ROAdNoC: Runtime Observability for an Adaptive Network on Chip Architecture

Mohammad Abdullah Al Faruque, Thomas Ebi, and Jörg Henkel
University of Karlsruhe, Chair for Embedded Systems, Karlsruhe, Germany
{alfaruque, ebi, henkel} @ informatik.uni-karlsruhe.de

Abstract— **Hard-to-predict system behavior and/or reliability issues resulting from migrating to new technology nodes requires considering runtime adaptivity in future on-chip systems. Runtime observability is a prerequisite for runtime adaptivity as it is providing necessary system information gathered on-the-fly.**

We are presenting the first comprehensive runtime observability infrastructure for an adaptive network on chip architecture which is flexible (e.g. in choosing the routing path), hardly intrusive, and requires little additional overhead (around 0.7% of the total link bandwidth). The hardware overhead is negligible, too, and is in fact less than the hardware savings due to resource multiplexing capabilities that are achieved through runtime observability/adaptivity. As an example, our on-demand buffer assignment scheme increases the buffer utilization and decreases the overall buffer requirements by an average of 42% (the buffer area amounts to about 60% of the entire router area [19]) in our case study analysis compared to a fixed buffer assignment scheme [7]. Our runtime observability on an average also increases the connection success rate by 62% compared to the case without runtime observability for the applications from the E3S benchmark suite [6]. We show the advantages obtained through runtime observability and compare with state-of-the art communication-centric designs.

I. Introduction and Motivation

The 100 Billion transistor chip is predicted to emerge within a decade [3]. It will allow for integration of hundreds or even thousands of processor cores on a single die. It is obvious that such a large number of cores requires a sophisticated on-chip communication architecture. Hence, it is anticipated that future designs need to be *communication-centric* [3]. The fact that interconnects need special attention even in current Multi Processor Systems on Chip (MPSoCs) has already been recognized several years ago when research started to focus on Networks on Chip (NoCs) [5], [9]. Application specific NoCs [1], [14], are design-time parameterized architectures with a custom topology, fixed routing scheme, and a fixed number of allowed virtual connections at each output port [1], [7]. They are generally tailor-made for a certain application or an application domain and fail in scenarios of hard-to-predict system behavior and/or in situations where reliability is a concern. Some scenarios are as follows :

- The system constraints may change during runtime.
- The user of the system may change their pattern of how to operate/use the system.
- Smaller feature sizes in the nano age will cause reliability concerns. It will require building future reliable systems out of un-reliable components [3].

A reliable communication-centric System on Chip (SoC) may, for example, depend upon the ability of the NoC to route traffic in such a way that it can efficiently bypass faulty areas at runtime. All these scenarios – from user behavior to reliability issues – require designing systems with adaptivity capabilities in mind which allow application variations and to react on faulty situations accordingly. Adaptivity is required in both the system-level as well as in the architecture-level where it is realized through modification thereof, or even new paradigms in architectural design. We consider the software part between the application and the underlying hardware layer executed in the processing element as the system-level and the data transmission part which is implemented in hardware as the architecture-level. Changes in user behavior, system constraints, and/or reliability issues can be effectively compensated at system-level by, for instance, dynamically (re-)mapping a running application at runtime. Architecture-level modifications on the other side may help to increase the resource utilization at runtime as proposed in [7].

In order to assure a certain degree of *quality-of-service* (e.g. guarantees in performance and bandwidth), a feedback of the current system state must be available. This can be achieved through runtime observability in an adaptive system. A runtime observability infrastructure with small hardware and communication overheads would be more than compensated by the degree of freedom achieved using adaptation. Within this paper **we propose** an event-based NoC monitoring component[1] at architecture-level that offers runtime observability. The prime challenges for runtime observability are scalability, flexibility, non-intrusiveness, real-time capabilities, and cost. For the monitoring components to be as non-intrusive as possible, they need to keep their interference with normal system execution (*probe effects*) [12] at a minimum. An example of these effects would be the sending of monitoring packets[2] through the regular data network. If these packets are injected too rapidly, they demand resources which otherwise may have been used for regular traffic. It is therefore necessary to limit monitoring traffic by keeping its bandwidth usage and occurrence frequency minimal.

The rest of the paper is organized as follows. After presenting our novel contribution and related work in II, in III we introduce our *adaptive on-chip communication architecture*. In IV our novel *ROAdNoC* infrastructure is explained in detail. Our hardware implementation for the monitoring components is shown in V. Experimental results are discussed in VI with VII concluding the paper.

II. Related Work and our Novel Contribution

Runtime adaptivity in both the system-level and the architecture-level considering the user behavior and reliability issues, is a relatively new aspect of SoC design introduced in [7], [10]. Recently, several general-purpose NoCs such as Tile64™, an embedded multicore by Tilera [19], and an 80-core general-purpose processor from Intel [11] have been proposed. They are design-time parametrized (e.g. the number of output ports and the worst case amount of concurrent virtual connections in a single output port) but focus more on general-purpose issues and are hardly capable of changing different architecture-level parameters such as buffer assignments to different output ports on-demand and thus suffer from low

[1] In this paper we denote *runtime observability* as a complete infrastructure and *monitoring* as a hardware component attached to each tile.

[2] This is the traffic that is generated during runtime observation of the system state and is described in detail later in this paper.

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

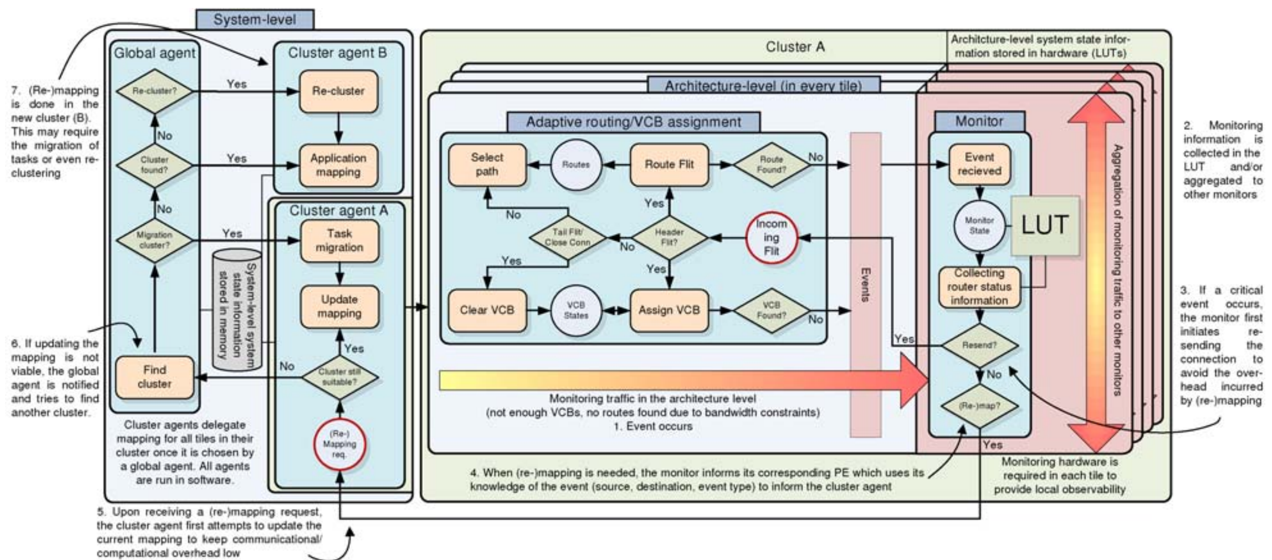

Fig. 1. Overview of our adaptive on-chip communication architecture

resource utilization. They also lack a sophisticated resource management scheme (e.g. runtime application (re-)mapping).

There is also related work in the domain of on-demand interconnection schemes in different problem spaces. In [18] the authors proposed to provide interconnection on-demand by adapting the physical network. The required number of links increases exponentially relative to the number of processing elements making this approach non-scalable. In [16] the authors have proposed a power-aware network whose links are turned on and off on-demand in response to bursts and dips of traffic. Their approach assumes that future traffic characteristics are predicable based on recent traffic patterns which may not be possible in situations similar to those we itemized earlier. In [2] the authors present a dynamic communication infrastructure which routes traffic around modules placed dynamically on a reconfigurable device. It is built on top of a reconfigurable hardware, i.e. on an FPGA and it is limited to such devices.

In summary, it can be stated that observability capabilities for on-chip communication have not been proactively investigated in the NoC domain. In [13] authors have mentioned an operating system controlled on-chip runtime collection of traffic statistics at the Network Interface (NI) to optimize the usage of communication resources in a NoC using a centralized resource management scheme. In [4] authors have presented a generic event-based NoC Monitoring Service (NoCMS) for Æthereal. It is not designed specifically to detect faults in network traffic during runtime adaptation but instead to gather NoC behavior statistics (debugging). In [17] authors further used the monitoring probes proposed in [4] for a new communication service to control congestion. In a nutshell, the Æthereal monitoring framework is not used to adapt the underlying on-chip communication architecture. Runtime observability is also not included in general-purpose NoCs (e.g. [11], [19]) as these architectures do not adapt at the architecture-level to increase resource utilization. Recently, authors in [15] have also focused on self-monitoring components for NoCs considering reliability factors.

We have integrated a runtime observability infrastructure for our adaptive NoC at the architecture-level. It analyzes the communication infrastructure during runtime and self-adapts depending on the monitoring traffic on when and how a certain router should be configured for a certain connection. Our runtime observability infrastructure on an average increases the connection success rate by 62% compared to having no runtime observability for the automotive application from the E3S benchmark suite [6]. The extra overhead that stems from the monitoring component is smaller than the hardware saving due to resource multiplexing in the architecture. Our on-demand buffer assignment scheme increases the buffer utilization and decreases the overall buffer use on an average of 42% in our experiment compared to a fixed buffer assignment scheme [7]. **Our novel contribution is as follows:**
To employ successful adaptation to the communication infrastructure needs to be observed. Therefore, to provide runtime observability for realizing a successful on-demand adaptation, we present a novel low cost runtime observability infrastructure. It is highly flexible and hardly intrusive.

III. Our Runtime Adaptive Approach

As most NoCs, our adaptive on-chip communication architecture is pipelined, utilizes packet-based communication, deploys wormhole routing, and has a regular 2-D mesh topology. The overview of our adaptive scheme shown in Fig. 1 is divided into two main parts along with the runtime observability infrastructure, the *system-level*, and the *architecture-level*.

The adaptivity at system-level is deployed using a runtime agent-based distributed application mapping scheme. An *agent* is a computational entity, realized in software, that acts on behalf of other entities. A detailed description of our agent-based runtime application mapping is presented in [8]. The architecture-level handles the *runtime routing algorithm* and the *on-demand VCB*[3] *assignment* besides the normal dataflow functionalities of the router. To accomplish a successful adaptation, both the system-level and the architecture-level require runtime observability.

A. System-level Adaptation

Our proposed agent-based distributed application mapping algorithm dynamically maps applications at runtime as needed. Therefore, one or more application tasks are mapped onto NoC tiles which fulfill the task's requirements (computation/communication). The detailed scheme is explained in [8]. To obtain a scalable mapping solution we have reduced the

[3] A *Virtual Channel* (VC) is a unidirectional virtual connection between two tiles and is realized by message buffers, *Virtual Channel Buffers* (VCB).

computation load by confining mapping to *clusters* which are a connected subset of NoC tiles. The clusters have a variable size that can be adjusted during runtime and each cluster has one *cluster agent* which is responsible for (re-)mapping.

Among others, there are two main reasons for (re-)mapping. The first is due to changing user behavior, i.e. new application tasks scheduled to run at a specific time t. The second is as a response to faults during adaptation reported by the monitoring component associated with the cluster agent. This causes the cluster agent to attempt to update the current mapping instance based on received information (i.e. connection source, destination tiles, and fault type). This limits the more expensive complete (re-)mappings to when they are absolutely needed. If the cluster agent is unable to find a new mapping instance it contacts a *global agent*. These special agents are responsible for cluster selection, coordination, and re-clustering. The global agent then first tries to resize the cluster associated with the cluster agent. If this fails, a different cluster is chosen and a new mapping is done. The actual mapping is accomplished using a heuristic which is also explained in [8]. All agents are implemented in software and may be migrated to run on any PE in every tile within their deployment area. All the information necessary to manage a cluster/global agent is stored in the local memory associated with the tile where the current instance of the cluster/global agent dwells. No extra hardware is necessary for managing this information and it is transmitted as regular system configuration traffic.

B. Architecture-level Adaptation

Once a mapping instance has been set up at the system-level, the architecture-level must then handle the resulting connections in every tile. For a requesting connection, the route is first checked in every possible direction and the *VCB* is assigned accordingly on-demand. The adaptive routing algorithm assigns each output port a weight based on available bandwidth, the horizontal distance, and the vertical distance between the current and the destination tiles which maximizes the number of sensible routing choices along its route.

Up until now, the number of *VCBs* at one port has always been fixed at design-time [1], [11]. With on-demand assignment (the runtime routing algorithm and on-demand buffer assignment schemes are explained in [7]), the *VCBs* are not tied to ports, but only to the router itself. The router may distribute the *VCBs* to any route as needed by assigning a connection to the *VCB* through the *Virtual Channel Arbiter* (VCA) and then assigning the *VCB* to an output port. The benefits of such an on-demand assignment are evident: through on-demand assignment, buffers are only assigned when needed meaning that *VCBs* can be reused by different ports and therefore, the buffer utilization increases and that decreases the overall buffer use on an average of 42% in our case study analysis compared to a fixed buffer assignment scheme [7].

IV. Runtime Observability Infrastructure

After explaining the main features of our NoC platform we now focus on the contribution of this paper. Our *Runtime Observability for an Adaptive Network on Chip (ROAdNoC)* that supports successful architecture-level adaptation is implemented using monitoring components inside each tile.

A. Events

ROAdNoC is event-based. Events are caused by failures in a subsystem of an individual router: i.e. the adaptive routing algorithm. The list of events are explained in the following:[4]

[4]The event list is generic and can be enhanced depending on the architecture. It covers our on-demand buffer assignment and routing algorithm.

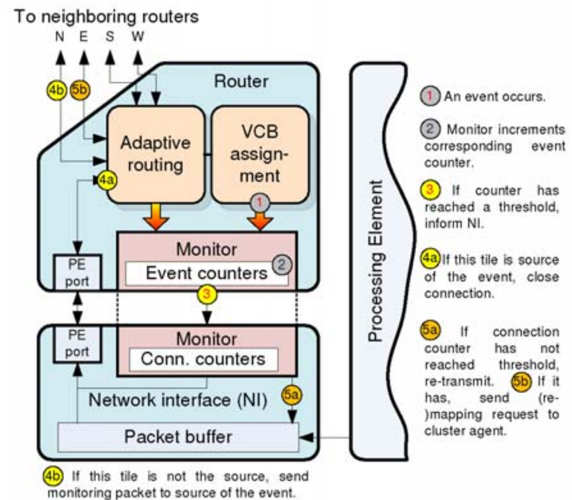

Fig. 2. Overview of the monitoring component

- **TTL-expire-event:** In order to assure deadlock-free routing, each packet is given a maximum *time-to-live* (TTL) hop count. If a packet fails to reach its destination within the TTL, it is removed from the network. The TTL is the Manhattan Distance plus a given maximum number of misroutes.
- **No-route-found-event:** If the routing algorithm fails to find any available routes inside a router, i.e. there are not enough available bandwidth slots in any direction, the packet is removed from the network.
- **No-buffer-event:** If the *VCA* fails to find a free *VCB* to hold the incoming packet it is removed from the network.
- **Buffer-full-event:** Occurs when the *VCA* already has assigned a *VCB* to a connection but cannot write to it because it is full. This does not directly result in packet loss but is a sign of congestion in the network. This situation is resolved automatically, however it should be observed and be reacted to if it persists.

Unlike in Æthereal [4], these events are used to identify the faults during NoC adaptation at architecture-level and are used to invoke the necessary steps to remedy it. The events given here are binary in nature; that is, either an event has occurred or not (except *buffer-full-event* which is invoked for a given specified value). This simplification eliminates the need for attributes to be supplied for events as with the monitoring component for Æthereal [4]. The *user-configuration-events* of Æthereal (high level communication configuration events such as *connection-opened* and *connection-closed*) are indirectly observed. However this is only done in order to set up the counters for each connection and to free them when the connection closes.

B. Design and Event Collection

The monitoring component of a router for our *ROAdNoC* consists of a look-up-table (LUT) containing a set of counters for each connection going through the router. These are tied to events which can occur in the on-demand buffer assignment and the adaptive routing part of the router and are incremented every time one is reported, thereby collecting data on events.

The counters are stored in the LUT with the corresponding *connection ID* and the source address of a connection. In particular, the source address can be the same address as the monitoring component if the corresponding PE is the source of the connection. This is a special case, as the counters

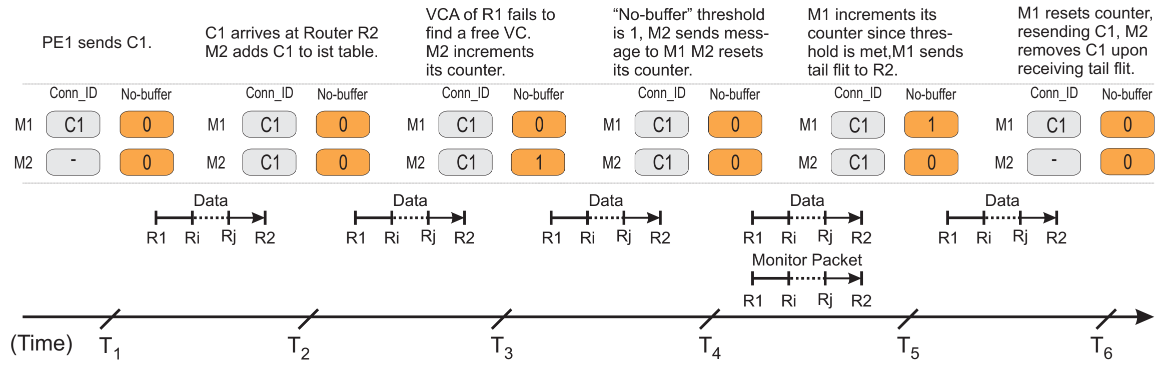

Fig. 3. Runtime observability capabilities of ROAdNoC

are not only incremented by events occurring within the router but also through messages received from monitoring components in other routers.

Algorithm 1 Aggregation and processing of monitoring traffic

input: event $e = \{$event type t, connection ID C, connection source $S\}$

definitions: X: current router; E: event queue
LUT[connection ID, event type]: event counter look-up-table
τ_t: given threshold for events of type t
δ: given threshold for re-sending
s_c: send counter in S for C; NI: network interface
CA: cluster agent associated with X

```
1: get next event e from E
2: event_counter ← LUT[C, t]
3: increment event_counter
4: if event_counter > τ then
5:   if X != S then
6:     send event message e = {t, C, S} to S
7:   else
8:     signal NI: send tail flit from packet buffer for C to close conn.
9:     if s_c < δ then
10:      signal NI: re-send packet for C
11:    else
12:      send (re-)mapping message {remap,S,t} to CA
13:    end if
14:  end if
15:  LUT[C, t] ← 0
16: else
17:  LUT[C, t] ← event_counter
18: end if
```

C. Aggregation and Processing

An adaptation fault occurs when an event counter reaches a certain value. The event aggregation and processing scheme is explained in Fig. 2 and the functionality upon problem detection is given in Algorithm 1. The aggregation is done through the *NI* by sending messages to the source of the connection. The processing is done partially in the *NI* and in the cluster agent. The *NI* takes care of re-transmission while the cluster agent is invoked if a (re-)mapping is needed. A time-line diagram portraying a certain scenario of the *ROAdNoC* infrastructure can be seen in Fig. 3.

T_1: The processing element PE_1 associated with router R_1 begins to send data to another PE. The *NI* assigns this connection the *connection ID* C_1. Thus, the monitoring component M_1 of R_1 adds this connection to its list of observed connections. Here, only one exemplary counter representing the event of *no-buffer* being available in a router is given. Its initial value is zero.

T_2: On its way to the destination the header flit of this connection arrives at R_2. R_2 is generally not reached after one hop; the header flit may already have been routed through other routers. Upon arrival of the header flit, C_1 is added to the connection table of M_2 at R_2.

T_3: The flit then progresses through R_2 until it reaches the *VCA* which fails to assign a free buffer for C_1. This event is reported to M_2 which increments its *no-buffer* counter. The *VCA* then simply discards the header flit and all subsequent flits and does not send a `NACK` signal as it would if the buffer were already successfully assigned but simply full. This prevents blocking in previous routers and a tail flit can arrive to close the connection.

T_4: For the *no-buffer* counter the threshold is one. That is, one *no-buffer-event* is enough to invoke a monitoring packet from the monitoring component. M_2 informs the sender of the connection through its *NI* which sends a monitoring packet to the monitoring component M_1 at time T_4. At the same time, M_2 resets its *no-buffer* counter for connection C_1.

T_5: Once the monitoring packet arrives at M_1 it increments its own counter for C_1 and, since the threshold is met, it informs its *NI* to close the current connection C_1. This is done by simply sending a tail flit.

T_6: M_1 has already reset its *no-buffer* counter and the *NI* is in the process of re-sending its data. The tail flit arrives at R_2 causing M_2 to remove C_1 from its connection table.

D. Monitoring Related Traffic

The monitoring component is situated partially between the router and the *NI* (Fig. 2). It is therefore able to interact with the *NI* to send its own packets over the regular communication network. This means, however, that the monitoring traffic must compete with regular transmissions for network resources. The monitoring packet must be of a higher priority to allow it to preempt regular connections in a *VCB*. When using only two priorities, one for regular traffic and one for monitoring traffic, a packet's priority requires a one-bit field in the header flit. For the rest of the fields we give an example monitoring packet in a 4×4 NoC. It is two flits in size and is composed of a regular header flit for transmission plus a tail flit with payload data containing at least the triggering *connection ID* and the type of event. In addition, it may also contain information such as the source of the transmission which is used to provide the cluster agent with additional knowledge it may exploit during (re-)mapping. It does not require a source field in the header since monitoring packets are not monitored themselves. The size of the monitoring packet can be calculated from the formulas given in Table 1.

Flit part	Size $n \times m$	Size 4×4
Type	2 bit	2 bit
Priority	$\log_2$ (priorities)	1 bit (2 priorities)
Destination	$\log_2$ $(n \times m)$	8 bit
TTL	$\log_2(n + m + 2x^\dagger)$	4 bit
BW	$\log_2$ (BW slots)	3 bit (8 slots)

$^\dagger x$ = number of misroutes

Table 1: Flit size in an $n \times m$ NoC and in a 4×4 NoC

The frequency with which monitoring packets are generated is also important. They are event-based and are only sent when an event occurs. Since events are only generated on faults during adaptation, there is no monitoring traffic when the network operates normally. Events can also eventually initiate (re-)mapping which comes with a high communication overhead. It is, however, also through observability that unnecessary (re-)mapping can be avoided compared to a scheme where any connection fault automatically calls for (re-)mapping.

V. Hardware Implementation

We implemented *ROAdNoC* and evaluated the area overhead on a XILINX Virtex2 FPGA [20] board. The event-counter values are stored in an LUT. One entry in the LUT ties the *connection ID* to the source of the connection and to its associated counters. If there are n *VCBs* and k inputs, then $(n+k)$ entries are needed. Connections are added to the LUT when a header flit arrives at a router (Fig. 4(b)). The arrival causes the set and configure flags to be triggered, initiating a write to the LUT and setting the counter values to zero. Similarly, a tail flit arriving at the router causes the *configure flag* to be triggered while the *set flag* remains zero. This causes the monitoring component to remove the connection from the LUT. Once an event occurs it initiates a *read* from the LUT using the event *connection ID* (Fig. 4(a)). It then compares the counter value returned from the LUT of the event type corresponding to the event type that arrived. If the counter value has reached its threshold, the *NI* part of the monitoring component is informed and the counter value is set to zero in the LUT. If not, the incremented value is written to the LUT.

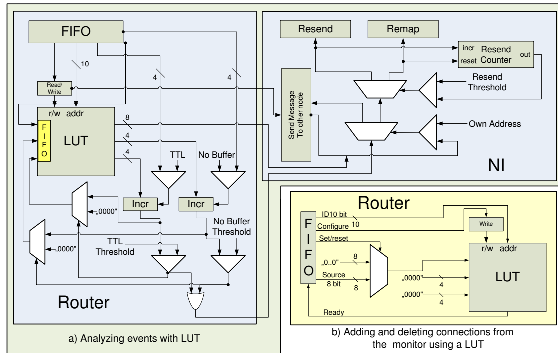

Fig. 4. Hardware for adding and analyzing monitoring events

The *NI* part of the monitoring component, upon receiving an event, first compares the connection source with its own address. If it is not the sender then a packet is sent to the remote sender. Otherwise, the connection send count is examined to find out if there are previous send attempts by comparing the *connection-send* counter stored in a register with a given re-send threshold. Based upon this, the *NI* is either told to re-send the packet if the threshold has not been met or a (re-)mapping is required. If the packet is re-sent, the *connection-send* counter is incremented. A (re-)mapping causes the counter to be reset to zero.

Each router has 5 input ports resulting in 5 possible simultaneous connections needing to be set up in a monitoring component. Also, using a LUT entails a few cycles delay in which new connections/events cannot be processed. To allow each tile to function using only one monitoring component, FIFOs are added to buffer its inputs.

VI. Results and Case Study Analysis

We have evaluated our *ROAdNoC* infrastructure with several parameters that directly influence the monitoring traffic and bandwidth usage:

- The *packet injection rate* determines the arrival frequency of the new packets to the network in each router.
- The *packet flit size* is responsible for the duration of traffic (along with the allocated bandwidth slots).
- The allocated link *bandwidth slots* per connection influence the number of simultaneous connections per link.
- The number of *VCBs* limits the number of simultaneous connections per router.

A number of assumptions are made to determine the simulation parameters: the traffic distribution used is uniform, the data packet size is 200 flits, the monitoring packet size is 2 flits, and the bandwidth is 20 slots. These parameters have been chosen to observe the effects of both *no-buffer-event* and *no-route-found-event*.

For the first simulation the data packet injection rates are based on allocated bandwidth slots. They are chosen as to supply a (near) continuous stream of data by using the highest possible rates. For instance, a connection allocated 1 slot out of 20 can at most send one flit every 20ns (20 cycles). Hence, the highest accommodatable data packet (200 flits) injection rate is one packet every 4us. The traffic is streaming with packet injection being normally distributed with some variance. This traffic is the worst case for *VCB* usage as continuous traffic also requires constant *VCB* assignment. In the simulation each router has 8 *VCBs*. For the adaptive routing algorithm, the worst case is any slot value greater than 10 as each link can only transmit one connection of this type. The simulation results (Fig. 5) show a gradually increasing monitoring packet injection rate for increasing traffic density. However, the monitoring traffic remains low considering the overall link bandwidth – less than 0.7%.

Fig. 6 shows the effect of different number of *VCBs* per router and allocated bandwidth slots on the monitoring packet injection rate. There is a clear distinction between the low bandwidth/continuous traffic on the left and the high bandwidth/burst traffic on the right. This is due to the *VCB* assignment being the dominant cause of monitoring events in the left part and the adaptive routing in the right part.

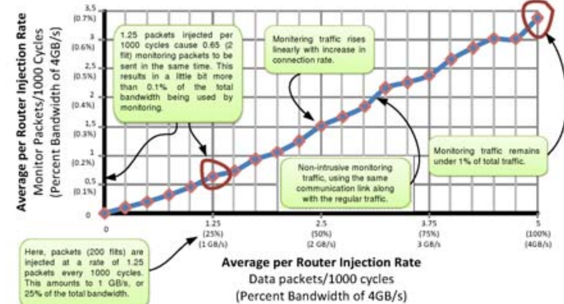

Fig. 5. Monitoring packets injection and traffic density

The traffic generated by *ROAdNoC* cannot directly be compared to that of the Æthereal monitoring component. The occurrence of events in Æthereal monitoring component is different to ours as they are mainly managing events necessary for debugging whereas we are managing different types of events that are needed to adapt the on-chip communication architecture. Using only *connection-opened-events* and *connection-closed-events* to calculate the resulting data rate assumes that all connections are set up successfully and specifically no *alert-events* occur. Under such circumstances our implementation would generate no traffic. For comparison we assume Æthereal to have a comparable routing algorithm which is able to choose alternative routes. It is assumed to produce Æthereal NoC *alert-events* when no route is found.

Furthermore, it is assumed that any failed connection attempts are resolved through re-routing or (re-)mapping if needed.

Taking the assumptions from [4] but expanding the traffic model by the number of successful connections setup by the initial attempt to set up 200 connections, the two approaches may be compared. To calculate the total monitoring traffic t_M we require the number of unsuccessful connections u per second, the number of total connections c (200) per second, the monitoring traffic for an unsuccessful connection t_u, and the monitoring traffic for a successful connection t_s. For the first attempt we calculate the monitoring traffic t_{M1}:

$$t_{M1} = u \cdot t_u + (c - u) \cdot t_s \tag{1}$$

Unsuccessful connections are assumed to be successful after they are re-routed causing the additional monitoring traffic, t_{M2} to be $u \cdot t_s$. By adding t_{M1} and t_{M2} we obtain the total traffic shown in Table 2. For the first comparison, the Æthereal monitoring component produces both *connection-opened/closed-events* for successful connections and *alert-events* for unsuccessful ones. The unsuccessful ones then also produce *connection-opened/closed-events* as they are established successfully after routing. However, since Æthereal can switch the monitoring of specific events on and off, a more direct comparison is given by limiting the Æthereal monitoring to only *alert-events*. The results show that *ROAdNoC* generates less traffic even with the simple profile of the Æthereal monitoring. In conclusion, both monitoring components are

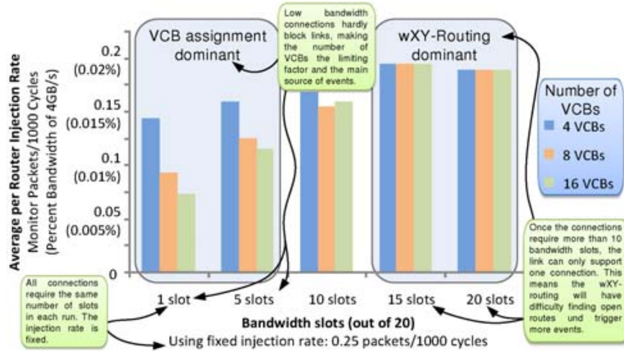

Fig. 6. Causes of monitoring events.

Successful connections	Monitoring traffic **(ROAdNoC)**	Æthereal (Alert & Config.-events)	Æthereal (Alert-events) only)
50	1.2KB/s	6.6KB/s	1.8KB/s
100	0.8KB/s	6KB/s	1.2KB/s
150	0.4KB/s	5.4KB/s	0.6KB/s
200	0KB/s	4.8KB/s	0KB/s

Table 2: Traffic comparison using 200 connections/s

designed with entirely different goals in mind. Our *ROAdNoC* infrastructure is designed specifically to facilitate the adaptivity of the NoC and thus only monitors events required to control the NoC configuration.

Fig. 7 shows the effect of re-sending packets on the number of packets which are able to be successfully transmitted for the E3S benchmark suite [6]. On an average, a re-sending threshold of 1 is able to increase the success rate by 62% compared to the case without runtime observability. For higher threshold values this value increases even further, allowing our infrastructure to avoid a costly (re-)mapping.

VII. Conclusion

We have introduced our approach of an infrastructure that provides runtime observability for an adaptive network on chip architecture. It is hardly intrusive, i.e. in worst case it may require a mere 0.7% of the total link capacity. Besides the main

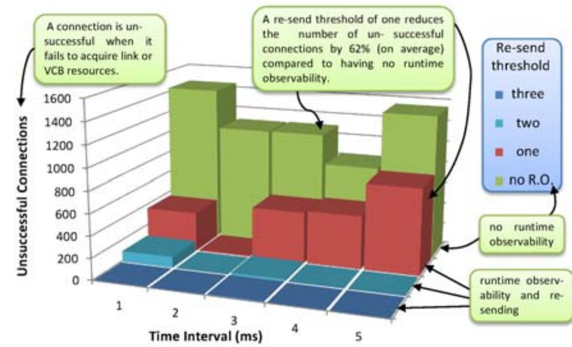

Fig. 7. Unsuccessful connection for various re-sending thresholds

objective of achieving flexibility in the communication architecture for higher resource utilization the hardware overhead at architecture-level due to runtime observation is rather small (46 slices). As a result, ROAdNoC increases the connection success rate by 62% in average compared to state-of-the-art approaches. It is currently the first prototype of its kind that can efficiently cope with hard-to-predict system behavior as a result of constraints that may change during runtime, reliability issues etc.

References

[1] L. Benini and G. D. Micheli. "Networks on Chips: a new SoC paradigm". *Computer*, 35(1):70–78, 2002.

[2] C. Bobda and A. Ahmadinia. "Dynamic interconnection of reconfigurable modules on reconfigurable devices". *IEEE Des. Test*, 22(5):443–451, 2005.

[3] S. Borkar. "Thousand core chips: a technology perspective". *DAC'07: Proc. of the 44th Conf. on Design Automation*, pages 746–749, 2007.

[4] C. Ciordas, T. Basten, A. Rădulescu, K. Goossens, and J. V. Meerbergen. "An event-based monitoring service for networks on chip". *ACM Trans. Des. Autom. Electron. Syst.*, 10(4):702–723, 2005.

[5] W. J. Dally and B. Towles. "Route packets, not wires: on-chip interconnection networks". *DAC'01: Proc. of the 38th Conf. on Design Automation*, pages 684–689, 2001.

[6] E3S. http://ziyang.eecs.northwestern.edu/ dickrp/e3s/.

[7] M. A. A. Faruque, T. Ebi, and J. Henkel. "Run-time adaptive on-chip communication scheme". *ICCAD'07: Proc. of the 2007 IEEE/ACM Int. Conf. on Computer-aided design*, pages 26–31, 2007.

[8] M. A. A. Faruque, R. Krist, and J. Henkel. "ADAM: run-time agent-based distributed application mapping for on-chip communication". *DAC'08: Proc. of the 45th Conf. on Design Automation*, pages 760–765, 2008.

[9] R. Ho, K. Mai, and M. Horowitz. "The future of wires". *Proc. of the IEEE*, pages 490–504, 2001.

[10] P. Horn. "Autonomic computing: IBM's perspective on the state of information technology". *IBM Corporation*, 2001.

[11] Y. Hoskote, S. Vangal, A. Singh, and S. Borkar. "A 5-GHz mesh interconnect for a teraflops processor". *IEEE Micro*, 27(5):51–61, 2007.

[12] W. Karl, M. Leberecht, and M. Oberhuber. "SCI monitoring hardware and software: Supporting performance evaluation and debugging". *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters*, pages 417–432, 1999.

[13] V. Nollet, T. Marescaux, D. Verkest, J.-Y. Mignolet, and S. Vernalde. "Operating-system controlled network on chip". *DAC'04: Proc. of the 41th Conf. on Design Automation*, pages 256–259, 2004.

[14] U. Y. Ogras and R. Marculescu. "Application-specific network-on-chip architecture customization via long-range link insertion". *ICCAD'05: Proc. of the 2005 IEEE/ACM Int. Conf. on Computer-aided design*, pages 246–253, 2005.

[15] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, 2007.

[16] V. Soteriou and L.-S. Peh. "Design-Space exploration of power-aware on/off interconnection networks". *ICCD'04: Proc. of the IEEE Int. Conf. on Computer Design (ICCD'04)*, pages 510–517, 2004.

[17] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten. "Congestion-controlled best-effort communication for networks-on-chip". *DATE '07: Proc. of the Conf. on Design, Automation and Test in Europe*, pages 948–953, 2007.

[18] S. Vassiliadis and I. Sourdis. "FLUX networks: Interconnects on demand". *Proc. of the Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 160–167, 2006.

[19] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, and A. Agarwal. "On-Chip interconnection architecture of the tile processor". *IEEE Micro*, 27(5):15–31, 2007.

[20] Xilinx. "Virtex2 complete datasheets". *http://www.xilinx.com/*.