level diagnosis and analysis, like ARM's Coresight [1] and First Silicon's OCI [15], is gaining momentum. Computation and communication observability in current SoCs are a recognized must, and its importance is growing.

Networks on chip (NoCs) [3, 8, 10, 12, 16, 13] are the preferred interconnect solution for large scale multiprocessor SoCs. With the emerging trend of NoC-based SoCs, system busses are replaced by NoCs. The on-chip communication becomes more sophisticated, relying on run-time programmable solutions. While the on-chip debug infrastructure at core level can be reused in NoC-based SoCs, standard bus monitors no longer work with such systems, as multiple truly parallel communication paths between the IPs exist, as opposed to a (single) centralized bus in bus-based SoCs. This led to the addition of debug infrastructures for the NoC interconnect [5]. This debug infrastructure copes with the run-time communication observability in NoC-based SoCs.

Even if basic observability of NoCs is achieved in the form of bits, it is still the question of what the monitored

bits mean, e.g. a configuration packet for the NoC itself or a write action issued by an IP connected to the NoC. The abstraction level at which monitoring can be done can vary from the bit-level to the level of inter-IP communication. For NoCs, the interconnected IPs communicate to each other by means of transactions, which are read and write actions from IPs. To increase the operational speed of system-level debugging, the NoC debugging infrastructure must bring the abstraction level of the monitored data at transaction level, and allow run-time transaction monitoring in particular, at a reasonable cost.

Transactions are composed of one or more messages, e.g. a read action from an IP, and the data coming back as the result of it. In efficient packetization schemes, e.g. in the Æthereal NoC, the transaction components are considered part of a data stream which is packed at Network Interfaces (NI) into packets without consideration for alignment of packets and transaction components. The monitored bit stream (raw data) corresponding e.g. to a monitored router link contains multiple time-multiplexed connections of different traffic classes, e.g. guaranteed throughput (GT) and best effort (BE). These connections are established between different pairs of spatially distributed IPs. Identifications of transactions out of this monitored bit stream is a challenging problem.

1.2 Related Work

There has been a lot of work in providing observability for bus-based systems. ARM's Coresight [1] technology combines ETMs [2] for ARM cores, with the AHB Trace Macrocell which gives visibility on AMBA AHB busses, offering an understanding of multilayer-bus utilization and visibility of accesses to memory areas. First Silicon's on-chip instrumentation technology (OCI), provides on-chip logic analyzers [21] for AMBA AHB, OCP, and Sonics SiliconBackplane bus systems. These solutions allow the user to run-time capture bus activity, and can be combined in a multicore-embedded debug system [15] with in-system analyzers for cores, e.g. for MIPS cores. Although state-of-the-art, both solutions are not able to cope with a NoC-based SoC.

While the test and verification implications of using NoCs have been inventoried in [22], in the research community, focus is on the design [4, 14, 8, 12, 16, 10], analysis [9] and use [18] of NoCs. Currently, there is little support for run-time communication observability in NoC-based SoCs in general.

[5] proposes the concepts of a generic NoC Monitoring Service (NoCMS) to cope with communication observability in NoC-based SoCs and advocates the use of abstract events in reducing the size of the monitored data. The NoCMS can be instantiated at design time together with the NoC by means of a monitoring-aware NoC design flow [7].

The impact of run-time monitoring on NoC design flows is shown in [6], in case of shared or separated NoCs for application and monitoring data. As far as we are aware, there is no support for transaction-level monitoring in existing NoCs.

[20] proposes on-chip run-time collection of traffic statistics at network interfaces to optimize the usage of communication resources in a NoC-based SoC; this is done using the centralized resource management of [19].

1.3 Contribution

This paper presents the *concepts of monitoring NoC transactions on chip at run-time*. It shows how we can reconstruct the transaction view (read and write actions from IPs) from the raw, low-level monitored data. The raw data can be monitored at any router, a module which has no understanding of any notion of transaction. We show that *transactions can be reconstructed* regardless the way in which packetization has been done in network interfaces, covering all existing NoC packetization schemes.

We further propose a *hardware monitor which supports on-chip transaction reconstruction and several intermediate levels of abstractions needed for it*, raising the abstraction level gradually from physical 'raw' to logical connection-based, transaction-based and abstract transaction event-based. We further show that such monitoring is *feasible area wise*, as the transaction monitor supporting both GT and BE traffic classes is 0.026mm^2 in a 0.13μm CMOS technology, small even when compared with a corresponding 0.13mm^2 combined GT/BE NoC router in the same technology.

We call the resulting monitoring system a *NoC analyzer* and the supported abstraction levels, *analyzer modes*. An analysis of the generated traffic for each of the analyzer modes is presented in the context of four realistic Æthereal NoC designs based on an MPEG codec, underlining advantages and potential problems for each of them. An example of monitoring NoC configuration master activity in the connection-based mode is shown.

1.4 Overview

Section 2 presents the Æthereal NoC in general and the transaction-based communication model and packetization in particular. The NoC analyzer and the two basic data transport scenarios used throughout the paper are introduced in Section 3. In Section 4 we describe transaction monitoring by presenting the four NoC analyzer modes and the underlying monitor architecture. The area and traffic implications are presented in Section 5 and the conclusions in Section 6.

2

2   Æthereal Experimental Setup

We have used the Æthereal NoC [10] as an example for our work but all the concepts presented are more general and can be reused for other NoCs. Several NoCs [3, 8, 10, 12, 16, 13] have been proposed. In general they are composed of network interfaces (NI), which implement the NoC interface to IPs, and of routers which transport data from NI to NI.

The Æthereal NoC runs at 500 MHz and offers a raw link bandwidth of 2GB/s in a $0.13\mu m$ CMOS technology. Æthereal offers transport-layer communication services to IPs, in the form of connections, comprising best-effort (BE) and guaranteed-throughput (GT) services. Guarantees are obtained by means of TDMA slot reservations in NIs. Æthereal NoC instances are reconfigurable at run-time. This is achieved by programming the NIs using standard memory-mapped I/O ports. The current setup uses centralized programming of the NoC and source routing, but distributed solutions are also possible.

Transactions (reads and writes) are performed on connections. One transaction comprises one or more messages. Messages are differentiated as request and response messages. E.g. a request message can be the write message depicted in Fig. 1. A response message is for example data coming back as a result of a read operation, or an acknowledgment as a result of a write operation.
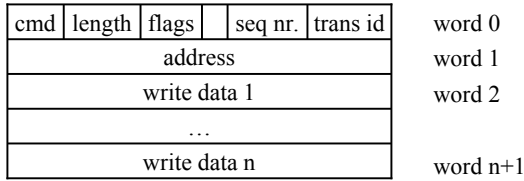

Figure 1. Æthereal message format

The NIs convert these messages into packets, by chopping them into pieces of a maximum length and adding a header to each of these pieces, resulting in packets. Packets may be of different lengths. The packet header consists of: (1) a path to the destination NI as a sequence of router output ports, (2) a queue identifier at the destination NI, and (3) piggybacked credits for end to end flow control. An example packet is shown in Fig. 2.

Packets are further split into flits, the minimum transfer unit between hops. The flit format is presented in Fig. 3, with the mention that this particular flit contains a packet header as the first word. One flit corresponds to one TDMA slot. Note that the GT packet presented in Fig. 2 corresponds to three consecutively allocated slots. One flit comprises three 32-bit words. For each of the three words there are two bits of sideband information. The first two sideband information bits, id in Fig. 3, show whether the flit is BE or
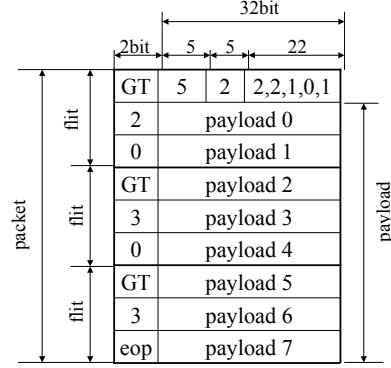

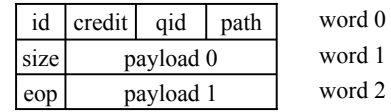**Figure 2. Æthereal packet example**

Figure 3. Æthereal flit format

GT and whether it contains a packet header or not. The second two bits show the number of valid words in the flit. The last two bits show the end of packet.

3   NoC Analyzer

For the basic access to NoC flits, we have used the NoC Monitoring Service (NoCMS) of [5]. The NoCMS is offered by the NoC in addition to the communication services offered to the IPs. It consists of configurable probes attached to NoC components. The generic probe modular design comprises three parts: the sniffer (S), the event generator and the monitoring network interface (MNI). The MNI can be a separate NI or can be merged with an existing NI.

Our specialized transaction monitor replaces the generic event generator of [5]. Fig. 4 presents an MPEG codec with a 2x3 NOC with its NOC Analyzer. All routers of the NoC example from Fig. 4 have a transaction monitor (TM). The general process works as follows; the sniffer obtains the raw flits (bit-level) from the NoC components and passes this information to the transaction monitor. The transaction monitor performs local processing, specific to each of the analyzer modes. It then forwards the results to the MNI. The MNI packetizes the result as payload and sends them over the network to the Monitoring Service Access point (MSA), over a previously established monitoring connection, just like any other data in the NoC. The monitoring connections can be either BE or GT. We use the same NoC for monitoring as well as for the user traffic because this solution allows a logical, dynamic partitioning of NoC resources; resources can be used for monitoring when needed, and freed when not. Note that a separate interconnect for monitoring may
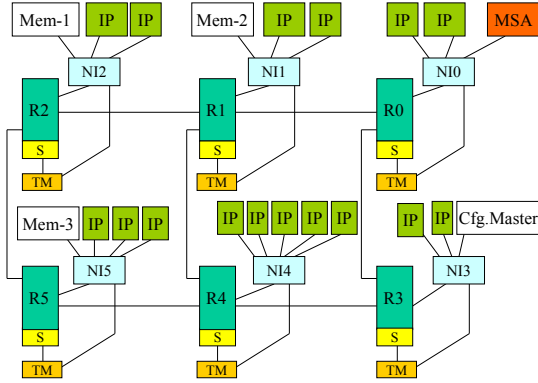
Figure 4. 2x3 MPEG codec with NOCMS

be used as well [6].

The NoC analyzer has to be able to check the functional details of user traffic from the observed router link data, at different levels of abstraction. For this, the transaction monitor architecture is defined. A schematic of the transaction monitor internal architecture is depicted in Fig. 5. All router links are sniffed by the sniffer which provides this info to the link selection block. In the link selection block one link is selected for further analysis. Then, there are *four transaction monitor processing blocks*. One or more of these blocks can be enabled and configured through the *enable/configuration block*. Two ports connect the transaction monitor to the MNI, one slave port (SP in Fig. 5), for programming the transaction monitor, and one master port (MP in Fig. 5), for sending the transaction monitor data to the MSA. Transaction monitors are programmed using memory-mapped I/O, by means of write transactions. Each processing block corresponds to one of the analyzer modes that we have identified. Each analyzer mode is detailed in the following section.

If the sniffed data is memory-mapped, e.g. the MSA is a memory, the transaction monitor uses write transactions to send the data to the MSA, and a command and address have to be added to the transaction monitor data. Therefore, a write transaction from a transaction monitor to an MSA will always contain a command, an address, and the useful payload. In the remainder we refer to this as the *memory-mapped* scenario.

If the sniffed data is not memory-mapped, e.g. the MSA is another IP which takes care of streaming the data off chip, no commands and addresses are added to this data. Therefore, a peer-to-peer streaming data transaction from transaction monitor to MSA will contain just the useful payload. In the remainder we refer to this as the *streaming data* scenario. We have implemented both *memory-mapped* and *streaming-data* scenarios in our simulator.
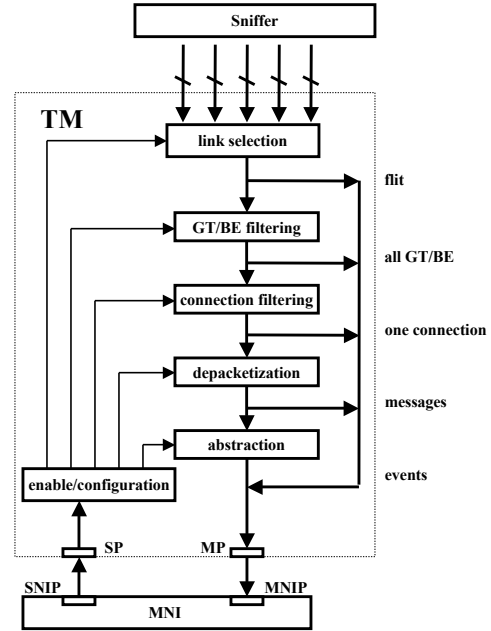

Figure 5. Transaction Monitor Architecture

4 Analyzer Modes

4.1 Raw Mode

In the raw mode the analyzer provides full observability on all bits passing a certain physical link. The link desired can be selected from all router links at run-time through the enable/configuration block. The link selection block provides at its output all flits passing one link. These flits can be part of different connections as TDMA is used for every link. The flits can be directly forwarded to the MNI or passed as input to the GT/BE filtering block. Looking only at the flit structure and not beyond we can do only limited filtering. Local filtering is possible based on the traffic category. For the Æthereal NoC we are able to filter GT or BE traffic from the raw flits. This is made possible by the 2-bit sideband information (see Fig. 2) of each flit which specifies whether the flit is GT or BE. The resulting flits are forwarded to the MNI. The MNI packetizes the flits as payload of a write transaction (memory-mapped scenario) or a data stream (streaming-data scenario).

A potential problem may arise: assuming for a certain link that utilization is very high, and that we do raw sniffing, the sniffed data has to be sent over a connection to the MSA. Due to the packetization overhead (the packet headers added to the useful sniff payload), the total can be more than the physical link bandwidth, making the transport of the sniffed data impossible. Filtering whether the traffic is GT or BE can alleviate this problem in certain cases, but not always solve it.

The raw mode is useful in case all details of the flits are important to be examined. Bit-level details can be inspected. In practice it works for low bandwidth connections, or for short snapshots of high bandwidth connections.

4.2 Connection-based Mode

As sending the raw sniffing results to the MSA cannot always work in practice, a more advanced mode is needed. In the connection-based mode the analyzer provides full observability on all bits of a selected connection, raising the abstraction level from physical raw to logical connection. The transaction monitor must allow further filtering of the sniffed data, besides the traffic class (e.g. BE/GT) in order to reduce the traffic from monitoring probe to MSA. All the NoC user traffic goes over connections, and connections share links based on a TDMA scheme. Filtering of the sniffed data is possible if a certain connection can be identified from the other connections sharing the same link.

The connection filtering block of the transaction monitor uses as input the output of the GT/BE filtering block. Connection identification can be done for the Æthereal NoC by means of the queue identifier and path; this pair uniquely identifies a connection. Both have to be programmed in the transaction monitor using the transaction monitor's slave port SP. Both the queue identifier and path can be found in the header of packets, see Fig. 2. A packet header can be identified by the 2-bit sideband information of each flit. Further more, a packet header is always the first word in the flit. Once a header is intercepted the queue identifier of the destination queue in the NI, and the path to that NI, can be extracted from the header. The transaction monitor must have the desired connection set and compare the value stored locally with run-time values from the headers. Once a match is found we have identified a packet belonging to the desired connection. The resulting packets are forwarded to the MNI.

Note that the proposed transaction monitor enabled in the connection-based mode is in the current setup able to monitor a single connection at a given time. However, the extension to support the monitoring of multiple connections at a given time is straightforward.

The connection-based mode is useful in case all details of a certain connection have to be examined, e.g. packet headers or connection utilization. The previously mentioned problem of exceeding the physical link bandwidth is still possible although this would require extraordinary circumstances, e.g. all slots reserved for a single connection. This is unlikely, and this mode is feasible in practice.

4.3 Transaction-based Mode

In the transaction-based mode the analyzer raises the abstraction to transaction level and provides bit-level full observability on transactions over a certain connection. This implies full observability of all transaction components, which are messages. A full transaction may involve a single message (e.g. a single write without acknowledgement) or multiple messages. The case of transactions using multiple messages possibly on multiple connections, involves the combination of data from multiple transaction monitors or from one transaction monitor at different points in time, which can for example be done at the MSA.

Being able to identify all the flits of a certain connection, the next step is therefore to identify messages belonging to the connection. This allows to see, from within the NoC, when a write or a read message has been issued and from where or to which of the IPs or memories, providing a transaction level view.

The main problem is how to identify the messages. It is difficult to detect the start of a message, because messages are considered payload and are packed in packets without any alignment. The routers, where monitoring is done, have no notion of messages. A packet may contain a single message, part of a message, or parts of multiple messages. It is therefore complex to see where a message starts just looking at a packet.

Message identification requires depacketization, a procedure usually done at the NI at the receiving side, at every slave NI port (SNIP). Hardware modules for depacketization are available for Æthereal. These modules assume that the first packet over a connection carries the first message header, immediately after the packet header. From there they only count the number of words in the received message, knowing that after it there is a new message header. Having detected the start of the message, the rest becomes simple. In the majority of existing NoCs, the size of the message is coded in the first word of the message, the exact place depending on the exact protocol message format. So, if the message start is detected, the rest of the message can be obtained by counting the words in the following sniffed flits belonging to the same connection, till the message size is reached. Counting of the message words does not take into account the headers of the packets involved, which are discarded. However, the main difficulty is detecting the start of the message.

There are three possible situations, covering all existing NoCs, see Fig. 6, with regard to alignment of packet header and message header, each of the situations having pros and cons regarding the NoC design:

(A) *The NoC does not distinguish between messages and packets, message/packet correspondence is one to one, see Fig. 6A.* As previously explained, a packet header
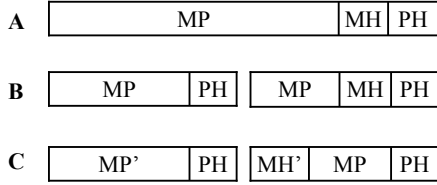
| A | MP | | MH | PH |

| B | MP | PH | MP | MH | PH |

| C | MP' | PH | MH' | MP | PH |

Figure 6. Message-packet alignment

(PH) can be easily identified. In this case we have also identified the message header (MH), which follows the packet header, and the message decoding can start. From the NoC point of view this is a simple packetization scheme, but it may require long packets. Therefore, it may not fit well with the TDMA scheme, and may have partially empty packets.

(B) *The message header is aligned with the packet header but the NoC splits messages in multiple packets, see Fig. 6B.* Even if a packet header can be easily identified, as message/packet correspondence is no longer one to one, it is important to know with which packet header the message header is aligned. Some packet headers are followed by message headers, others are followed by parts of the message payload (MP). This situation fits well with the TDMA scheme, but may have partially empty packets.

(C) *The NoC does not align messages and packets, see Fig. 6C.* This is the most general and difficult case and is used by the Æthereal NoC. A message header can be anywhere in the payload of a packet. This is the most efficient packing of messages in packets and it is good for TDMA.

There are at least two solutions to solve the message identification problem described at points B and C:

(1) *One solution is to explicitly specify the message boundaries.* This means that each packet specifies whether it contains the start of the message and where in the packet is the message header. The presence of a message header in the packet and its offset can be coded either in the packet header itself or in the sideband information in the form of a control bit. This can be enforced by adapting the master NI port (MNIP) to include this information in the header or sideband information, which is not difficult because the NI has knowledge of the message start. Using this solution may require design modifications of the NI.

(2) *A second solution is to make sure that we can monitor the first packet going over the connection, and all the*

following flits belonging to the same connection. In this case we can continue identifying messages. This is because the first packet will contain a packet header immediately followed by a message header and from there we can keep accounting for the following messages like the SNIP is already doing for the user traffic. This can be enforced e.g. by setting and enabling the transaction monitors before the connection is used. The advantage of this solution is that it requires no modifications of the NoC components.

For our experiments with the Æthereal NoC, we have enforced the second solution **(2)**. We have made this choice because the Æthereal packet header does not contain information about message headers. The potential drawback of this solution, the fact that the transaction monitors must be enabled before the actual monitored connection is set up, is considered acceptable. Our solution requires a strict correlation between the (re)configuration moments of the NoC and the configuration of the NoCMS transaction monitors. It is possible to apply this solution for the Æthereal NoC, because our transaction monitors are run-time configurable by means of MMIO write operations, and because precedence of the transaction monitor configuration in front of the user connection configuration is enforced. The transaction monitors are configured before the actual connections are configured, being able to sniff all the data from a connection starting with the first flit of the first packet. In case of a NoC reconfiguration, it is again possible to reconfigure all the transaction monitors at the beginning of the reconfiguration when the rest of the connections are not yet configured.

The traffic introduced by the analyzer transaction-based mode is lower than in the connection-based mode as packet headers are removed in transaction reconstruction, when converting from packets to messages. This is done in the depacketization block of the transaction monitor, in fact a reused SNIP from the Æthereal NoC.

By identifying messages, local filtering of messages per connection is possible; all these options can be added to the depacketization block. For example, filtering of only write or read messages, or filtering of certain address range writes can be done, handy for debug purposes.

The transaction-based mode is useful when all details of transactions or transaction components are required to be inspected. This is especially useful when inspecting IP to IP communication. However, details regarding packetization, like the content of packet headers, are no longer visible.

4.4   Transaction Event-based Mode

In the transaction event-based mode the analyzer provides full observability on relevant transaction features or

6

components and abstracts other irrelevant transaction features. Being able to identify messages over a certain connection is indeed very useful. However, not all of this information is always needed for getting the picture of what is really going on in the NoC. Therefore, the abstraction level can be raised; whenever a transaction component is sniffed, a transaction event can be generated. E.g. a transaction event can state what was the command, address and the number of words in a message. A write message at address #0000 with 10 words of payload has a total of 12 words for the entire message. All this can be abstracted in an event of two words containing only the relevant features (command, address, nr. words), getting rid of the irrelevant (for this example) 10 words of payload.

Local filtering of relevant features of transactions is done in the abstraction block of the transaction monitor. The traffic introduced is lower than in the transaction-based mode as we get rid of packet headers and irrelevant transaction features by means of event abstractions.

Table 1. Comparison of analyzer modes

Mode	TM capability	Filtering	Potential pb.
raw	id. traffic	GT/BE	link bw.
connection	+id. connection	connections	link bw.
transaction	+id. msg.	messages	msg. start
tr. event	+evt. generation	msg. features	msg. start

Table 1 summarizes this section, showing a comparison between all analyzer modes focusing on the capabilities built in the transaction monitor, the potential filtering and the potential problems in each of the modes.

5 Analysis and Examples

5.1 Implementation

Our final point is to prove that on chip run-time monitoring of NoC transactions is feasible in resource constrained NoC designs. Therefore, we have investigated the area and traffic implications of our NoC analyzer. For the experimental validation of our NoC analyzer we have built a flit accurate SystemC model, and a cycle accurate synthesizable VHDL model of the transaction monitor. We have used these models in conjunction with the Æthereal NoC and design flow.

The placement of the transaction monitors at routers is a design time choice. Currently, our NoC design flow [10] has been extended to support monitoring in general [6], and fully supports the (automatic) insertion of transaction monitors in particular [7].

For our experiments all routers are instrumented using the monitoring-aware NoC design flow with transaction monitors, thus resulting in a fully probed NoC. For our traffic experiments with the SystemC models we have used transaction monitors supporting all four analyzer modes. For our area experiments with the VHDL models we have used transaction monitors supporting the first three analyzer modes (full transaction reconstruction without transaction abstraction). We use GT connections for each of the transaction monitors to transport the sniffed data from the transaction monitors to the MSA. The application and monitoring traffic share the same NoC. For NoC design-time aspects related to monitoring, such as how to provision the monitoring requirements or how to automate the insertion of monitors please refer to [6] and [7] respectively.

To quantify the complete effects of monitoring we had a look at four different SoC designs, using NoC mesh topologies, supporting combinations of several MPEG instances (from one to four) with a single audio instance consisting of sample rate conversion, MP3, audio-postprocessing and radio as presented in [17].

5.2 Area Analysis

The area overview of three probes, corresponding to the first three analyzer modes (raw, connection-based and transaction-based), realized in a 0.13μm CMOS technology is presented in Table 2. Since transaction monitors support both GT and BE traffic classes, a comparison is made with the area of an Æthereal arity 6, GT/BE router, presented in [11], which is 0.13mm^2 in the same 0.13μm CMOS technology.

Table 2. Area Impact

Probes	area(mm^2)	comp. to router
raw	0.020	15%
connection-based	0.024	18%
transaction-based	0.026	20%

The results show that offering raw data monitoring capability would require 15% more area compared to a single router, while connection-based capabilities would require around 18%. Full fledged on-chip transaction reconstruction is feasible at the cost of 20% of the router area. The probe area presented includes the configuration unit allowing to (re)configure the transaction monitors at run-time. Configuration includes the start and end time of transaction monitor activity, the selection of the desired mode with the required characteristics. It also includes three words of internal storage.

The first three analyzer modes realize the transaction reconstruction, by decoding the transaction components, while the fourth one is doing the transaction abstraction. Therefore, the area cost of the probe supporting the first

three modes is fixed and the area cost of a probe supporting all four analyzer modes may vary, depending on the transaction abstraction capability implemented. The area cost of transaction abstraction is left for future work.

Looking at the NoC level, on average, for our designs, the overall monitoring system adds 5% to the original NoC area (routers and NIs). This accounts for the transaction monitors area, and the increase in the number of NI ports with the corresponding buffers.

5.3 Traffic analysis

A traffic comparison for all analyzer modes is presented in Table 3. The monitoring targets are two user connections of 40 MB/s and 60 MB/s respectively. The monitoring was done by activating a single transaction monitor for each of the target connections, and configuring it in turns for each mode. The traffic figures correspond to a single transaction monitor and are independent of the number of transaction monitors present in the NoC.

Table 3. Monitoring traffic details

user data		MB/s	MB/s
user payload	P	40	60
NoC payload	P+C+A	59,84	89,68
NoC traffic	P+C+A+H	65,20	95,04
raw			
debug payload	P'= $\sum$(P+C+A+H)	255,28	
(mm) NoC payload	P'+C'+A'	340,37	
(mm) NoC traffic	P'+C'+A'+H'	355,76	
(sd) NoC payload	P'	255,28	
(sd) NoC traffic	P'+H'	270,88	
connection-based			
debug payload	P'=P+C+A+H	65,20	95,04
(mm) NoC payload	P'+C'+A'	86,93	126,72
(mm) NoC traffic	P'+C'+A'+H'	92,08	131,84
(sd) NoC payload	P'	65,20	95,04
(sd) NoC traffic	P'+H'	70,40	100,24
transaction-based			
debug payload	P'=P+C+A	59,84	89,68
(mm) NoC payload	P'+C'+A'	79,79	119,57
(mm) NoC traffic	P'+C'+A'+H'	84,96	124,72
(sd) NoC payload	P'	59,84	89,68
(sd) NoC traffic	P'+H'	65,04	94,88
tr. event-based			
debug payload	P'=E	19,95	29,89
(mm) NoC payload	P'+C'+A'	39,89	59,79
(mm) NoC traffic	P'+C'+A'+H'	45,04	64,96
(sd) NoC payload	P'	19,95	29,89
(sd) NoC traffic	P'+H'	25,12	35,04

The user payload, e.g. 40 MB/s, denoted with P in Table 3, represents the application data. To this user data,
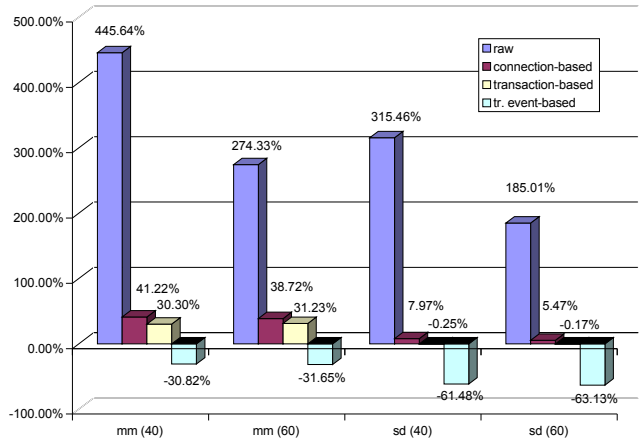

Figure 7. Monitoring traffic comparison

commands (C) and addresses (A) still have to be added before the NIs, the value is shown as P+C+A in Table 3. This represents the actual payload for the NoC. Taking into account the slot allocation, due to packetization, headers are added in the NIs to the actual payload, P+C+A+H in the table. P+C+A+H is the traffic going through the network.

When monitoring in the *raw mode* the sum of P+C+A+H for all connections passing the monitored link becomes the actual payload P' for the debug connection. In our experiment the two monitored connections share the same link, together with another 60MB/s GT user connection, which explains the large payload in Table 3. Note that the raw traffic is only directly related with the the monitored link utilization.

When monitoring in the *connection-based mode* P+C+A+H becomes the actual payload P' for the debug connection. When monitoring in the *transaction-based mode* P+C+A becomes the actual payload P' for the debug connection, because the headers are removed in this analyzer mode, assuming no further message filtering which is the worst case for this mode. When monitoring in the *transaction event-based mode* the actual payload E is computed in this example by abstracting the 6-word messages (P=4 words, C=1 word and A=1 word) used on this connection in 2-word events. In general it may vary depending on the abstraction capability of the event-model.

When using the *memory-mapped* scenario, the (mm) tag in Table 3, for the transport of the monitored data, new addresses and commands are added to the previously explained payload P', denoted P'+C'+A' in Table 3. New packetization is done, and new headers (H') are added to this, getting the final debug connection traffic to P'+C'+A'+H'. When using the *streaming-data* scenario, (sd) in Table 3, for the transport of the monitored data, new

addresses and commands do not need to be added to P'. Headers are added though, getting the final debug connection traffic P'+H'.

Fig. 7 presents the NoC analyzer traffic for all analyzer modes in percentages, compared to the initial NoC traffic of 65.20 MB/s and 95.04 MB/s for the 40MB/s and 60 MB/s connections respectively. This is done for both the *memory-mapped* (mm) and *streaming-data* (sd) scenarios. In the raw analyzer mode the numbers show that there is a tremendous traffic on a link, several times bigger than the considered user connections. Note that the raw mode shows the overall link traffic comprising three connections in total. In the connection-based mode the numbers show that we introduce in the NoC new traffic, bigger than the monitored traffic. In the (mm) scenario this is 41% and 39% more, while in the (sd) scenario it is only 8% and 5% more. In the transaction-based mode we introduce in the NoC new traffic around 30% higher than the monitored traffic in the (mm) scenario, and comparable but slightly lower than the monitored traffic in the (sd) scenario. In the transaction event-based mode we introduce in the NoC new traffic, lower than the monitored traffic, which is always the case for sufficiently abstract events. This represents a real gain over the monitored connections. The gain amounts to around 30% and 60% in the (mm), respectively (sd) scenario, in the concrete example.

As expected, traffic wise it is a good idea to use transaction abstractions when doing online transaction monitoring, in case other details are not of interest. Combining them with a *'streaming-data'* scenario is beneficial. Monitoring at the lowest level of detail would produce the most load for the NoC.

5.4 Debugging the NoC Configuration Master

As previously mentioned, the Æthereal NoC can be configured at run-time. In the current Æthereal setup, a centralized programming module, e.g. an ARM processor, is doing all the configuration work. This centralized module is called the *Configuration Master*, see the Cfg. Master in Fig. 4. As support for NoC debug, it is important to see what the Configuration Master is doing at run-time, as all the inter-IP communication is going over connections. The observed behavior can then be compared to the expected behavior in order to catch possible errors.

The Configuration Master uses BE packets to configure the NIs. Our NoC analyzer can monitor configuration in the connection-based mode. For this we take advantage of the Æthereal configuration details. All NIs have a port, with qid 0 , called a configuration port. Through this port the NIs are configured at run-time. The observation of the Configuration Master requires a single probe. Therefore at run-

time, a single transaction monitor was activated in one test NoC, depicted in Fig. 4, namely the transaction monitor attached to router R3. This transaction monitor monitors the link between NI0, where the Configuration Master is connected, and router R3. The transaction monitor was enabled in the connection-based mode, and was configured only with the queue identifier 0 and not also with the path. In this way all the outgoing traffic from NI0 towards any destination with qid=0 is filtered, and then sent to MSA. This amounted to 205 flits containing 529 (4-byte) words of configuration data. In this way all the Configuration Master behavior is observed. A single preestablished GT connection is used to transport the monitored data to the MSA. As another experiment the transaction monitor was enabled in the transaction-based mode and the path was set. Transactions are only monitored over this path, corresponding to NI4 being configured, and this amounts to 24 write transactions.

Due to the centralized programming of the NoC using a single Configuration Master in the current setup, one transaction monitor is currently sufficient to monitor NoC configuration. However, in the near future, distributed programming of the NoC may be another option. In order to keep up with this option multiple Configuration Master monitors will have to be employed (or activated), one for each Configuration Master.

6 Conclusions

We have presented a NoC analyzer able to perform run-time NoC transaction monitoring. The proposed NoC analyzer alleviates the run-time observability problem by providing hardware transaction monitors able to work on four different levels of abstraction. They correspond to four analyzer modes, ultimately being able to on-chip reconstruct transactions from low-level monitored router data and abstract them to events. All of the analyzer modes can be enabled and configured at run-time. They match difficult debug situations, and are a valuable asset when debugging multiprocessor NoC-based SoCs.

In NoC monitoring it is important to go beyond the raw low-level data (bits), to understand what data means (transactions). Due to nonalignment of packets and messages it is generally difficult to (re)construct a transaction level view. We have conceptually shown how this problem can be solved for all existing NI packetization schemes. Thus our concepts can be reused for any existing NoC. A transaction monitor for the most difficult packetization scheme was implemented at the cost of one fifth of the router area. The total monitoring system leads to an increase of NoC area of around 5% for several MPEG/audio SoCs.

A traffic analysis of analyzer modes has been presented. The traffic introduced, compared to the traffic of the mon-

itored connection, varies from a penalty of 41% in the connection-based mode memory-mapped scenario, to a gain of 63% in the transaction event-based mode streaming-data scenario. We have proven the versatility of our NoC analyzer by debugging the NoC configuration master.

Acknowledgement

References

[1] ARM. *Coresight*. http://www.arm.com/products/solutions/CoreSight.html.

[2] ARM. *Embedded Trace Macrocell*. www.arm.com, 2002.

[3] L. Benini and G. De Micheli. Networks on chips: A new SoC paradigm. *IEEE Computer*, 35(1):70–80, 2002.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Journal of Systems Architecture*, 50(2–3):105–128, Feb. 2004. Special issue on Networks on Chip.

[5] C. Ciordas, T. Basten, A. Rădulescu, K. Goossens, and J. van Meerbergen. An event-based monitoring service for networks on chip. *ACM Transactions on Design Automation of Electronic Systems*, 10(4):702–723, Oct. 2005. HLDVT'04 Special Issue on Validation of Large Systems.

[6] C. Ciordas, K. Goossens, A. Rădulescu, and T. Basten. NoC monitoring: Impact on the design flow. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*, pages 1981–1984, May 2006, IEEE 2006.

[7] C. Ciordas, A. Hansson, K. Goossens, and T. Basten. A monitoring-aware NoC design flow. In *Proc. Euromicro Symposium on Digital System Design*, Aug. 2006, IEEE 2006.

[8] W. J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. Design Automation Conference (DAC)*, pages 684–689, 2001, ACM 2001.

[9] S. González Pestana, E. Rijpkema, A. Rădulescu, K. Goossens, and O. P. Gangwal. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 764–769, Feb. 2004, IEEE 2004.

[10] K. Goossens, J. Dielissen, O. P. Gangwal, S. González Pestana, A. Rădulescu, and E. Rijpkema. A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1182–1187, Mar. 2005, IEEE 2005.

[11] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):21–31, Sept-Oct 2005.

[12] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 250–256, 2000, IEEE 2000.

[13] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, Sept. 2002.

[14] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. In *Proc. Symposium on VLSI*, pages 117-124, 2002, IEEE 2002.

[15] R. Leatherman and N. Stollon. An embedded debugging architecture for socs. *IEEE Potentials*, 24(1):12–16, Feb-March 2005.

[16] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proc. Int'l Conference on VLSI Design*, pages 693–696, 2004, IEEE 2004.

[17] A. Moonen, R. van den Berg, M. Bekooij, H. Bhullar, and J. van Meerbergen. A multi-core architecture for in-car digital entertainment. In *Proc. of GSPx Conference*, 2005.

[18] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 896–901, 2004, IEEE 2004.

[19] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 234–239, Mar. 2005, IEEE 2005.

[20] V. Nollet, T. Marescaux, and D. Verkest. Operating-system controlled network on chip. In *Proc. Design Automation Conference (DAC)*, pages 256–259, June 2005, ACM 2005.

[21] First Silicon. *BusNavigator*. http://www.fs2.com/busnavigator.html.

[22] B. Vermeulen, J. Dielissen, K. Goossens, and C. Ciordas. Bringing communication networks on chip: Test and verification implications. *IEEE Communications Magazine*, 41(9):74–81, Sept. 2003.