Efficient Latency Guarantees for Mixed-criticality Networks-on-Chip

Sebastian Tobuschat and Rolf Ernst
Technische Universität Braunschweig
38106 Braunschweig, Germany
{tobuschat | ernst}@ida.ing.tu-bs.de

***Abstract*—Networks-on-Chip (NoCs) for future mixed-criticality systems must handle a growing variety of traffic requirements, ranging from safety-critical real-time traffic to bursty latency-sensitive best-effort traffic. Additionally, safety standards (e.g. ISO 26262) require sufficient independence among different criticality levels or a full system certification according to the highest applicable safety level. Hence, a NoC must provide performance isolation for safety-critical traffic, while sustaining low latency for best-effort traffic. This paper presents a run-time configurable NoC design enabling latency guarantees for safety-critical traffic with reduced adverse impact on the performance of best-effort traffic. In contrast to existing approaches, we prioritize best-effort over safety-critical traffic and only switch priorities when required. Doing this, we exploit the latency slack of safety-critical applications, while providing sufficient independence among different criticality levels w.r.t. timing properties. We present a formal analysis and an experimental evaluation, showing that the approach provides performance isolation for safety-critical applications, while reducing the adverse effects through strict prioritization on best-effort applications.**

I. INTRODUCTION

Multicore systems become increasingly interesting for safety-critical domains due to their performance, power, and size benefits. The use of multiprocessor system on chips (MPSoCs) allows integration and concurrent execution of multiple functions, which previously had been distributed and isolated by external buses. Consequently, MPSoCs must frequently accommodate heterogeneous workloads with different timing and safety requirements, forming mixed-criticality multicore systems [1], [2]. The challenge in integrating mixed-critical applications comes from the multicore inherent architecture. Shared resources, such as the communication infrastructure or memories, couple the execution behavior across cores. This impacts non-functional system properties like timing, which are of particular interest in safety-critical environments. Safety standards explicitly address this problem and require *sufficient independence* between functions of different criticalities (e.g. IEC 61508 [3]).

Networks-on-Chip (NoC), as a scalable and modular interconnect, are used as a promising solution for MPSoCs, due to their performance, power, and size benefits [4]. In a NoC resources, such as output ports in routers, are typically shared among different functions and safety-classes. Hence, applications of different safety levels will inevitably compete for resources in a NoC. One approach to tackle this problem is to develop all functions to the highest relevant safety level. However, this leads to higher development costs and lower system utilization. Therefore, it is crucial to provide methods guaranteeing *sufficient independence* in a NoC.

978-1-5090-5269-1/17/$31.00 ©2017 IEEE

Most of today's NoC designs achieve the independence through static (over) provisioning (e.g. TDMA) or static prioritization of safety critical traffic. This typically leads to a degradation of the performance, especially for non-safety critical applications. But for most safety-critical applications, it is sufficient to arrive shortly before or by their deadline and an early arrival provides no benefits [5]. On the other hand, a low latency for best-effort (BE) traffic can drastically increase the performance of BE applications [6], [7].

Contribution: In this paper we introduce a run-time configurable NoC design enabling latency guarantees for critical senders with reduced impact on the performance of best-effort traffic. We prioritize best-effort over guaranteed-latency traffic in NoC routers and only switch priorities when required, based on the actual blocking. For this we derive the allowed additional blocking time (slack) of a stream through timing analysis and store it in the packet header. This value is then adapted and evaluated in the routers to monitor the remaining slack time. This allows exploiting the latency slack of critical applications, while providing sufficient independence among different criticality levels w.r.t. timing properties. We present a formal analysis and an experimental evaluation, showing that the approach provides sufficient isolation for safety-critical applications, while reducing the adverse effects through the common used strict prioritization [8], [9] on best-effort applications.

The rest of the paper is structured as follows. After an overview of related work in Section II, we provide the description of our approach in Section III. In Section IV we introduce the analysis framework, which we then use in Section V to validate the sufficient isolation by providing latency bounds for guaranteed latency traffic. This is followed by an experimental evaluation in Section VI and the conclusion in Section VII.

II. RELATED WORK

There exist various packet-switched networks-on-chip providing quality-of-service (QoS) for mixed criticality systems that can be categorized by how they enforce service guarantees. One group uses time-division multiple-access (TDMA) to limit the interference between applications, e.g., [10]–[12]. These rely on a pre-allocation of time-slots in the network. Through assigning different safety classes to different time-slots, they provide strong isolation. The Nostrum [10] architecture uses *looped containers* that are continuously routed through the network. Applications are statically mapped to certain containers to isolate them. Aetheral [11] defines schedule tables in each node and router. These are used to define a static schedule for the whole communication in the NoC. The DPSIN [13] network combines rate control and TDMA. For providing guaranteed service (GS), a virtual channel (VC) in every switch is reserved for GS traffic as a TDMA channel. In this TDMA channel, guaranteed service is obtained by allocating time slots to the streams.

The fixed allocation of time-slots often leads to inflexibility and a high static overhead, as each packet might wait for its slot at multiple network nodes, even when other slots are empty. This inflexibility makes these architectures not suitable for high performance mixed-criticality systems. PhaseNoC [12] tackles this problem by defining scheduling in form of *waves* through the network. Through aligning the time-slots of all routers on a path, packets can travel with a reduced latency through the NoC. Another approach is the concept of channel-trees [14]. In this scheme, time slots are shared between a selected set of applications, enabling to utilize otherwise unused time slots. However, as these approaches rely on static timing schedules and slot assignments, they still introduce some inflexibility.

To relax the conservatism and timing overhead introduced by TDM, more flexible approaches were proposed. These are based on the idea of combining interface-based design and system analysis [15], [16]. Components and applications provide well defined interfaces and therefore introduce a bounded interference to the system. With this, more dynamic and work-conserving QoS mechanisms can be constructed based on rate controlling and dynamic scheduling as for example in [8], [17]–[19]. In [8] the authors propose the QNoC architecture. It uses four traffic classes and a fixed priority scheme in the switches to arbitrate between packets of the different classes. To isolate critical traffic from best-effort traffic, the critical traffic has a higher priority. In the Mango NoC [17] switches consist of two parts, a best-effort (BE) and a guaranteed service (GS) switch, and implement virtual channels. The GS streams are prioritized over BE streams and a fair-sharing arbitration is used between multiple GS streams. The latency of a message is bounded and mainly depends on the number of VCs sharing a particular connection and the selected arbitration policy. The Kilo-NoC [9] focuses on reducing the overhead of QoS mechanisms. While using a priority based QoS approach, it tries to reduce the overhead by a topology aware QoS design. Kilo-NoC only provides QoS mechanisms in the parts of the network where needed and uses simple routers for the remainder.

The authors of [18] present *WPMC*, a protocol for priority-preemptive VC arbitration, guaranteeing that all (critical) packets will arrive by their deadlines. It uses runtime monitoring at the network interfaces (NI) to check whether critical traffic stays within a predefined behavior (i.e. message sizes and inter-arrival time). If an injecting NI detects a deviation from this behavior, routers on the desired path switch to a critical state, in which all best-effort traffic is dropped by a router to favor the critical packets. This scheme is improved in [19] to not drop best-effort traffic but allow it to use idling ports of a router even in the critical state. These schemes are similar to our approach, as they allow prioritizing best-effort traffic over critical traffic, while monitoring is used to change the priority during run-time. The main difference is the monitored behavior. In [18], [19] the behavior of critical senders at the NI is monitored (i.e. message sizes and inter-arrival times), while our approach monitors the interference packets of critical senders experience in the network. Additionally, the monitoring at the injecting NI in [18], [19] only allows to differ between critical packets, that are transmitted either with low or with high priority on the whole path. And after switching to the critical state, the routers remain in this state and henceforth prioritize critical traffic. Hence, the exploitation of the slack of critical applications, to increase the performance of best-effort traffic, is limited in the current design of [18], [19]. However, as these approaches monitor a different part of the NoC, they can be combined with ours to further exploit a system through monitoring the behavior of critical tasks as well as the interference induced by non-critical ones.

In summary, most of today's NoC architectures do not meet the requirements on isolation, flexibility, and high system utilization at the same time. TDM based architectures introduce static overhead due to the static time schedule, reducing the performance in most cases. Most of the dynamic QoS approaches favor safety-critical over best-effort (BE) traffic (e.g. strict prioritization), thus reducing the BE performance. However, as the behavior of the safety-critical applications is well known and they do not benefit from finishing earlier than their deadline [5], this degradation of BE performance is typically unnecessary. To tackle these problems, our approach exploits the slack of safety-critical applications, thus increasing the BE performance compared to other approaches, while still providing sufficient isolation.

III. PROPOSED ARCHITECTURE

In this section we describe the new approach providing latency guarantees while reducing the adverse effects of state-of-the-art QoS mechanisms on best-effort applications. The goal is to exploit the latency slack of critical applications to increase the performance for best-effort traffic. In this work, the slack denotes the time budget between the worst-case latency of a packet and its deadline at the destination, i.e., the time a packet can be additionally delayed without violating its deadline in the worst-case. Safety-critical applications do not benefit from finishing before its deadline and thus the slack can safely be used to schedule other traffic [5], [7].

Although the proposed mechanism is not specific to a certain network architecture, we restrict the explanations to a baseline architecture as used in [20]. We assume a mesh network, where every router is connected to up to four neighboring routers and one client (e.g. processing element or memory). The routers use wormhole flow control, i.e., buffer management is performed on equally sized flits, and have a four-stage pipeline. The inputs provide multiple separate virtual channels (VC) to prevent head-of-line blocking between certain traffic streams. To reduce the size of the crossbar, each input can only send a flit from a single VC at a time over the crossbar and the routers use a two-stage arbitration, while other implementations are possible.

A. Baseline Architecture

The baseline implementation of our approach divides the traffic in two different QoS classes, where each class uses a dedicated set of virtual channels and is assigned a distinct priority level: guaranteed latency (GL) and best effort (BE). We use the GL channels for safety-critical traffic, while best-effort only contains non-critical traffic. If distinct priorities between different safety-critical applications are needed, more GL channels with different priorities can be used, similar as for standard priority based NoCs.

To enable the exploitation of the latency slack, we extend the packet header (for GL) with an additional field, which holds the slack information: the blocking counter (*BC*). This value is decremented in each router, based on the actual blocking experienced by the packet, and evaluated to monitor the remaining slack time. In the baseline version, this counter can be on flit or packet level. Each time a GL packet is blocked by a higher priority stream, the counter is decremented by one. For a packet level counter this leads to some conservatism, as it is also decremented, if GL is only blocked by a partial packet (e.g. the tail flit of a packet). If the network only supports a single packet size for all streams and this size is only a few flits, the packet level counter can be used to reduce the induced overhead, as less bits are needed in the header. For networks with different packet sizes and long BE packets, it can be adverse for GL to be delayed by a full BE packet. Hence, the flit level counter enables a more fine granular capturing of the interference.

The use of the slack information in the header enables to freely distribute the allowed additional blocking over the network, allowing to account for dynamics in the network that typically leads to temporarily local traffic hot-spots. Additionally, this offers the ability to adapt the allowed blocking at the traffic source during runtime and use different values for applications sharing a VC or even packets of the same application. Thus, it also offers to integrate local monitoring information of the network interface [19], or global information [21] in the allowed blocking to handle uncertainties of the system load.

The initial value of the blocking counter is set in the network interface based on the results of a performance analysis. As the analysis used (cf. Section V) assumes that backpressure does not occur, we do the same in this work. This can be ensured through the use of source rate limiters or sufficiently sized buffers in the router. The parameters for these can be obtained using the performance analysis. To extend the approach to a network with backpressure, a backpressure aware analysis [22], [23] must be used. Additionally, the routers must provide a mechanism to forward a priority signal if the slack is depleted but the downstream router has no free buffer space available. Otherwise the backpressure of the downstream router would repeal the local priority signal obtained from the blocking counter.

B. Dynamic Prioritization

The arbitration logic in the routers selects the next request (i.e. VC) based on the priority of the VC. In our approach, the priority relation of the BE and GL class can be adapted dynamically by the routers. We distinguish for each GL channel two different states in each router. In the *normal* state, the BE class has a higher priority than the GL channel. In the *critical* state, the GL channel obtains a higher priority than the BE class (and other GL channels in the normal state). With these priority levels and state definitions, the blocking counter (slack) of a GL packet in the *normal* state is decremented, each time the packet is blocked by a BE or a GL stream in the *critical* state. To determine the state of the VCs, a monitor checks the remaining slack time of the GL packets. If there is any GL packet in a queue which has no more slack available, the state of the queue is set to *critical* until only packets with slack available remain. Even though this scheme has the drawback of prioritizing flits, that still allow more blocking (i.e. another packet at the queue head might still allow blocking, while the last packet requests for prioritization), it allows an easy implementation with a low hardware overhead. Other possibilities are to move the packets that reached the maximum blocking to another queue (bypass) or to order the whole queue according to the remaining allowed blocking. This leads to a tradeoff between increased BE performance and induced overhead when deciding whether a bypass, reordering mechanism or simple queue prioritization should be used.

The arbitration of streams is packet based to increase the average performance of the system. Packet based arbitration also guarantees that the head and body flits of a packet experience the same additional blocking (e.g. the body flits can not experience additional blocking through BE after the header has been transmitted). Hence the blocking counter must only be stored in the header flit. To prevent additional priority inversion after the slack is depleted, GL packets can preempt BE packets on flit level when being in the *critical* state.

C. Arbitration Logic

This section introduces an exemplary extension of the arbitration stage in the routers to implement our mechanism. In the arbitration stage, requests from all virtual channels and input ports must be processed along with their traffic class and a priority signal, denoting if the slack of a packet is used up. As we change the priority of the GL traffic according to the actual blocking in the network, we need an arbiter that can switch between different priorities for certain requests. A solution for multi-priority arbiter is presented in [24].

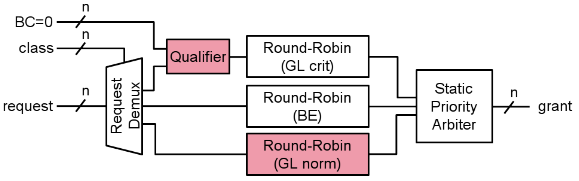

Fig. 1. Simplified output port arbitration logic

Figure 1 shows a simplified diagram of our arbitration stage of an output port. It consists of a request multiplexer that assigns incoming *requests* from n input ports to the different class arbiters based on their *class* identifier. In the baseline implementation, all classes use a round-robin arbiter, while other arbitration policies are also possible. For the guaranteed latency class, we use two arbiters. One for the normal state (*GL norm*) and one for the critical state (*GL crit*). The two GL arbiters are needed to individually keep track of the most recent sender on both priority levels, while other implementations, such as a single arbiter using two different states, are possible. If a GL request is raised, the request multiplexer sends it to the corresponding GL arbiters. A small qualifier logic then only allows requests that reached maximum blocking (*BC=0*) to be assigned to the arbiter of critical GL requests. Next to the individual class arbiters, we have a strict priority arbiter, sending out the highest priority grant to the requesting input ports.

Comparing the design with the arbitration logic of basic routers, only small changes are needed. A simple router, using two different priorities for the virtual channels and round-robin arbitration between requests of the same priority, requires all blocks but the *qualifier* and the additional (low priority) GL arbiter (state) [24]. Hence, our approach only extends an additional round-robin arbiter (or state) and the qualifier logic, which is mainly composed of *and*-gates forwarding only requests for which the slack is used up.

IV. COMPOSITIONAL PERFORMANCE ANALYSIS

This section provides an overview on the *compositional performance analysis (CPA)* [25] used for the analysis. Note that any other NoC analysis can be used too, if it allows the extension for the additional allowed blocking of our mechanism. The CPA approach uses a similar composition and event models as *Real-Time Calculus (RTC)* [26], but differs in the local analysis for the links and routers. For this, the *network-on-chip (NoC)* domain is translated to the processor resource model known from real-time scheduling [27].

CPA uses a multicore processor model to represent the NoC [20]. In the model, *processing resources* represent the output ports of a router and *shared resources* with mutually exclusive access the input ports. The exclusive access models the limitation of an input port to send only one flit at a time to an output port. A traffic stream is modeled as a *chain of tasks* mapped to the resources according to its network path. An exemplary mapping of a router with four streams is shown in Fig. 2. In the example, streams 2 and 3, represented by tasks τ_2 and τ_3, share the same input port and thus access the same shared resource InS. Stream 3 additionally shares the same output port with stream 4.

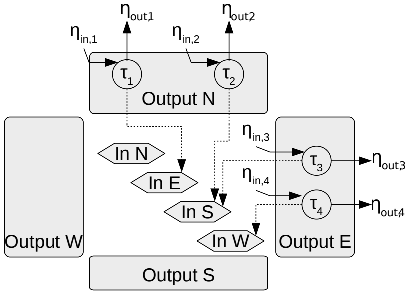

Fig. 2. Four-port router with four traffic streams as a multiprocessor with four processing resources (*Out*) and mutually exclusive shared resources (*In*)

In CPA, the arrival of a flit is a task activation at the processing resource and the transmission of a flit at an output port is the execution of that task. Each task τ_i is assigned a best- and worst-case execution time C_i^- and C_i^+ for each task activation. The activations of a task can be triggered by an external source (network interface) or other tasks (routers). Activation events are modeled by minimum and maximum arrival curves $\eta_i^-(\Delta t)$ and $\eta_i^+(\Delta t)$, defining the minimum and maximum number of events for task τ_i that can arrive within any half-open time window $[t, t+\Delta t)$. These functions have pseudo-inverse counterparts, the so-called distance functions $\delta^+(n)$ and $\delta^-(n)$, which define respectively the maximum and minimum time interval between the first and the last event of any sequence of n consecutive event arrivals.

System-level analysis is performed iteratively using individual resource-level analysis steps to obtain the worst-case timing information. For this, CPA performs a local *busy window* analysis for each resource to obtain worst-case timings and output event models for each task. The local resource-level analysis uses a *critical instant* scenario, assuming the worst-case arrival of all interfering tasks to obtain the maximum delay for the task under analysis. The output event models from the local analysis are then forwarded as input models for all dependent tasks and resources, which are analyzed again if their input models have changed. The local analysis and propagation are iteratively applied until all output event models remain stable [25].

V. WORST-CASE ANALYSIS

In this section we present an analysis for the upper latency of GL traffic using CPA for our mechanism, proving sufficient independence. To improve readability, we restrict the analysis to the case where all GL streams share a single VC. However, it can easily be extended to handle multiple virtual channels and additional priority levels [20]. For the analysis according to CPA, we need to derive the corresponding worst-case multiple activation processing time $B_i^+(q, a_i^q)$ of a stream in a router. It denotes the maximum time required to transfer q flits of a stream i, given that all but the first flit arrive before their respective predecessor has been transferred and the q-th flit arrives at time instant a_i^q. Based on this, we then derive metrics (e.g. path latency) for a single router and for a complete network.

To conservatively capture all possible worst-case scenarios, we break down the multiple activation processing time into a sum of different terms addressing different blocking factors. For a router as described in Section III the processing time is influenced by:

- **Flit transfer time C**: the time to transfer a flit in a router excluding any kind of blocking. For the sake of simplicity and since it is the usual case in NoCs, we consider that all flits have the same constant transfer time of 1 cycle.
- **Output blocking B_i^{out}**: the amount of time that stream i is blocked by any stream j from other inputs that use the same output.
- **Input blocking B_i^{in}**: the amount of time that stream i is blocked by any stream j that use the same input and precedes the q-th flit of stream i.
- **Low priority blocking $B_{i,q}^{LP}$**: the (remaining) amount of time stream i is blocked being in the normal state.

To upper bound the longest time $B_i^+(q,a_i^q)$ required to transfer q flits of stream i at a router, we maximize all blocking effects and sum them up:

$$\begin{aligned} B_i^+(q,a_i^q) &\le q\cdot C + B_{i,q}^{LP}(B_i^+(q,a_i^q)-C) \\ &+ B_i^{out}\left(B_i^+(q,a_i^q)-C,q\right) + B_i^{in}\left(B_i^+(q,a_i^q)-C,a_i^q\right) \end{aligned} \quad (1)$$

For the worst-case, we assume full blocking while being in the normal state (B^{LP}), followed by the maximum blocking from other GL streams on the router ($B^{out}+B^{in}$). To derive the multiple activation processing time on a single router, we now derive the individual sources of blocking. For this we first define an auxiliary function:

Definition 1. *Let $\rho_i^+(\Delta t)$ denote the maximum number of flits arriving in any time interval Δt at a stream* i *considering whole packets:*

$$\rho_i^+(\Delta t) = \left\lceil \frac{\eta_i^+(\Delta t)}{size(i)} \right\rceil \cdot size(i), \quad (2)$$

where size(i) *is the packet size of stream* i *in flits.*

Theorem 1. *The output blocking B_i^{out} that a GL stream* i *observes in any time window consists of the blocking through other GL streams from other input ports. It can be bounded by:*

$$B_i^{out}(\Delta t,q) \le \sum_{p\in P(i)} C\cdot\min\left\{\left(\left\lceil\frac{q}{size(i)}\right\rceil+1\right)\cdot\hat{n}, \sum_{j\in Out_{GL}^{p}(i)} \rho_j^+(\Delta t)\right\} \quad (3)$$

where $\hat{n}$ is the maximum packet size of GL streams, P(i) *is the set of other input ports, and $Out_{GL}^p(i)$ denotes all GL streams on input* p *that share the same output port with stream* i.

Proof. As we already accounted for the full blocking being in the normal state in Equation 1 ($B_{i,q}^{LP}$), we only need to account for GL streams from other input ports. Due to wormhole switching, once the scheduler grants access to an output port, no other input port can access this port until the packet is fully transmitted. This is captured by ρ_j^+, which considers that after a head flit from j arrives within the time interval Δt, the whole packet will be served before i. Additionally, due to the round-robin arbitration, each head flit belonging to stream i may only be blocked once by each other input port. But as the state of the GL sender, and hence also the arbiter state, might have changed, an additional round-robin cycle can occur. This is addressed with the *min*-function, where $\left\lceil\frac{q}{size(i)}\right\rceil$ is an upper bound on the number of head flits. Each of these head flits can be blocked at most for $\hat{n}$ flits from each other input port. And each of the interfering flits then blocks stream i for the flit transfer time C. ☐

Theorem 2. *The input blocking B_i^{in} of a GL stream* i *in any time window consists of the blocking of other GL streams that share the same input port. It can be bounded by:*

$$B_i^{DI}(\Delta t,a_i^q) \le \sum_{j\in In_{GL}(i)} \left\{C\cdot\rho_j^+(a_i^q) + B_j^{out}\left(\Delta t - C, \rho_j^+(a_i^q)\right)\right\}, \quad (4)$$

where $In_{GL}(i)$ denotes the set of all GL streams that share the same input port with stream i *and a_i^q the arrival time of the* q*-th flit.*

Proof. The blocking caused by other streams at the same input consists of the transmission time of the flits that arrived before the q-th flit of stream i and the interference those flits observe. In the sum the first part accounts for all whole packets that can arrive before the q-th flit (ρ^+) and their transmission time ($C\cdot\rho^+$). The second part then accounts for the output blocking these flits can experience. However, we only need to account the flits that can arrive before the arrival time of the q-th flit of stream i (a_i^q) considering whole packets.

As i and j share the same interferer set at the input port, all input blocking of j is accounted for in the input blocking of i and must not be accounted here again. ☐

Theorem 3. *The blocking while being in the normal (i.e. low priority) state $B_{i,q}^{LP}$ of the* q*-th flit of a stream* i *can be derived from the blocking counter (BC) in the flit header. This LP blocking can be denoted as:*

$$B_{i,q}^{LP}(\Delta t) \le C\cdot\min\left\{\sum_{j\in hp(i)} \eta_j^+(\Delta t), \kappa_i(\Delta t)\right\}$$
$$\text{with} \quad \kappa_i = \begin{cases} BC_i^q\cdot\hat{n}, & \text{if BC counts packets} \\ BC_i^q, & \text{otherwise} \end{cases} \quad (5)$$

where BC_i^q denotes the current blocking counter value of the q*-th flit and $hp(i)$ the set of all interfering streams for stream* i *that might have a higher priority (i.e. BE and critical GL) at the router under analysis.*

Proof. From Section III we know, that the flit only is in the normal state as long as the blocking counter has slack available. Depending on the implemented granularity (packet or flit based counter) it is decremented for each packet or flit with an higher priority (BE or critical GL) that blocks stream i. When the counter reacher zero, no more blocking is allowed. Hence, only BC_i^q packets or flits of be can interfere with the q-th flit of i. Additionally, we only have to account for interfering flits that can arrive during the time the q-th flit is waiting at the router, covered by the min-function and the sum of arrivals of all interfering streams (η_j^+). □

With all sources of blocking defined, the multiple activation processing time from Equation 1 for a single router is fully defined. Based on this we now derive an upper limit for the worst-case single hop latency and then extend this to obtain end-to-end metrics for the whole network.

A. Derived Metrics

With the multiple activation processing time we can derive the worst-case single hop latency R_i^+ of a stream i. It denotes the maximum time between the arrival time a_i^q of the q-th activation and the processing of q activations $B_i^+(q,a_i^a)$ [28]:

$$R_i^+ = \max_{q \in Q_i} \left\{ \max_{a_i^q \in A_i^q} \left\{ B_i^+(q,a_i^a) - a_i^q \right\} + O_r \right\}, \tag{6}$$

where O_r denotes the static router overhead, such as the time a router requires to determine the output port and virtual channel.

This equation considers for all numbers of activations q within a busy-period all possible arriving times a_i^q of the q-th event. This is necessary, as a later arrival time might increase the interference in input queues. Additionally, a later arrival reduces the response time. However, in [28] the authors proved that the number of possible candidates for a_i^q is finite. The authors showed, that it is sufficient to consider only candidates that coincide with activations of the interfering workload. Hence, we can define the set of candidates as:

$$A_i^q = \bigcup_{j \in I(i)} \Bigg\{ \delta_j^-(k) \mid \delta_i^-(q) \leq \delta_j^-(k) < \delta_i^- \left(\left\lceil \frac{q}{size(i)} \right\rceil \cdot size(i) + 1 \right) \Bigg\}_{k \geq 1} \tag{7}$$

where $I(i)$ defines the set of all interfering streams for stream i using the same queue (i.e. virtual channel) on the same input channel in the router.

Additionally, we need to consider all scenarios where an activation arrives within the busy-period of the previous activation when defining the number of activations q. Thus, we have to find all $q \geq 1$ that are smaller than the maximum number of events q_i^+ forming one busy-period:

$$Q_i = \{1, 2, \ldots, q_i^+\} \qquad with$$
$$q_i^+ = \min \left\{ q \in \mathbb{N}^+ \mid \delta_i^-(q+1) \geq \max_{a_i^q \in A_i^q} \left\{ B_i^+(q,a_i^q) \right\} \right\}. \tag{8}$$

Based on the multiple activation processing time, we can also derive the worst-case backlog in each buffer. The worst-case backlog of a buffer b of a router can be defined as:

$$b_p = \sum_{i \in Buf} \left\{ \max_{q \in Q_i} \left\{ \max_{a_i^q \in A_i^q} \left\{ \eta_i^+(B_i^+(q,a_i^q)) - q + 1 \right\} \right\} \right\}, \tag{9}$$

where Buf denotes the set of streams sharing the buffer. This equation examines all numbers of activations q which arrive during the processing time of their predecessor. For each, $B_i^+(q,a_i^q)$ yields the completion time of the q-th activation. At this instance of time, at most $\eta_i^+(B_i^+(q,a_i^q))$ activations may have arrived since the start of the busy window, of which $q-1$ have already been processed. Hence, the difference yields the number of backlogged activations, of which we have to determine the maximum for all $q \in Q_i$ providing the worst-case backlog of a stream. The sum of the worst-case backlogs of all streams sharing a buffer then derives the worst-case buffer backlog. Note that the analysis assumes that backpressure does not occur, which can be validated during design phase using this (backlog) analysis.

B. Derived Metrics for the Network

With the analysis of a single router, we can analyze the whole network using the CPA approach. For this we iteratively perform a local analysis of the routers and propagate the event models to the subsequent routers. Based on the local analysis, we can define the output events models that become the input event models of the subsequent routers according to [29] as:

$$\delta_{i,out}^-(n) = \max \Big\{ (n-1) \cdot C, \min_{j \in Q_i} \{ \delta_{i,in}^-(n+j-1) - B_i^+(j) \} + C) \Big\}, \tag{10}$$

where $(n-1) \cdot C$ denotes the best-case execution time and the second term accounts for the influence of the busy-times on the model propagation.

Besides the event models, we need the maximum LP blocking that a GL packet allows in each router on its path (i.e. the current value of BC). A naive approach is to assume the initial value in all routers, which leads to pessimistic (but valid) results. The pessimism can be reduced, if we can assume a certain minimum and maximum load from the interfering workload, such that there is always some or no LP blocking for a stream i in the normal state and derive more accurate propagation models for possible values. If there is a minimum load through BE on a router, some of the LP blocking will always occur. And if there is an upper bound on the interfering load, not more than this can occur. These information can be used to reduce the set of possible combinations. For example, if we can assume the full initial blocking value for the first router on a path, we know, that there will be no additional LP blocking on the subsequent routers as the stream switched to the critical state. To optimize the analysis results, we have to analyze all possible distributions of the blocking counter on the path p of a stream i and derive the maximum end-to-end latency $l_p^+(q)$ for transmitting q flits from all distributions. It consists of the worst-case response time for each hop on the path p, the time to inject the q flits, and the packetization overhead:

$$l_p^+(q) \le \delta^-_{First(p)}(q) + O_p + \max_{d \in Dist} \left\{ \sum_{j \in Tasks(p)} R^+_{j,d} \right\}, \tag{11}$$

where $First(p)$ defines the first task of the chain (i.e. network path), $Tasks(p)$ the set of all tasks of path p (i.e. one per hop), O_p the constant de/packetization overhead, $R^+_{j,d}$ the worst-case single hop latency assuming a distribution d of the LP blocking, and $Dist$ the set of all possible distributions of the LP blocking for the stream p. Basically the equation computes how long it takes a stream to inject q flits and then assumes the last one of these to experience the worst-case blocking on all intermediate routers. Due to the in-order delivery of the network, all previous flits will have arrived at the destination before the last one. And the delay previous flits may observe is included as interference in the worst-case blocking of the last flit.

The analysis also provides a framework to find admissible values for the allowed LP blocking (i.e. initial blocking counter value) for a GL stream to exploit its slack. For this, we can define an iterative approach that compares the worst-case end-to-end latency against the deadline of a stream to derive its slack. First, we analyze the system, assuming all blocking counters to be zero (i.e. classic prioritization of GL) and the system to be schedulable. Then we identify the GL streams that have slack by comparing their derived worst-case latency with their deadline. If any stream has slack, we can increase the initial BC value of this stream and reanalyze the system to check whether all deadlines are still met. Doing this, we can find feasible initial values of LP blocking for all streams. Besides this simple approach, more complex strategies for defining initial BC values are possible. For example, the approach from [19] can be used to define BC values for different sender states based on the core- or NI-local behavior (e.g. message size and inter-arrival times).

VI. EVALUATION

In this section, we evaluate the performance of our mechanism and compare it to the classic and widely used prioritization scheme, as for example [8], [9]. We divide the evaluation in two parts. In the first part, we use synthetic workloads to evaluate the basic functioning and certain properties of our mechanism, such as isolation between BE and GL and the correctness of the analysis. In the second part, we use memory access and communication traces of general purpose applications, to investigate the performance of the mechanism on realistic workloads. All experiments were carried out with the *OMNeT++* simulation framework and the *HNOCS* library [30] using routers with a four-stage pipeline, buffers to store 6 packets in each virtual channel of each input port, and a packet size of four flits.

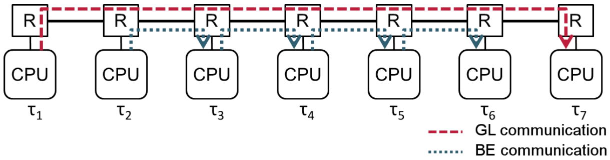

Fig. 3. Simple communication scenario

A. Synthetic Workloads

The first set of experiments uses synthetic workloads, generated based on average link loads, and the simple system shown in Figure 3. It comprises a simple line topology to maximize the overlap between traffic streams. In the example, we have five traffic streams, each periodically injecting packets with a jitter of 25% of the period. There is a guaranteed latency (GL) communication from task τ_1 on the first node to τ_7 on the last node. In between there are four best-effort (BE) communications, where the BE tasks send data to their direct neighbor. This example is compact enough to be comprehensively displayed but shows all relevant effects of the analysis. Especially, the synthetic workload allows to more easily approach the worst-case behavior than using an application trace.

In a first experiment, we set the GL task to generate an average network load of 10% and increase the load of the BE tasks (i.e. reduce the period between packets), to measure their interference on GL. For GL, we evaluate four different values for the allowed LP blocking (BC) on flit level granularity. Figure 4 shows the end-to-end latency of full packets for the GL sender in this experiment. In the figure solid lines show the measured maximum latency and dashed lines the results of the worst-case analysis for GL assuming maximum BE load. If we allow zero LP blocking (BC=0), GL traffic always has a higher priority than BE traffic, which corresponds to classic prioritization of GL. In this case, an increasing BE load does not influence GL traffic.

With the allowance of LP blocking (BC>0), BE can interfere with GL traffic, where a higher BC value leads to a higher possible interference. However, the interference is upper bounded and thus from a certain BE load onwards, the GL latency is not further increasing. In all cases, the observed upper bound is below the analysis results. We can also see, that for small loads, the observed latency for GL is similar for different BC values. This is due to the fact, that the possibility for GL and BE flits to compete in a router for resources depends on the load in the system. With a low load, the flits rarely compete in the routers, and hence the worst-case might not be observed in simulation. For high loads, however, the flits compete more often, leading to a higher interference and thus latency, especially for higher values of BC.

In a second experiment, we investigate the influence of GL on BE traffic. For this, we compare the classic prioritization of GL traffic against our approach for a BE load of 20%. Figure 5 shows the results of this experiment. For classic prioritization of GL (denoted as *HP*), the latency for best-effort tasks τ_2 and

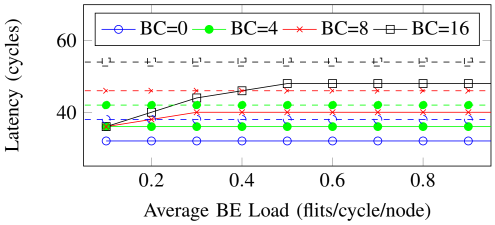

Fig. 4. Measured (*solid*) and analyzed (*dashed*) worst-case packet latency of GL for synthetic BE loads and various blockage values

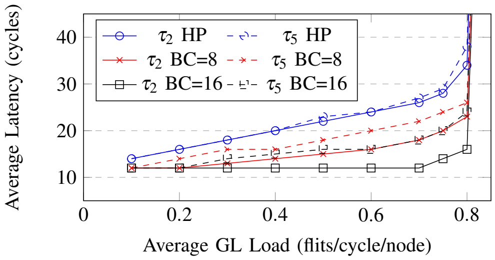

Fig. 5. Average BE latency for synthetic GL loads

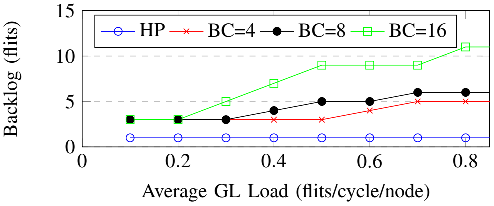

Fig. 6. Worst-case backlog of GL from analysis

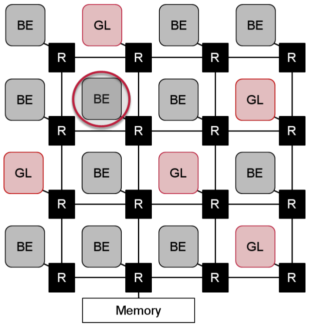

Fig. 7. Communication scenario for benchmark applications

τ_5 behaves the same. With an increasing GL load, the latency increases, due to the higher priority of GL. The latency of BE tasks thus depends significantly on the load of GL traffic. For the new approach, the experiment shows a better performance for BE tasks and loosens the dependency on GL load.

Additionally, we can see, that the value of the allowed LP blocking and the distance to the GL sender influence the performance benefit of BE tasks. Here the task τ_2 has a better performance than τ_5. This results from the fact, that for τ_2 the allowed LP blocking of the GL flits is still at its initial value. Thus, flits from τ_2 can pass the router with a higher priority than GL, even for increased loads. With an increasing distance, the remaining allowed LP blocking tends to be lower. Thus, the task τ_5 has a higher probability to be blocked by GL flits and to experience a higher latency.

With the allowance of blocking through BE, the GL packets also stay longer in the router, which can increase the backlog. Figure 6 shows the backlog derived from the analysis for the different configurations for the same experiment. As can be seen, the backlog depends on the GL load and BC value. Especially for high loads and high values of BC the backlog can increase drastically. To prevent this, rate limiters at the source can be used, which exists in many existing NoCs, or a backpressure aware analysis must be used (cf. Section III).

B. Benchmark Workloads

In the second set of experiments we use realistic workloads to evaluate the performance of the proposed mechanism. For the experiments we obtained traces from the CHStone benchmark suite [31]. The traces were extracted using the Gem5 simulator and an ARMv7-a core with a 32 kB L1 cache and contained 100.000 accesses to the network, where each access can be a direct memory access, communication or a cache access. The compilation was performed using standard *gcc* compiler (ver. 4.7.3). For a simulation run, the different traces were randomly assigned to nodes in a *4x4* mesh network as shown in Figure 7 to obtain different traffic distributions, while the nodes and the network were running with the same clock frequency. In the network, we assign fixed QoS classes to certain nodes. There are five nodes initiating GL traffic and eleven nodes for BE traffic. Additionally, there is one memory with a single interface to the network. For the routing policy we use an XY-routing. Thus, when regarding the highlighted BE node, all GL flits accessing the memory will compete with flits of this node. This setup is compact enough to be comprehensively displayed but shows all relevant effects of the mechanism. Based on this setup, we conduct two series of experiments. In the first series we use the memory as a hot-module. That is, all traffic generated by the nodes is sent to the memory, leading to higher interference on links near the memory. In the second series we use pseudo random destinations for the traffic of all nodes.

The results of the first experiment, with the memory as the hot-module, are presented in Figure 8. It shows a box plot of the measured latencies for eight different applications mapped to the highlighted node. For each run, the box covers 50% of the latencies, with its lower and upper borders giving the 25% and 75% quartiles, respectively. The whiskers indicate the measured minimum and maximum observed latency. The median and average among the measured latencies are respectively marked by a black bar and a black dot. For each of these applications, we generated 1000 different sets of interfering workloads for the other nodes. The results are presented for three different prioritization approaches of the GL nodes: the classic prioritization (*HP)* and the new approach

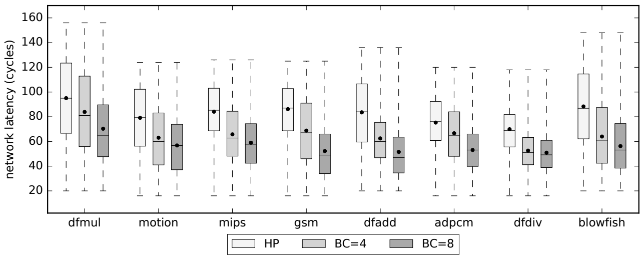

Fig. 8. Performance of CHStone benchmarks as BE traffic with memory as hot-module

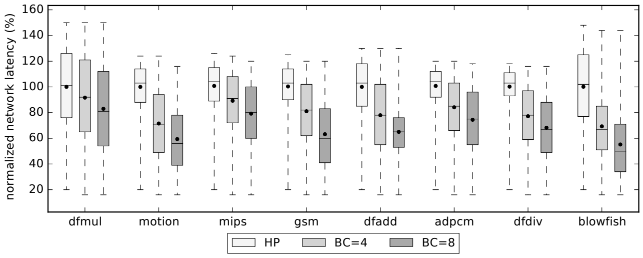

Fig. 9. Performance of CHStone benchmarks as BE traffic for random destinations normalized to average latency at classic prioritization

with two different values for the allowed LP blocking on flit level (*BC=4*) and *BC=8*).

As we can see, the new approach reduces the average latency for both configurations compared to classic prioritization by up to 36% for a blockage value of four and up to 43% for eight. The minimum and maximum latencies are similar for all schemes. For a single hot-module, the different GL flits experience high interference on the way to the hot-module. Hence, it is likely that the maximum allowed blocking can be consumed, leading to a worst-case blocking for BE. On the other hand, the dynamic behavior of the applications also leads to cases, where flits pass the network with nearly no interference, leading to the best-case latency. In this experiment, our approach increased the backlog of GL from 16 flits using classic prioritization to up to 20 flits with $BC = 8$. Compared to the synthetic workloads, the backlog of GL is higher even for classic prioritization, as multiple GL senders now share a virtual channel.

Figure 9 shows the normalized latencies as box plots for the system using pseudo random destinations for the traffic. We generated for each listed benchmark 1000 different sets of interfering workloads (i.e. random assignment of application to nodes) and a random destination for each sender. We selected the destinations such that a traffic stream has to pass at least three routers (i.e. no traffic to direct neighbors). For the figure, we normalized for each run the latency to the average latency when using classic prioritization for GL (*HP*).

As we can see, our approach leads to a performance increase for the BE applications compared to classic prioritization of GL traffic. If we allow four blockages through the best-effort class, we can reduce the average BE latency by up to 30%. And when we increase the allowed blockage to eight, we can reach a latency reduction by up to 45%. However, the achievable reduction of the latency depends on the application behavior. At the same time, the approach also increases the variability in the occurring latencies and backlog. With an allowed blockage of $BC = 8$ the backlog increases from 11 to 14 flits compared to classic prioritization. This increase is smaller as for the case of the hot module, as with random traffic less GL senders compete for the same buffer.

C. Synthesis Results

In this section we briefly present synthesis results for our approach. For this we implemented and synthesized a *2x2* NoC on a *Virtex-6 LX760 FPGA* using *Xilinx ISE 14.6* with default optimization setting and no special optimizations for the *VHDL* implementation. The device utilization data were collected from the *Module Level Utilization Summary Report* produced by ISE. Note that, as this NoC is not fully connected, each of the four routers has only three input ports fully instantiated. The results for the whole NoC are summarized in Table I. The table compares the used *registers*, *LUTs*, and achievable clock frequency for four different implementations. The *baseline* implementation corresponds to a basic round-robin router with 5 virtual channels (VCs) and a buffer depth of 6 packets for each VC. This was extended in *FP* to provide one prioritized VC, e.g. *VC0*. In the *DP* design the priority of *VC0* can be changed via a configuration flag from the highest to the lowest priority during run-time. And finally, the *BC* implementation denotes our approach, where the priority of *VC0* is dynamically changed by the router based on the current blocking counter value in the flit header

TABLE I
SYNTHESIS RESULTS ON VIRTEX-6 LX760 FPGA

Unit	Baseline	FP	DP	BC
#Registers	9365	9395	9389	9740
#LUTs	12149	12205	12199	12688
Frequency (MHz)	210	210	210	210

We used 4 bits to store the blocking counter in the header that were previously unused. Note that if no spare bits are available in the header, the header, and possibly the signal width between the routers, must also be extended, resulting in additional overhead. The synthesis shows, that the new approach introduces less than 5% overhead for the used *2x2* NoC. This corresponds to an approximately increase of 1.67% of a router per instantiated port. However note that, if no source rate limiters are used, our approach can increase the worst-case backlog (e.g. from 11 to 14 flits in the benchmark example) and hence might require bigger input buffers (if no backpressure aware analysis is used) and further increasing the overhead. The achievable clock frequency was the same for all designs, showing that the extensions did not influence the critical timing path. The frequency was restricted by the minimum achievable period, caused by a *data path delay* of *4,75ns*, consisting of *1,31ns* for logic and *3,44ns* route delay.

VII. CONCLUSION

In this paper we presented a novel arbitration scheme for NoC routers in mixed-criticality systems. Unlike many other existing approaches, we prioritize best-effort over safety-critical guaranteed-latency traffic whenever possible. To limit the interference, we online monitor the blocking experienced by the safety-critical traffic and based on this increase its priority only when necessary. Doing this, we can exploit the latency slack of critical applications. This leads to an improved performance for general purpose functions in mixed-criticality systems.

Our experimental evaluation showed that the approach can improve the latency of best-effort traffic by up to 45% compared to a standard prioritization scheme when the safety-critical traffic provides sufficient latency slack. At the same time, the approach provides latency-guarantees to safety-critical real-time functions and thus sufficient isolation as requested by safety standards. However, the approach also increases the hardware overhead of the network routers by up to 5% for a network with four routers, leading to a trade-off between performance and complexity.

REFERENCES

[1] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pp. 13–22, 2010.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review (7-th ed)," *Department of Computer Science, University of York, Tech. Rep*, January 2016.

[3] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety related systems," 1999.

[4] L. Benini and G. D. Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, pp. 70–78, Jan 2002.

[5] J. A. Stankovic, K. Ramamritham, and M. Spuri, *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[6] N. Muralimanohar and R. Balasubramonian, "Interconnect design considerations for large nuca caches," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, (New York, NY, USA), pp. 369–380, ACM, 2007.

[7] S. Tobuschat, M. Neukirchner, L. Ecco, and R. Ernst, "Workload-aware shaping of shared resource accesses in mixed-criticality systems," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on*, pp. 1–10, Oct 2014.

[8] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Qnoc: Qos architecture and design process for network on chip," *J. Syst. Archit.*, vol. 50, pp. 105–128, Feb. 2004.

[9] B. Grot, J. Hestness, S. Keckler, and O. Mutlu, "Kilo-noc: A heterogeneous network-on-chip architecture for scalability and service guarantees," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 401–412, June 2011.

[10] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, pp. 890–895 Vol.2, Feb 2004.

[11] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proceedings of the 47th Design Automation Conference*, DAC '10, (New York, NY, USA), pp. 306–311, ACM, 2010.

[12] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "Phasenoc: Tdm scheduling at the virtual-channel level for efficient network traffic isolation," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, (San Jose, CA, USA), pp. 1090–1095, EDA Consortium, 2015.

[13] I. Miro Panades, A. Greiner, and A. Sheibanyrad, "A low cost network-on-chip with guaranteed service well suited to the gals approach," in *Nano-Networks and Workshops, 2006. NanoNet '06. 1st International Conference on*, pp. 1–5, Sept 2006.

[14] A. Hansson, M. Coenen, and K. Goossens, "Channel trees: Reducing latency by sharing time slots in time-multiplexed networks on chip," in *CODES+ISSS*, pp. 149–154, Sept 2007.

[15] E. Wandeler and L. Thiele, "Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling," in *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, (New York, NY, USA), pp. 80–89, ACM, 2005.

[16] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pp. 34–43, Nov 2011.

[17] T. Bjerregaard and J. Sparsoe, "Scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip," in *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pp. 34–43, March 2005.

[18] A. Burns, J. Harbin, and L. Indrusiak, "A wormhole NoC protocol for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pp. 184–195, Dec. 2014.

[19] L. Indrusiak, J. Harbin, and A. Burns, "Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip," in *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pp. 47–56, July 2015.

[20] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, (San Jose, CA, USA), pp. 537–542, EDA Consortium, 2015.

[21] A. Kostrzewa, S. Saidi, L. Ecco, and R. Ernst, "Dynamic admission control for real-time networks-on-chips," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 719–724, Jan. 2016.

[22] L. S. Indrusiak, A. Burns, and B. Nikolic, "Analysis of buffering effects on hard real-time priority-preemptive wormhole networks," *CoRR*, vol. abs/1606.02942, 2016.

[23] S. Tobuschat and R. Ernst, "Real-time communication analysis for networks-on-chip with backpressure," in *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition*, DATE '17, (San Jose, CA, USA), EDA Consortium, to be published.

[24] D. U. Becker, "Efficient microarchitecture for network-on-chip routers," *PhD dissertation, Stanford University, Dept. of Electrical Engineering*, pp. 11–27, 2012.

[25] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques, IEE Proceedings -*, vol. 152, pp. 148–166, Mar. 2005.

[26] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 4, pp. 101–1044, 2000.

[27] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Syst.*, vol. 6, pp. 133–151, Mar. 1994.

[28] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of ethernet topologies with strict-priority and avb switching," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pp. 1–10, June 2012.

[29] S. Schliecker, J. Rox, M. Ivers, and R. Ernst, "Providing accurate event models for the analysis of heterogeneous multiprocessor systems," in *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '08, (New York, NY, USA), pp. 185–190, ACM, 2008.

[30] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny, "Hnocs: Modular open-source simulator for heterogeneous nocs," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pp. 51–57, July 2012.

[31] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis," *Journal of Information Processing.*, vol. 17, pp. 242–254, 2009.