A Configurable Monitoring Infrastructure for NoC-Based Architectures
Leandro Fiorin, Gianluca Palermo, and Cristina Silvano

Abstract—In this brief, we propose a monitoring architecture for networks-on-chip that provides system information useful for designers to efficiently exploit, at design time and run-time, the system resources available in multiprocessor system-on-chip platforms. We focus on the analysis of the architectural details and design challenges of such a system, by describing powerful tools for monitoring information that can be used both at run-time for detecting dynamic changes in system behavior and at post-execution time for debugging and profiling of applications. This brief describes the design of the monitoring probes, together with the events detectable by them, and discusses an architecture for collecting, storing, and analyzing the information gathered during an application execution.

Index Terms—Hardware counters, networks-on-chip (NoCs), performance monitoring, systems-on-chip (SoCs).

I. INTRODUCTION

In multiprocessor architectures nowadays, the monitoring of system resources is needed for performance optimization, debugging, as well as for run-time tuning of resources utilization. First, using a set of selected hardware event detectors and counters, the information concerning system components can be collected during program execution to be analyzed and then exploited for the optimization of user applications [1], [2]. Second, in modern systems-on-chip (SoCs), the system observability becomes a significant bottleneck, making on-chip instrumentation modules essential for system debugging [3]. Third, highly changing run-time requirements call for platform adaptivity and run-time optimization of resources utilization [4], and a monitoring system offers the possibility to react and rapidly adapt to changing application behaviors, requirements, and/or customer needs.

In complex communication-centric platforms, such as network-on-chip (NoC)-based architectures, a monitoring system should be similarly deployed within the on-chip communication system, supporting scalability, nonintrusiveness, and run-time configurability. NoC monitoring has been already recognized as fundamental for debugging and testing SoC architectures [5], detecting congestion in best effort networks [6], platform run-time management/adaptivity [4], and security purposes [7].

In this brief, we address the problem of NoC monitoring by proposing a configurable architecture that enables a better flexibility on the choice of the parameters that can be observed at run-time through the communication subsystem, and provides system designers and application developers with a comprehensive set of tools and information that can be exploited at the NoC level for: 1) tuning system performance during the postmanufacturing optimization phase; 2) supporting online debugging; and 3) enabling run-time management of system resources. We consider in particular passive monitoring, and we focus on events that can be observed by the NoCs network interfaces [8]. The proposed monitoring infrastructure is complementary to the use of conventional simulators and emulators during the early design-time optimizations: while those design tools are fundamental for predicting system behavior and obtaining the best performance/power tradeoff at design time, a monitoring infrastructure is needed for observing the run-time behavior of the system and triggering its adaptation to changing requirements.

II. MONITORING ARCHITECTURE

In this brief, we target a shared-memory multiprocessor architecture based on a tiled-based NoC, where processing/ storage element (PE), network interface (NI), and router (R) are elements of the same tile. We consider a transaction-based protocol and memory mapped elements over the NoC. IP modules acting as initiators perform load (read) and store (write) transactions towards IP modules acting as targets. For each transaction, an acknowledgment ack (or negative acknowledgment, nack) message is sent back to the initiator by the target. The NI, compliant with the Open Core Protocol (OCP) [15] specifications, is composed of an OCP adapter, which serves as an interface between the OCP transactions of the core and the NI, an NI kernel, providing the basic NI services (lookup of destination target addresses, packing/unpacking of data), and a scheduler, to send packets on the NI output queue. The NoC implements a packet transmission based on the wormhole switching strategy.

The four main components of the proposed monitoring system are as follows: probes (P), located inside each NI and, by snooping OCP signals at the interface of the core, observing cores’ operations and events as well as resources utilization and the generation of communication events in the router and the NI; a data collection and a data storage subsystem that manages the collection and storage of the information generated by probes; a centralized element [probes management unit (PMU)], in charge of the configuration of the probes, the retrieval of the collected data, and its elaboration both after the execution of applications and at run-time, depending on the functionalities offered by the platform.

A. Programmable Probes

We propose the implementation of a multipurpose probe that can be programmed for detecting several types of events. Fig. 1 shows...
the general architecture of the probe. It is composed of six main elements.

- An events detector, which reacts every time the event to be monitored occurs.
- An accumulator, used for collecting measured values for the observed event.
- A collection of preprocessing modules for early data elaboration.
- A message generator, which creates messages to be sent to the PMU and the storage elements, and inserts them into the probe output queue.
- A probe output queue for buffering the message generated by the message generator before being scheduled for transmission.
- The configuration register, used to configure the probe for monitoring different events.

1) Events Detector: The events detector observes OCP signals as well as NI and router internal signals by working at the core interface. It operates in parallel with the NI kernel without introducing additional latency and can monitor the following types of events: throughput events, timing/latency events, resources’ utilization events, and message characteristics and statistics about messages and packets generation. The information characterizing the detected event is stored in the accumulator.

The throughput detector keeps track of the amount of traffic to/from a selected range of addresses generated by an initiator (i.e., a thread running on a PE). When the elements identifying the transaction (i.e., initiator, target, and type of operation) match those specified in the configuration register, the information about the amount of data transferred in the transaction are stored into the accumulator. Throughput can be collected for the whole execution, for specific time windows, and for active connections at the initiator.

The timing/latency detector is in charge of measuring timing properties of transactions: the time between an initiator’s request and the reception of the acknowledge message (initiator-to-initiator), the time needed for a request (initiator-to-target), the time needed by the target for executing its task (execution time), and the time needed by the acknowledgment message of the target to reach the initiator and complete the transaction (target-to-initiator). This event detector involves the collaborative action of probes both at the initiator and the target of the transaction. Latency can be collected for every single transaction and for different connections.

The resources utilization detector monitors the status and the occupation of the internal queues of NIs and routers by observing changes in the signals cnt_queue, directly taken from the counter used in general implementations of a hardware queue for measuring the number of slots occupied. From the configuration register, it is possible to specify which queue (among the NIs and router’s ones) should be monitored, as well as the sampling time, in number of clock cycles.

The messages characteristics detector aims at detecting user and NoC configuration events [5]. User configuration events include the detection of information about the characteristics of the communication between two cores, such as for instance, the connection identifier, the type of connection, and so on. NoC configuration events are mainly related to communicate modifications in the configuration of the elements of the NoC (such as changes in routing table entries of routers), and of the system in general.

2) Data Preprocessing: The detection of some of the events can generate too much traffic to be supported or managed. We implemented the possibility to reduce the traffic generated by the probes by processing collected data before sending them to the PMU. Preprocessing is enabled by selecting the desired functionality in the probe configuration register. In particular, we implemented the possibility to generate messages at the end of time windows and when data collected are higher, equal, or lower than a specified threshold. Moreover, it is possible to send an average calculation of the sampled values collected during the entire application scenario or for each time window.

3) Message Generator: At the end of a time frame, when a specific event occurs, or when reaching the end of the application, the message generator creates messages to communicate to the PMU the detected event or set of events, by generating a packet whose payload contains the information kept in the accumulator and by inserting it into the probe output queue. It acts as an initiator performing a write operation that sends a packet to the memory location associated with the probe and specified into the probe configuration register. Transmitted information can be aggregated by enabling the dispatch of multiple measurements in the same packet and reducing in this way the amount of traffic generated by the probe.

4) Probe Configuration: The functionalities of the probe can be activated by enabling the corresponding field in the configuration register [8]. Through the configuration register, it is possible to select the type of event to be monitored by the probe, the identifiers of the source and the destination of the connection, the type of operation performed (read, write, or all), as well as specifying which connections to monitor between initiators on the source core and targets on the destination core (one-to-one, one-to-all, all-to-one, and all-to-all). Moreover, it is possible to enable the execution of the preprocessing operations and specify the length of the time window, as well as the value of the threshold to be used as comparison with the information collected by the accumulator of the event detector, and the type of comparison to be performed on that information. The aggregation factor can be specified as well, to reduce the amount of traffic generated by the probe. Additional fields of the register are used for selecting the internal queue to be monitored by the probe and configuring the sampling time (in clock cycles). The configuration register specifies also the memory address storing the value of the event detected by the probe. An additional bit in the configuration register is used for specifying whether notifying the PMU on the arrival of the event message from the probe, in particular, in the case of the use of the monitoring system for the run-time management of the platform.

When this bit is enabled, messages transmitted by the probes generate an interrupt at their arrival at the NI of the PMU, allowing the central unit to immediately receive the information. Configuration registers are memory mapped and they are configured by the PMU before program execution or at run-time. To be able to interpret correctly and process the received data, the PMU keeps a record of the configuration of each probe.

### Table I

**Cost of Probes and Tile Components**

<table>
<thead>
<tr>
<th></th>
<th>Area (μm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single multipurpose</td>
<td>0.052</td>
</tr>
<tr>
<td>probe (init.)</td>
<td></td>
</tr>
<tr>
<td>timing/latency probe</td>
<td>0.037 / 0.017</td>
</tr>
<tr>
<td>(init. / target)</td>
<td></td>
</tr>
<tr>
<td>4 multipurpose probe</td>
<td>0.155</td>
</tr>
<tr>
<td>(init. / target)</td>
<td></td>
</tr>
<tr>
<td>Router (5 p)</td>
<td>0.143</td>
</tr>
<tr>
<td>NI initiator / target</td>
<td>0.141 / 0.172</td>
</tr>
<tr>
<td>ARM920T</td>
<td>4.7</td>
</tr>
</tbody>
</table>

### B. Implementation Cost

We implemented the monitoring probes considering a 0.13-μm technology library and optimizing the synthesis for a clock frequency of 500 MHz, i.e., the frequency imposed when synthesizing the routers and the network interfaces of the NoC. Due to the relatively simple implementation, the maximum frequency of the components of the monitoring system is, however, higher than the one of the
implemented for calculating the latency of the transactions. As a
detector, due to the monitoring options available and the protocol
timing/latency probe is for this reason the most power hungry event
detector is proportional to its architectural complexity and the
other components of the probe. The energy consumption of each
component with the same technology).
consider as reference an ARM920T with 16 kB of data/instruction
processing element in the initiator tile is approximately 3% (we
the overhead obtained when considering in the evaluation also the
for debugging purposes [5], but we added flexibility to the system.
comparison, we also reported in Table III the energy dissipation
for the network components derived by [10]. Summarizing, the
synthesis results demonstrate that both energy and area overheads
are acceptable given the service provided at network level by the
proposed infrastructure.

C. Data Collection and Storage
We consider an NoC in which communication resources are shared
among nominal data traffic and monitoring traffic. This solution
presents a limited overhead on the NoC implementation, given that
the network is able to guarantee enough bandwidth to support both
data flows without significant interference between them [5].

1) Collection: Table IV presents the bandwidth expected to be
generated by a selected set of events, in terms of total execution
time of the application scenario (exec_time), the length of the
observed time window (time_window), the sampling time of the
queue utilization (queue_sampling_time), the number of transactions
executed during the entire observation period (#transactions), and the
aggregation factor (α). Similar formulas can also be found for other
events not shown in this table. The minimum traffic generated by
the probe is equivalent to 64 bits, i.e., a two-flit packet composed
of a header and a payload flit (we assume a 32-bit data width for
the NoC physical links). Such traffic can be observed for single
event messages, such as those generated in the case of the detection
of the measurement of the throughput during the execution of the
application scenario. A 96-bit packet (three flits) is employed for
sending information about averaged measurements. The first data
flit contains the measured value, whereas the second contains the
counted number of occurrences of the event. For some measurements,
such as the one counting the number of times the initiator-to-initiator
latency (I2I latency, in Table IV) is above a certain threshold, the
bandwidth depends on the number of occurrences of the event.
Table IV reports therefore the maximum value, i.e., the bandwidth
generated when all the transactions are above threshold. In the case
of messages depending on the number of events appearing on the
NoC, an estimation of the needed bandwidth should be performed
before data collection by preceding the measurement of the event
to be estimated with another execution of the application scenario,
to measure the number of occurrences of the event. The overhead
in terms of performance and power on the NoC due to the data collection
is directly proportional to the bandwidth generated by the probes.

2) Storage: We can distinguish between two storage strategies:
the use of the local memory in the PMU and the use of an on-chip
streaming memory for larger amount of data. Those two different
approaches can support the run-time management and the profiling
features, respectively.

The PMU local memory can be employed for events generating
a limited number of values for each execution (e.g., the throughput
measured in one communication). Local storage is moreover
important in the case of utilization of collected data for the analysis
of run-time system behavior for applications that may require a run-
time adaptive reaction from the system. Values stored in the PMUs
local memory or registers can be accessed by the operating system

### Table II

<table>
<thead>
<tr>
<th>Event</th>
<th>Energy (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput probe</td>
<td>0.29</td>
</tr>
<tr>
<td>Timing/latency probe</td>
<td>2.10</td>
</tr>
<tr>
<td>Resource utilization probe</td>
<td>0.15</td>
</tr>
<tr>
<td>Message characteristics</td>
<td>0.87</td>
</tr>
<tr>
<td>Accumulator</td>
<td>11.51</td>
</tr>
<tr>
<td>Output queue</td>
<td>12.90</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>Event</th>
<th>Energy (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput probe</td>
<td>0.29</td>
</tr>
<tr>
<td>Timing/latency probe</td>
<td>2.10</td>
</tr>
<tr>
<td>Resource utilization probe</td>
<td>0.15</td>
</tr>
<tr>
<td>Message characteristics</td>
<td>0.87</td>
</tr>
<tr>
<td>Accumulator</td>
<td>11.51</td>
</tr>
<tr>
<td>Output queue</td>
<td>12.90</td>
</tr>
</tbody>
</table>

### Table IV

<table>
<thead>
<tr>
<th>Event</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput for execution</td>
<td>64b/exec_time</td>
</tr>
<tr>
<td>Throughput for time window</td>
<td>64b/time_window</td>
</tr>
<tr>
<td>Throughput for transaction</td>
<td>64b/#transactions/exec_time</td>
</tr>
<tr>
<td>I2I latency above threshold</td>
<td>max(64b * #transactions/exec_time)</td>
</tr>
<tr>
<td>queue utilization</td>
<td>16byte/queue_sampling_time</td>
</tr>
<tr>
<td>Avg. I2I latency in time window</td>
<td>96b/time_window</td>
</tr>
</tbody>
</table>

NoC. Table I shows the area, in mm², of a single multipurpose
probe, as well as the cost of a monitoring system composed of four
probes, able therefore to monitor four different events in parallel in
the tile. The implementation of a multipurpose probe implies an area
overhead of around 13% with respect to a probe implementing the
most expensive single monitor, i.e., the timing/latency probe (at the
initiator). However, this overhead can be considered acceptable given
the increased flexibility of the probe and the possibility to configure
it for monitoring different events.

An alternative choice to the use of multipurpose probes could
have been to implement all the event detectors separately, such
as in [2]. However, the cost of a complete implementation would
have been prohibitive, in particular, because of the high number
of possible connections to be monitored, and for the significant
amount of buffering space needed for accumulating locally at the
probe all the possible events for guaranteeing nonintrusiveness in
the message generation. The area saving obtained when comparing
a monitoring system with four multipurpose probes with such a full
monitoring system (able to monitor all the events in parallel) is around
52%, being its estimated area of 0.328 mm². In the comparison,
we considered the full monitoring system having probes for every type
detectable event, and able to follow four different connections
simultaneously [2], [9]. When considering as reference architecture
a shared memory multiprocessor composed of 10 initiators and two
target memories, mapped on a mesh tile-based NoC, the area of the
four-probe system at the initiator is approximately equal to the 55%
of the total area of the NI and the router, while at the target is
equal to the 5%. Overall, the monitoring system with four probes
occupies the 31% of the total area budget of the NoC. NIs and
routers were obtained using the PIRATE-NoC compiler [10], and
buffers length was imposed equal to 4 for the routers, while equal
to 8 and 16, respectively, for NIs at initiators and targets. Obtained
overhead is of the same order of magnitude of probes implemented
for debugging purposes [5], but we added flexibility to the system.
The overhead obtained when considering in the evaluation also the
processing element in the initiator tile is approximately 3% (we
consider as reference an ARM920T with 16 kB of data/instruction
caches implemented with the same technology).

Table II shows energy values per operation associated with the
data detection of each different event of the probe, as well as to
the other components of the probe. The energy consumption of each
event detector is proportional to its architectural complexity and the
amount of storage elements needed for performing its operation; the
timing/latency probe is for this reason the most power hungry event
detector, due to the monitoring options available and the protocol
implemented for calculating the latency of the transactions. As a
in a faster way, and they can be employed for evaluating appropriate reactions to changes in the system detected by the probes.

The streaming memory approach is employed for storing data exceeding the allocated space in the PMU’s local memory, and for data whose dimension is not known before the execution of the application to be monitored. The streaming memory, implemented using an approach similar to [11], is memory mapped to the probes through a field in their configuration registers. Once the application is completed, the PMU is able to retrieve the data stored into the memory.

D. PMU

The PMU can be in charge of managing the profiling or the run-time adaptation of the resources. In the first case, tasks performed by the PMU are mainly twofold: programming the configuration registers, and retrieving the collected data for processing them. These tasks can be implemented with two software routines running on a generic processor. No particular performance requirements are imposed on these two routines, as they are executed when the application is not running. The former task is executed before running the application to be monitored, and it is in charge of writing the memory-mapped configuration register of each probe for detecting the desired events. A record is kept by the PMU that associates the event detection and transmission is equal to the specified time window (the control interval in which the application is divided. This amount of time is conservatively large enough for completing the operations of event detection and communication to the PMU, execution of the simple control algorithm, and updating the routers with the new operating points [13]. We monitor the input queue of router R0 connected to the MC → ADD PE (QR0MC → ADD), being the PE requiring a higher bandwidth. Moreover, we monitor the input queue of router R1 connected to router R0 (QR1R0), which gives an indication about the capability of R1 of satisfying bandwidth requirements of messages coming from R0. By profiling the queues behavior in the four cases, we obtained a simple rule for adapting the frequency of the two routers. The R0 (R1) frequency will be increased when the utilization rate is lower than 0.05. When the two alerts are received in the same control interval, priority is given to the adaptation of R1.

Fig. 3 shows the results of the execution, showing 1.5-ms time windows in which there is a switch from one case to another. We run every application mode for 5 s with the sequence $M_{VH} \rightarrow M_{VL} \rightarrow M_{VH} \rightarrow M_{VM}$, for a total simulation length of 20 s. Fig. 3 shows the measured values for queuing utilization and routers frequency when varying video mode ($\beta$). At 5 and 10 s, the increment of the network load ($M_{V0} \rightarrow M_{VL}$ and $M_{VL} \rightarrow M_{VH}$ transitions) can be noticed by the high queue utilization that requires a subsequent increment of the router(s) frequency, that in the second case occurs through five intermediate steps (0.5 ms). The opposite occurs at 15 s ($M_{VH} \rightarrow M_{VM}$ transition) where the under-utilization of both queues required to reduce both frequencies. No other event occurs during the rest of the simulation. The average power consumption associated with the event detection and transmission is equal to 0.966 mW, while the traffic generated by the probes is equal to 72 Bytes. We evaluated the power consumption of our implementation (including the probe overhead that is less than 1 mW) with respect to a baseline architecture in which the routers have been designed.
for the worst case scenario ($M_{VH}$) running R0 at 1 GHz and R1 at 250 MHz, obtaining an average saving of about 19%, while the average saving increases up to 40% when comparing with the case in which both routers run at 1 GHz.

IV. CONCLUSION

In this brief, based on a comprehensive study of the most common events detectable through the communication subsystem, we proposed the utilization of a configurable multipurpose probe to face the monitoring problem in NoC-based systems. Moreover, we evaluated the intrusiveness of the proposed monitoring system, the area/energy/traffic overheads, as well as a discussing its use for run-time management.

REFERENCES