A Monitoring-Aware Network-on-Chip Design Flow

Calin Ciordas † Andreas Hansson † Kees Goossens ‡ Twan Basten †

† Eindhoven University of Technology
{c.ciordas,m.a.hansson,a.a.basten}@tue.nl
‡ Philips Research Laboratories Eindhoven
kees.goossens@philips.com

Abstract

Networks-on-chip (NoC) are a scalable interconnect solution for systems on chip and are rapidly becoming reality. Monitoring is a key enabler for debugging or performance analysis and quality-of-service techniques. The NoC design problem and the NoC monitoring problem cannot be treated in isolation. We propose a monitoring-aware NoC design flow able to take into account the monitoring requirements in general. We illustrate our flow with a debug driven monitoring case study of transaction monitoring. By treating the NoC design and monitoring problems in synergy, the area cost of monitoring can be limited to 3-20% in general.

1 Introduction

Advances in semiconductor technology have enabled very complex large scale systems on a chip (SoCs) designs. Each new SoC generation integrates more processing elements (IPs) and offers increased functionality. As the number of IPs increases, traditional interconnects, such as busses, become a bottleneck.

Networks-on-chip (NoCs) are a modular, scalable interconnect solution [1,2,6,9,10,13,15]. Currently, they tend to become the preferred interconnect solution for large scale inherently multiprocessor SoCs. However, NoCs require sophisticated tools to aid in design-time decisions [3, 8, 11, 18]. Furthermore, with increasing complexity there is also a strong need for run-time NoC monitoring [4, 5, 19, 20], which must be accounted for in the design phase. This is in turn driven by debugging [4, 5] and performance monitoring/Quality of Service (QoS) [19, 20, 22].

With the introduction of NoCs the on-chip communication becomes more sophisticated relying on run-time programmable solutions. In centralized bus-based systems a single bus monitor is enough to be able to track the whole history of the system. In NoC-based SoCs, due to the inherent parallel behavior of communications, where multiple pipelined parallel communications may exist between IPs, multiple monitors have to be employed. The problem of how many such monitors are needed, their automatic placement in the NoC-based SoC by means of a monitoring-aware NoC design flow and the associated area cost implications have not been previously investigated.

Monitors and the traffic they generate are traditionally added non-intrusively into the SoC by using a separate monitoring NoC [19]. The cost of such a solution is high however, and a more efficient solution is use the same NoC for both monitor data and user data, as suggested in [4, 5, 20]. When monitoring traffic uses an interconnect of its own, it can be dimensioned after the user data NoC is designed. This merely adds an extra step in the design flow. However, when monitor and user data must share the same NoC, the overall design flow must be revised [5].

NoC design flows for ASIC type designs are normally split in several steps as topology selection, mapping, path selection and slot allocation [3,8,11,18]. Some design flows may omit or combine various steps. Each step adheres to the decisions taken in the previous steps. As prerequisites for NoC design, communication requirements must be derived, and the set of IPs to be connected to the NoC must be specified. In the topology selection step, the router network together with the bordering NIs are generated, based on the previously derived communication requirements. Using this topology together with the IP specification, the binding of IP ports to NI ports is done in the mapping step. In the path selection step, paths are allocated for all the communication flows specified, and in the slot allocation step each of the flows gets its own TDMA time slots for the traversed NoC links.

We have two interdependent problems: the one of functional dimensioning of the NoC and mapping of cores while accounting for their communication requirements, and the other of monitor placement and monitoring bandwidth specification. If these two problems are solved sequentially, the monitoring communication requirements can be precomputed. However, if the communication requirements of the monitors do not fit directly on the generated application NoC, a new NoC must be generated, e.g., by increasing the topology and repeating the process. However, by increasing the topology, the number of NoC routers increases. In turn, the mapping, path selection and allocation of resources may change and the number of required monitoring probes may increase as well (e.g. if probing all routers is required) and their communication requirements may change. In the mentioned cases the monitoring problem (whether driven by debugging or by run-time performance analysis) must be solved within or at least tightly coupled with the NoC design process. The task of placing the monitors must therefore be automated and integrated in the NoC design flow.

Contribution. We propose a monitoring-aware NoC design flow able to take into account the monitoring requirements at all steps in the NoC design flow. We illustrate this with a debug driven monitoring case study. Simple, area-efficient transaction monitors, attached to selectively chosen NoC routers, are used to enable debugging of the NoC-based SoC at transaction level. This is one of the most difficult cases, where the monitoring requirements are only known after the path selection step. In the context of application specific designs, the proposed flow is able to automatically insert transaction monitors, by determining the number and placement of these transaction monitors and accounting for their communication requirements. The smallest NoC which satisfies the application requirements, as well as the monitoring requirements is generated as a result. The area implications are quantified and compared to original NoCs without monitoring. The efficiency of the flow is shown on several realistic examples.

2 Related Work

In [19], the use of end-to-end monitors is proposed in order to assist the operating system controlling the NoC. The work focuses on the use of such performance monitors to optimize communication resource usage. The monitored data uses a separate NoC, called the control NoC instead of the application NoC.

[20] uses router performance monitors to keep track of the network utilization. By means of a network manager this information is made useful to a QoS manager to increase/decrease the quality levels of running applications. The monitored performance data uses the same NoC as the user data.

[4] proposes a generic NoC monitoring service comprising monitors attached to NoC components, routers or NIs, offered by the NoC. Targeted at debugging, it focuses on generic concepts of the service, architectural and general cost implications. The monitored data uses the same network as the user data.

[5] shows that using the same interconnect for the user traffic and monitoring traffic is area-efficient but may require modifications in the NoC design flow. However, it falls short on showing how to solve this problem in general and what are the associated cost implications.

All previous works assumes that: (1) the placement of the monitors is known, (2) the monitoring generated traffic or communication requirements are known in advance, (3) this traffic fits on top of the user traffic on the shared NoC or (4) on a separate NoC.

For monitoring, in general, these assumptions are not valid. The number and placement of monitors and their associated monitoring communication requirements are usually not known beforehand, but only after the NoC to be probed has been fully designed, or at least some steps in the NoC design flow have been performed. For example, some requirements may be known only after topology generation, such as the number of routers employed in the NoC, which is relevant if all routers or a coverage of routers need to be probed e.g. with router monitors showing link utilization. In this case the number of routers determines the number of probes and their placement, while their communication requirements are fixed, depending only on the number of links being traced. Other communication requirements may be known after the path selection step in the design flow, e.g. router monitors able to trace a connection, e.g. the functional traffic for debug reasons (or for connection utilization). In this case, assuming a desired full coverage of the connections, the number of probes and their placement is given by the routers in the cover. Their communication requirements depend on the number of connections passing the probed router and their sizes. We propose a monitoring aware design flow that fully integrates the design of the NoC and its monitoring service, solving all the above mentioned issues.

3 Architectural Platform

3.1 NoCs and Æthereal

NoCs comprise two components: routers (R) and network interfaces (NI), as depicted in Figure 1. The routers can be randomly connected among themselves and to the NIs (i.e., there are no topology constraints). Note that in principle there can be multiple links between routers. The routers transport packets of data from one NI to another.

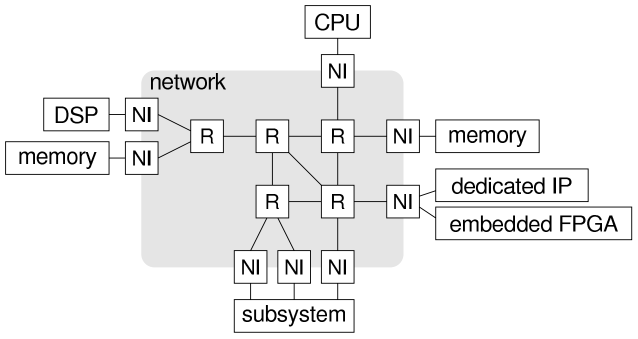


Figure 1. Example NoC

The NIs enable end-to-end services [21] to the IP modules and are key in decoupling computation from communication [2, 23]. The NI allows the designer to simplify communication issues to local point-to-point transactions at IP module boundaries, using protocols natural to the IP [23]. They are responsible for (de-)packetization, for implementing the connections and services, and for offering a standard interface (e.g., AXI or OCP) to the IP modules.

We use the Æthereal NoC [8, 9] as an example for our work. The Æthereal NoC runs at 500 MHz and offers a raw link bandwidth of 2GB/s in a 0.13μm 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. The Æthereal NoC allows the mapping of potentially multiple IPs per NI and potentially multiple NIs per router with any topology.

The interconnected IPs interact with each other by means of transactions, which are read and write transactions from IPs. Transactions consist of one request message and one optional response message. E.g. a request message can be a write message. 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. Transactions are performed on connections, consisting of one request and one response channel. The paths of request and response channels may be different.

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.

Packets are further split into flits, the minimum flow-control unit between hops. One flit corresponds to one TDMA slot.

3.2 Transaction Monitoring

3.2.1 The Transaction Monitoring Problem

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.

The problem of how many transaction monitors are needed relates to the desired coverage of the user communication flows. In general a full coverage is desired. However, it is prohibitively expensive to duplicate all traffic in the NoC; therefore the coverage may be full but has to be selective at certain moments in time. This means that the monitors must cover all channels, but not at the same time. At run-time, any (potentially more) of the desired channels can be selected to be monitored. The number of simultaneously active monitors in the system is bounded by the number of monitors deployed, as each monitor can only track a single channel.

The problem of the cost implications of the monitoring relates to the area of the monitors, the number of monitors involved and also to the area of the resulting NoC which supports both the application and monitoring communication requirements. The resulting NoC, potentially larger than the original NoC, accounts for the extra NIs, NI ports or enlarged topology to support monitoring in addition to the application communication.

3.2.2 NoC Monitoring Service

We use a monitoring service (NoCMS) as described in [4]. 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, see Figure 2 for details. The probe modular design comprises three parts: the sniffer (S), the event generator (EG) and the monitoring network interface (MNI). The MNI can be a separate NI or it can be merged with an existing NI. The monitoring service access point (MSA) is an IP which controls the configuration of the monitors at run-time and receives the monitored data from all monitors. E.g. the MSA can stream this data outside the chip through a debug port. The NoCMS is configured by means of probe programming via the NoC using memory-mapped I/O write transactions. The generic NOCMS concepts must be instantiated for the monitoring task at hand, in our case transaction monitoring. This implies the replacement of the EGs with transaction monitors, the placement of these monitors to offer a full channel coverage of the system, the placement of the MSA, and the the dimensioning of the communication requirements of the monitors (as this data should go to the MSA via the NoC). We use centralized monitoring with a single MSA.

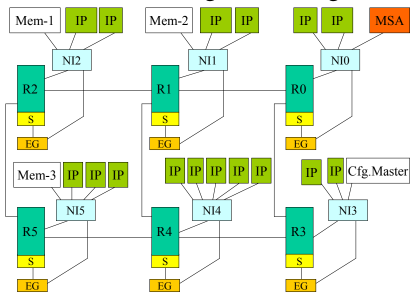


Figure 2. NoC Monitoring Service

3.2.3 Transaction Monitors

The transaction monitors can be attached to routers or NIs. For simplicity we only consider them as attached to routers. They can ultimately track transactions over a single channel passing any of the router's links. The monitors can be (re-)programmed at run-time to track any channel. They consist of a configuration block and a set of five pipelined filters. All run-time settings are done through the configuration block.

The raw data is provided to the transaction monitor by the sniffer, which captures it from the router links. The link of interest can be selected at run-time by configuring the first filter. The flits can be further filtered as BE or GT in the second filtering block. Further filtering of flits is done by identifying a single connection from the set of connections sharing the same link, in the next filter.

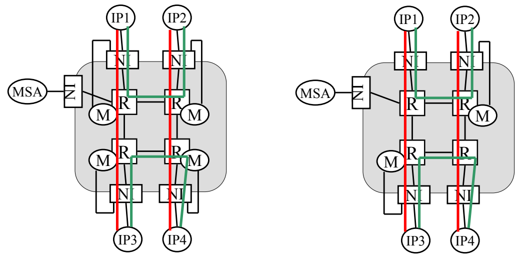
Transactions are composed of messages. Message identification allows to see, from within the NoC, when a write or a read message has been issued and from where or to which of the IPs or memories. Messages are payload packed in packets. Therefore, message identification requires depacketization, a procedure usually done at the NI. For the fourth filter, which is the essential one to provide transaction monitoring, we reused available Æthereal hardware modules for depacketization. The fifth filter has abstraction capabilities and is not discussed here because the details are not important.

A 0.13μm CMOS technology implementation of a transaction monitor supporting the first four filtering stages shows an area cost of $0.026mm^2$. Assuming that no filtering/abstraction is done locally at the monitor, the bandwidth requirements of the transaction monitors are comparable with the bandwidth of the monitored connection.

4 Application-aware placement

Since we are considering ASIC-like design, the application is known at design time. For the NoC-based SoC it means that also the set of connections (all request and response channels) is known at design time. The bandwidth and latency constraints of the channels are determined beforehand by means of static analysis or simulation.

At least one probe is required on the path of each channel, regardless whether it is a request or response channel. This means that any of the existing channels can be probed, achieving a full channel coverage. Furthermore, the concurrent observation of multiple channels is only limited by the number of probes in the NoC. We can simultaneously monitor one channel per probe. At run-time, the monitored channels per probe may change by means of programming the probes. This selectivity is acceptable as usually not all streams are required to be monitored at once (duplicating all traffic, even at a high abstraction level is prohibitive).



(a) *Naive placement of monitors*

(b) *Optimal placement of monitors*

Figure 3. Placement of Transaction Monitors

In ASIC design, a full coverage of routers with monitors may potentially be avoided, see for example the four monitors in Figure 3(a) covering each one of the four channels, versus the two monitors in Figure 3(b) covering each the two channels passing through. This leads to a reduction of the total monitoring solution area cost. Note that even assuming a full coverage of NoC routers with transaction monitors the communication requirements of these monitors are not known before the path selection step in the NoC design flow, as we do not know earlier what channels will pass through each of the monitored routers. Therefore, the problem of modifying the design flow to support monitoring constraints cannot be avoided.

5 Design Flow

5.1 UMARS

UMARS [11] is a QoS constrained NoC design algorithm. It unifies the three resource allocation phases: spatial mapping of cores, spatial routing of communication, and the restricted form of temporal mapping that assigns time-slots to these routes. UMARS considers the real-time communication requirements, and guarantees that application constraints on bandwidth and latency are met.

UMARS is a greedy algorithm, iterating over the monotonically decreasing set of unallocated channels until they are all accommodated in the NoC, or until allocation failed. The algorithm, as outlined in Algorithm 5.1, never backtracks to reevaluate an already allocated flow, enabling runtimes in the order of milli-seconds.

Algorithm 5.1 Outer loop of UMARS

```
1. While there are unallocated channels
     (a) Select the channel with highest bandwidth
     (b) Find a mapping and a path
     (c) Select slots on this path
```

An important property of UMARS that we exploit in this work is the fact that channels are allocated ordered on their bandwidth requirements. This is done as it: 1) helps in reducing bandwidth fragmentation [14], 2) is important from an energy consumption and resource conservation perspective since the benefits of a shorter path grow with communication demands [12], 3) gives precedence to flows with a more limited set of possible paths [12]. This ordering assures us that no channel succeeding the one currently being allocated has higher bandwidth requirements.

5.2 Monitoring-Awareness

The proposed monitoring aware NoC design flow is depicted in Figure 4. The coupling of mapping, path selection and time-slot allocation from the original UMARS is extended with the mapping of transaction monitors to routers such that a full coverage of user channels is achieved. Here, we do not discuss the original UMARS mapping, routing and slot allocation; for these refer to [11].

As a *preprocessing* step to the modified UMARS, transaction monitors are virtually added to all routers (as this would be the maximum set of transaction monitors that we consider). These virtual monitors are added to the set of IPs present in the system. They are connected to the closest local NI, attached to the router they monitor.

Due to the centralized monitoring used, a single MSA is further added to the set of IPs and it gets its own NI. A single GT connection is assumed from any monitor to the MSA although yet of unknown required bandwidth. We consider monitoring connections as latency insensitive, so no latency constraints are added to them.

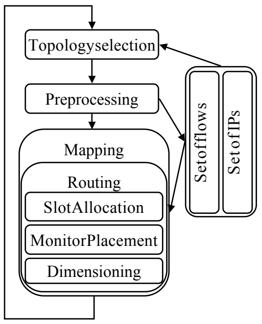


Figure 4. Monitoring-aware design flow

Monitor Placement. The loop of Algorithm 5.1 is extended with a fourth step, after a channel is allocated. This step is described in Algorithm 5.2. First, we check whether we need to insert additional monitoring. If the channel passes through a router that is monitored, we know, as channels are traversed in decreasing bandwidth order, that the monitor is able to monitor also this channel. Hence, nothing changes in this case. However, if none of the routers that the channel passes through are yet monitored, we select one in Step 1a of Algorithm 5.2. We select a router with the highest arity on the channel path, because it maximizes the number of potential observed channels for this monitor. Once we select the router to be probed we are sure that the router will stay in the final set of transaction monitors. Therefore, the virtually probed router is added to the set of probed routers.

In Step 1b of Algorithm 5.2 we then add a channel from the now monitored router (and its associated NI) to the MSA. This channel is added to the set of unallocated flows.

Dimensioning. The requirement in terms of bandwidth is derived as a function of the channel that mandated the insertion of the probe. Note that the way in which the communication requirements are dimensioned does not impact the overall proposed design flow. For the transaction monitoring example we set the traffic numbers for the monitoring channels equal to the bandwidth required by the monitored channel. The next channel to be monitored by the same monitor, whose monitoring channel has been allocated, is guaranteed to require a lower bandwidth. As one monitor can only monitor one channel at a time, the previously allocated monitoring channel would be reused. The same holds if the monitoring channels would require, e.g. 10% of the monitored connection bandwidth, due to a higher abstraction power of the monitors.

Algorithm 5.2 Step four

```
1. If the path does not pass a monitored router
    (a) Select a router on the path
    (b) Add a channel from this router to the MSA
```

The newly added channel is a monitoring channel. The only difference between a genuine user channel and a monitoring channel is that we only want to monitor the user channels and not the monitoring channels themselves. Besides allocating the user and monitoring channels we also take care not to monitor the monitoring channels. Therefore, Algorithm 5.2 is only executed for user channels.

Results. If UMARS completes the allocation successfully, we have as results the mapping, routing, slot allocation, monitor placement and monitoring dimensioning. After UMARS completes the allocation for all flows, all the routers in the set of probed routers have monitors attached. All the rest of virtual monitors are removed, as well as all the unallocated monitoring flows.

Iterations. If an allocation was not found by varying the slot table size till some predefined upper limit, the topology can be increased and the process repeated.

6 Experiments

6.1 Application Examples

Real Examples. We have used two real applications. *(mpeg)* an mpeg2 encoder/decoder using the main profile (4:2:0 chroma sampling) at main level (720x480 resolution with 15Mb/s) supporting interlaced video up to 30 frames per second. This application consists of 15 processing cores and an external SDRAM, and has 42 channels (with an aggregated bandwidth of 3GB/s), all configured to use guaranteed throughput, as presented in [8].

(audio) this application performs sample rate conversion, MP3, audio-postprocessing and radio. It closely resembles the chip presented in [16]. The application consists of 18 cores and has 66 channels all configured to use guaranteed throughput.

We have combined the two applications into four cases to be used as examples: mpeg (*Design1*), mpeg + audio (*Design2*), 2 $\times$ mpeg + audio (*Design3*), 4 $\times$ mpeg + audio (*Design4*).

Synthetic Examples. We have also generated synthetic application benchmarks for testing our proposed design

Table 1. Real Examples

Designs	*area*	*inc*	*size*	*mon*	*st*
1NI/R					
Design1	5.15	-	2x4	-	21
Design1+M	5.43	+5.5%	2x4	5	27
Design2	8.75	-	3x3	-	30
Design2+M	10.16	+16.1%	3x4	10	27
Design3	12.03	-	3x4	-	44
Design3+M	13.95	+16%	3x4	9	60
2NIs/R					
Design1	4.03	-	1x4	-	21
Design1+M	4.12	+2.2%	2x2	3	20
Design2	7.88	-	2x3	-	20
Design2+M	8.2	+3.9%	2x3	6	20
Design3	10.82	-	3x3	-	22
Design3+M	11.64	+7.6 %	2x4	8	29
3NIs/R					
Design1	3.62	-	1x2	-	30
Design1+M	3.85	+6.3%	1x3	3	18
Design2	6.97	-	1x3	-	27
Design2+M	7.16	+5.4%	1x3	3	30
Design3	10.26	-	2x3	-	21
Design3+M	10.78	+5%	2x3	6	22
Design4	18.45	-	3x4	-	21
Design4+M	19.07	+3.4%	2x4	8	36

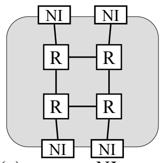
flow. These benchmarks are structured to follow the application patterns of real SoCs. We have generated applications into two classes of such benchmarks, as presented in [17]: (i) Spread communication benchmarks (*Spread*), where each core communicates to a few other cores. These benchmarks characterize designs such as the TV processor that has many small local memories with communication evenly spread in the design. (ii) Bottleneck communication benchmarks (*Bottleneck*) where there are one or multiple bottleneck vertices to which the core communication takes place. These benchmarks resemble designs using shared memory/external devices such as the set-top boxes.

We have used spread communication of 12 IPs, in which every IP communicates with three others. We have used bottleneck communication with two converging points and 12 IPs. We have generated 500 synthetic application examples with spread and bottleneck communication.

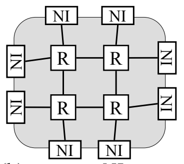
6.2 Results

6.2.1 Setup

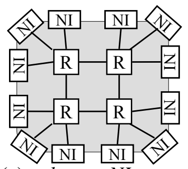
For both the real and synthetic application examples we have investigated what the original UMARS vs. monitoring-aware UMARS output is. The original UMARS generates the minimal NoC on which only the application requirements fit, while the monitoring-aware UMARS generates the minimal NoC on which both the application and monitoring requirements fit. To evaluate the performance of our approach, we looked at: *(i)* required number of transaction monitors,*(ii)* resulting topology size and *(iii)* resulting area.



(a) *one NI per router*



(b) *two NIs per router*



(c) *three NIs per router*

Figure 5. NIs per router

For each application we have evaluated all possible meshes, from one by one up to seven by seven. For each of these topologies we have added one, two and three NIs per router, as depicted in Figure 5 and evaluated slot table sizes up to 65 TDMA slots. A larger slot table size mitigates overprovisioning due to granularity, but is often associated with a growth in buffer sizes as network consumption tends to become more bursty. Out of all the configurations for which UMARS finds an allocation, we present the one with lowest total area cost.

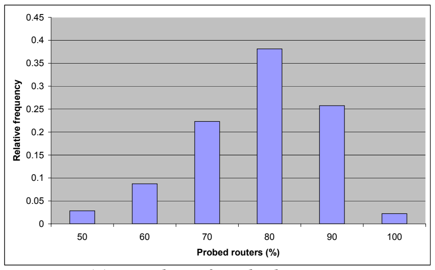
Table 1 summarizes the results for the real examples, when one, two or three NIs per routers are tried. Due to the large communication demands, and given the constraints on topology and slot-table size we set for our experiments, *Design4* only fits on a topology using three NIs per router. For the synthetic examples Figures 6 and 7 summarize the results for bottleneck and spread communication respectively.

6.2.2 Number of transaction monitors

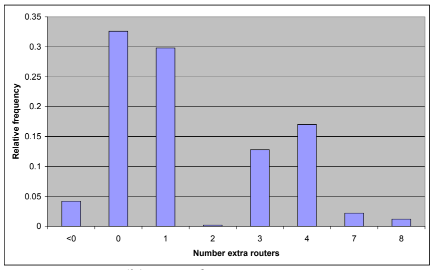
For the synthetic cases with bottleneck communication, we see that the number of routers needed to be probed for full coverage varies between 50% and 100% with an average of 75%. Figure 6(a) displays the distribution. For spread communication Figure 7(a) displays the distribution. We see that the number of routers requiring a probe is higher compared to the bottleneck cases, but that is no surprise as the communication is more balanced (spread out) over the routers. The minimum is 60% while the maximum is 90%. Hence, the interval is narrower than with bottleneck communication, the maximum is actually lower, and coverage of all routers was never required. Looking at the diagrams it is obvious that the number of routers needing probes is focused around the 80-90% bins.

Please note that the number of transaction monitors required is high because the Æthereal NoC allows multiple IPs to be connected to the same NI and multiple NIs to be connected to the same router. Therefore, channels can be very short, e.g. a channel between a master and a slave connected to the same NI will go through the NI starting from the master, then through one router and back to the same NI to the slave. All routers having at least one channel like this passing through will require one transaction monitor. Other NoCs may require a channel to pass through two different NIs, potentially lowering the number of transaction monitors being required.

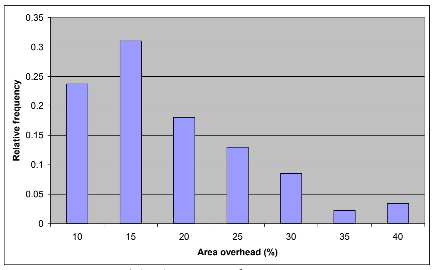
For the real examples, see column *mon* showing the number of monitors and compare it to column *size* show-



(a) *Number of Probed Routers*

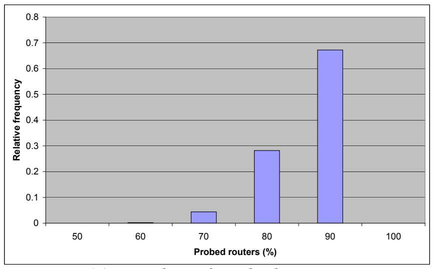


(b) *Topology Increase*

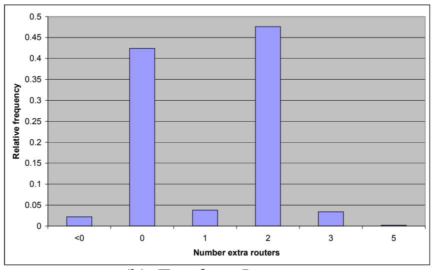


(c) *Area Implications*

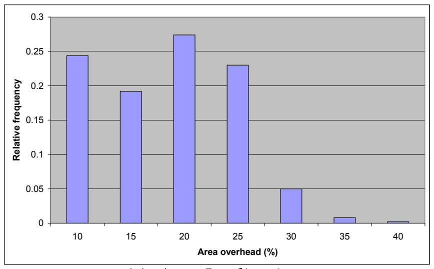
Figure 6. Bottleneck



(a) *Number of Probed Routers*



(b) *Topology Increase*



(c) *Area Implications*

Figure 7. Spread

ing the mesh size. On average 87% of the routers need to be probed, but full coverage of routers with probes was required in 60% of the cases. Relating this with the area numbers from the same table, it is interesting to observe that the most area-efficient solutions required all routers probed. Therefore probing all routers must not be associated with area-inefficient solutions, the number of monitoring probes (in our case transaction monitors) being just one component which influences the total area cost of the monitoring solution.

6.2.3 Topology size

For the topology size we looked at the total number of routers employed. Figures 6(b) and 7(b) display the distribution for the synthetic examples. On average, topology stays the same (no extra routers required) or one or two extra routers are required. Increases in topology size with more than two routers, but with a maximum of 8, are still required in other cases, especially in the bottleneck applications. This can be explained because in bottleneck designs it is harder to accommodate the new monitoring channels due to the existing bottleneck vertices. Interesting is the fact that in 3-4% of the cases the number of routers actually decreased. This we can attribute to the heuristic nature of UMARS and to the higher number of slots used in the NoCs with monitoring.

For the real examples we see the number of routers kept constant in six cases, and both an increase and a decrease in two cases. The latter is accountable to an allocation found with a higher slot table size, see column *st* in Table 1.

In both real and synthetic examples we see that there is a good chance(30-60%) to find a solution on the same NoC topology, without requiring extra routers.

6.2.4 Area

The total NoC area is derived according to the model in [7] extended with the area of the transaction monitors, $0.026mm^2$ per monitor in 0.13μm CMOS technology. Note that the total area presented includes NIs, routers and probes (transaction monitors). The area of NIs also accounts for buffer sizing in the NIs/NI ports corresponding to the real communication requirements of the users and monitors. The area numbers do not include the area of other IPs in the SoC, but refer to the NoC together with the complete monitoring service.

For bottleneck communication, area wise the cost is continuously below 50% with an average of 15%. Figure 6(c) shows the distribution of area overhead over the test cases and it is obvious that most lie in the left half of the span.

For spread communication Figure 7(c) shows the distribution. From an area point of view the overhead is between 10% and 40%, which again is a narrower interval than for bottleneck communication. In all it ends up on an average of 15% also for uniform traffic. No major difference in the area overhead is noticeable between uniform and bottleneck communication.

For the real examples the total area increase, see column *inc* in Table 1, amounts to between 2.2% and 16.1%. The

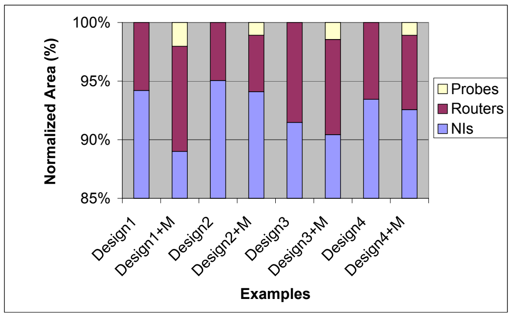


Figure 8. Overall distribution of area

area overhead is between 3% and 7% in the most area efficient case of three NIs per router which succeeded for all four designs. The resulting four designs we consider the end results of the monitoring-aware NoC design flow.

It is also interesting to see the overall distribution of this area between NIs, routers and monitors. This is presented in Figure 8 for the four designs in their most area-efficient case using *3NIs/R*. For the original designs the distribution of area between NIs and routers is shown. The main remark is that in all cases area of the transaction monitors is insignificant relative to the total area of the designs, dominated by the area of the NIs. Furthermore, in all designs the area of the monitors is even several times lower than the area of the routers involved.

7 Conclusion

We propose a NoC design flow in which monitoring is taken into account at design time and is fully integrated in the flow, automating the insertion of the monitors whenever their communication requirements are known, leading to a monitoring aware NoC design flow. Our flow was exemplified with the concrete case of transaction monitoring, in the context of the Æthereal NoC and UMARS design flow.

We are the first to quantify the complete cost of the complete monitoring solution accounting for the monitors, extra NIs, NI ports or enlarged topology needed to support monitoring in addition to the original application communication. Results show an area efficient solution for integrating monitoring in NoC designs. Monitors alone do not add much to the overall area numbers as the designs remain dominated by the area of NIs.

References

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

[2] T. Bjerregaard and J. Sparsø. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1226–1231, Mar. 2005.

[3] 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.

[4] 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.

[5] 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)*, May 2006.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] A. Hansson, K. Goossens, and A. Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 75–80, Sept. 2005.

[12] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, pages 688–693. IEEE Computer Society, 2003.

[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] I. Matta and A. Bestavros. A load profiling approach to routing guaranteed bandwidth flows. In *IEEE INFOCOM*, volume 3, pages 1014–1021, Mar 1998.

[15] 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.

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

[17] S. Murali, M. Coenen, A. Rădulescu, K. Goossens, and G. De Micheli. A methodology for mapping multiple use-cases on to networks on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Mar. 2006.

[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.

[19] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet. Operating-system controlled network on chip. In *Proc. Design Automation Conference (DAC)*, pages 256–259, 2004.

[20] M. Pastrnak et al. Combined reservation and adaptation QoS for improving picture quality and resource usage of multimedia (NoC) chips. In *International Symposium on Consumer Electronics*, 2006.

[21] A. Rădulescu, J. Dielissen, S. González Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 24(1):4–17, Jan. 2005.

[22] J. W. van den Brand. Runtime networks-on-chip performance monitoring. Technical Report 2006/00218, Philips Research, Mar. 2006.

[23] D. Wingard. Socket-based design using decoupled interconnects. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 15, pages 367–396. Kluwer, 2004.