Slack-Based Resource Arbitration for Real-Time Networks-On-Chip

Adam Kostrzewa, Selma Saidi, Rolf Ernst
Institute of Computer and Network Engineering
Technische Universität Braunschweig
{kostrzewa, saidi, ernst}@ida.ing.tu-bs.de

Abstract— Networks-on-Chip (NoCs) designed for real-time systems must efficiently deal with a broad diversity of traffic requirements. This requires providing latency guarantees for hard real-time transmissions with minimum impact on performance sensitive best-effort traffic. In this work, we present a novel mechanism which achieves this goal through a slack-based global and dynamic prioritization of data streams. This is performed using an overlay network and a scheduling unit combining local arbitration performed in routers with global scheduling of entire logical transmissions for end to end guarantees. Consequently, our approach allows to decrease both hardware and temporal overhead when compared with existing solutions and to achieve a performance improvement up to around 60%.

I. Introduction

Multiprocessor systems-on-chip (MPSoCs) allow the construction of highly complex, multifunctional consumer and mobile devices through the integration and concurrent execution of previously distributed functionalities. In such systems, when applied to safety critical real-time domains, the underlying interconnect infrastructure must efficiently accommodate running applications that have usually different time-related requirements.

For instance, transmissions conducted by applications with hard-real time constraints are not allowed to miss any deadline as it endangers user's safety or causes prohibitive reduction of the service quality. Consequently, safe accommodation of hard real-time transmissions (HRTTs) requires worst-case dimensioning and timing analysis during the design process. However, there is usually no advantage in completing an HRTT earlier than required since safety standards insist on the satisfaction of all timing constraints even if unnecessarily stringent i.e. as long as an application completes by its deadline, its response time is not important. On the contrary, transmissions from soft-real time and best effort applications (SRTTs) are rarely required to rigorously meet their deadlines but at the same time they must still comply to the overall real-time performance objectives e.g. low average latencies. Therefore, the integration of HRTTs and SRTTs in the same chip requires "sufficient isolation" between components with different criticality levels (e.g. safety standard DO-178B). However, this often compromises the performance of SRTTs.

Slack-based resource allocation is a well known solution from the scheduling theory [1], [2] to this integration challenge, providing high performance for soft-real time applications without violating the timing constraints of hard real-time applications. According to this approach, SRTTs are scheduled whenever the execution of HRTTs can be safely postponed without causing missed deadlines. This is possible whenever a slack is available, which is the time budget between the worst-case response time of a hard-real time application and its deadline.

Applying this principle to Networks-on-Chip (NoC) significantly increases the performance of soft real-time and best effort data streams which is particularly useful for general-purpose latency sensitive applications running on processors with caches. Only few existing work e.g. [3] have considered using slack-based resource allocation in the context of NoCs by applying, locally in routers, dynamic prioritization for different virtual channels. However, the main drawback is the high hardware overhead resulting from a custom router design and the high number of virtual channels (i.e. required buffers) corresponding to the number of priority levels in the system. Moreover, these solutions drastically increase NoC's complexity as they require to propagate the global state of the NoC to the local arbiters in routers. This is incompatible with the principle of non-blocking routers which requires no correlation between local arbiters and thereby increases the complexity of the worst-case timing analysis.

In this work, we confront this challenge and propose an alternative mechanism for providing efficient and safe sharing of NoC resources between soft and hard real time applications. This is performed using an *access layer* for a global and dynamic admission control, implemented within the existing NoC architecture. Interfering HRTTs and SRTTs are synchronized using a scheduling unit called the Resource Manager (RM), which uses dynamic priority arbitration to give SRTTs access to the NoCs whenever there is an available slack. In order to meet the HRTTs timing constraints, we compute using formal timing analysis a *safe* upper bound on the slack budget that can be used to postpone HRTTs without endangering their deadlines. Consequently, the proposed apporach significantly simplifies the efficient integration of HRTTs and SRTTs while reducing both temporal and hardware overhead resulting from the isolation of transmissions. Note that the mechanism does not require the modification of routers. Therefore, it can be used together with any NoC architecture utilizing the principle of non-blocking routers, hence improving the utilization of many existing NoCs.

II. Related Work

Existing approaches for supporting QoS in real-time NoCs can be generally divided into two categories: Time-Division Multiplexing (TDM) and non-blocking routers with rate control. The former solution (e.g. [4], [5]) considers the entire NoC to be a *single shared resource* protected by a *global arbiter*. Each application/transmission is granted, in a cyclic order, a dedicated time slot to have an *exclusive* access to the NoC. TDM offers a low support for mixed-criticality as all applications with assigned time-slots are considered to be of equal highest-priority. Because of that SRTTs must wait for all other HRTTs to complete and can only use idle slots e.g. [6]. Moreover, the characteristics of the traffic from general-purpose SRTT applications is usually unknown at design time

978-3-9815370-6-2/DATE16/

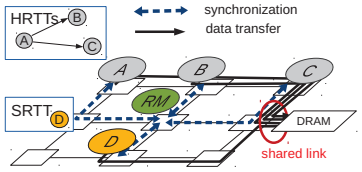


Fig. 1. Admission control using the RM for interfering SRTTs and HRTTs traffic in the NoC.

and transmissions often occur in unpredictable bursts which challenges the static and periodic TDM scheme.

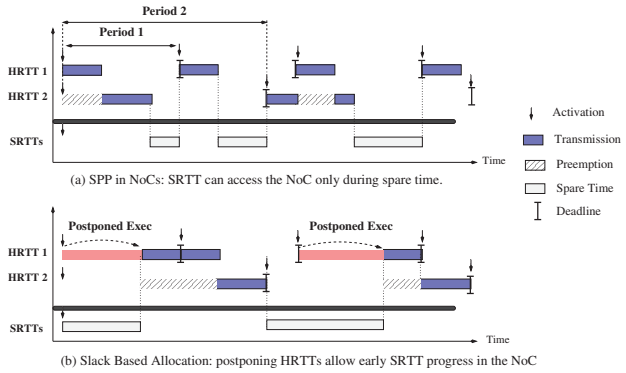
The second solution relies on non-blocking routers with rate control [7]. This scheme is based on a *local arbitration* performed *independently* in routers. The QoS is provided through *dynamic* scheduling between transmissions trying to acquire a shared output port in a router. In order to support mixed-criticality, physical isolation is commonly achieved through the usage of virtual channel (VCs). Each VC is assigned a criticality level (e.g. HRTTs or SRTTs) and routers use static or dynamic priority arbitration between transmissions on different VCs, such as in [3], [8]. Although this approach offers worst-case guarantees and work-conserving scheduling, it introduces high hardware overhead. Firstly, the arbitration is based on the assumption that packets can be forwarded as they arrive [9] i.e. there is no back pressure and no correlation between routers which requires large buffers [4]. Secondly, the constant increase in the number of applications integrated into a single chip, e.g. Flight Management System [10], requires the number of VCs to be equal to the number of criticality levels and to increase accordingly, otherwise the system is not predictable [4]. Finally, these approaches require custom router design.

Our mechanism offers a hybrid approach combining the local arbitration performed in routers with the global scheduling for the end to end guarantees. This allows to overcome the drawbacks of previously described approaches. It reduces hardware overhead compared to non-blocking routers due to the global arbitration (reduced number of VCs and small required buffering) as well as decreases average latencies compared to TDM, i.e. temporal overprovisioning, due to the work-conserving arbitration. Moreover, the introduced overlay network layer significantly simplifies efficient implementation of the slack-based resource allocation due to the global knowledge of the state of the NoC during runtime.

III. Mechanism Description

Concurrent SRTTs and HRTTs in NoCs compete for the following resources in routers: buffers and link bandwidth. Therefore, our mechanism arbitrates between data streams (which can be composed of multiple packets) sharing buffers and whose paths overlap in at least one physical link i.e. *interfering transmissions*, see Fig 1. This requires to provide timing guarantees since the effects of both back pressure and head-of-line blocking can lead to cyclic dependencies and endanger the system safety. In the rest of the paper, we refer to a set of interfering transmissions as a *synchronization scenario*.

The proposed slack-based approach requires a *global* and *dynamic* admission control during runtime. Therefore, we introduce an *access layer* applied within the existing NoC. The control layer is composed of a management unit: the Resource Manager (RM) and a synchronization protocol. Consequently, each interfering transmission (HRTTs as well as SRTTs) must first acquire the right to access the NoC from the RM before sending data. The RM conducts dynamic priority scheduling based on the available HRTTs slack.



(a) SPP in NoCs: SRTT can access the NoC only during spare time.

(b) Slack Based Allocation: postponing HRTTs allow early SRTT progress in the NoC

Fig. 2. Comparison between SPP and slack-based resource allocation in NoCs.

Consider the example depicted in Figure 2, a NoC shared between a set of HRTTs and SRTTs. It constitutes a mixed-critical system where each HRTT has a unique *static* priority (assigned for instance according to a rate-monotonic scheme) and remaining SRTTs have the same lowest priority. Figure 2-(a) illustrates the evolution over time of transmissions in a real-time NoC using a static priority preemptive (SPP) policy [8]. In such systems, SRTTs can progress through the NoC *only* when HRTTs are not sending data, therefore they are often unnecessarily delayed. HRTTs are scheduled as soon as they arrive, although they usually do not profit from faster execution as long as they are guaranteed to finish before their deadline.

On the contrary, in our approach the RM safely *delays* the execution of HRTTs, whenever it is possible, in order to improve the performance of SRTTs. We compute offline for each HRTT a budget called *slack* (see Sec. IV), which defines the maximum delay an HRTT can experience without endangering its deadline. Later, the RM monitors the slack budgets of *currently ongoing* HRTTs and dynamically adjusts the priority of SRTTs accordingly. SRTTs get the highest priority whenever there is an available slack and the lowest priority whenever there is no slack. Note that delaying the execution of an HRTT with the highest priority also delays all currently preempted HRTTs with a lower priority. Therefore, this delay can only occur if enough time (i.e slack) is available for all HRTTs to meet their deadlines. This is illustrated in Figure 2-(b), where HRTTs are postponed, to allow SRTTs to start earlier. Consequently, we manage to drastically reduce the latencies of SRTTs, compared to Figure 2-(a), without endangering the deadlines of HRTTs.

Although the proposed solution can be applied to different NoC architectures, for the initial considerations, we assume the standard baseline NoC: 2D mesh network with wormhole switching, virtual-channel and a deterministic XY-routing.

A. Workflow

The admission control is performed using local supervisors called *clients*. At each node, clients trap outgoing transmissions and synchronize them with the RM. The protocol-based communication between clients and a RM is accomplished using five control messages defining the synchronization protocol: *reqMsg* (request), *relMsg* (release), *ackMsg* (acknowledge), *preMsg* (preempt) and *resMsg* (resume).

At the activation of a new transmission, clients issue a request message (*reqMsg*) to the RM. Upon its arrival at the RM, each request is classified as SRTT or HRTT and stored in the appropriate queue. We use a FIFO for SRTTs and a priority queue for HRTTs.

As described previously, the RM monitors all HRTTs and keeps track of the amount of available slack. When an SRTT is granted access to the NoC, the RM decrements in real time the slack budgets of all active pending HRTTs i.e. it considers all requests in the HRTT queue. The slack budget is renewed at each request arrival ($reqMsg$) from an HRTT. If there are no pending SRTTs or the slack budget of at least one HRTT has been exhausted, HRTTs transmissions regain access to the NoC and are scheduled according to the standard static priority preemptive (SPP) policy. Note that, when SRTTs have access to the NoC, the RM must release the resource early enough to guarantee that HRTTs start on time (i.e not after the slack budget has been consumed). This will be discussed in details in Sec. IV-B.

When the NoC is free, the RM notifies the appropriate client with an $ackMsg$ for the pending transmission to start sending data. Once granted, the connection holds until the end of the transmission or its abortion after a predefined timeout to prevent unbounded connection times. If the NoC is occupied and the RM receives a new request with a higher priority, the RM must preempt the currently ongoing transmission. The preemption is realized with a *preMsg* sent to the client supervising the ongoing transmission. After receiving the *preMsg*, the client blocks the next packet belonging to the ongoing transmission. The preemption on a flit level is unacceptable due to the properties of the wormhole routing. Later, the RM sends the $ackMsg$ with a delay (cf. Sec. IV) to ensure that packets from previously ongoing transmission are not present in the NoC.

When the client detects the end of a transmission (e.g. based on its time-budget/timeout or injection of the last flit), it issues a *relMsg* to the appropriate RM which then removes the corresponding request from the head of the queue. Next, if there is a pending preempted transmission with a lower priority, the RM resumes its execution after sending a *resMsg* to the appropriate client. Otherwise, a new request is scheduled.

Note that the latency of control messages is crucial for the performance of the proposed mechanism. In principle, control messages can be allocated to any available VC capable of giving latency guarantees e.g. [3],[8]. However, for maximum performance one may apply a dedicated independent control NoC, independent signal lines or prioritized VCs so that control messages are not blocked by any other traffic.

IV. Timing Analysis

In order to provide guarantees, we must prove that the approach is predictable as, i) the RM allows to bound the interference between HRTTs and ii) the RM assures that SRTTs cannot postpone HRTTs longer than the available slack budget. Firstly, in Sec. IV-A, we compute for each HRTT the worst-case response time considering only interference from other HRTTs. Then, in Sec. IV-B, we define the bounds on the maximum tolerable slack which can be used to accommodate SRTTs without endangering the system safety. Finally, in the same Sec. IV-B, we present the timing constraints that the RM must enforce to ensure that SRTTs cannot violate slack budgets. Note that, we do not provide any formal timing guarantees for SRTTs since they do not require any.

A. Worst-Case Response Time of HRTTs

For each HRTT, we compute the worst case response time (i.e network latency of a single transmission) using the busy window approach [11], considering all interfering HRTTs (i.e in the same synchronization scenario) synchronized with the same RM. For this, we consider every RM as a *resource* under SPP scheduling and every transmission belonging to the synchronization scenario as a *task*.

Definition 1 (Basic Network Latency Bounds)**.** Let C_i^- and C_i^+ of a transmission i denote the minimum and maximum time required to transfer ω_i packets of a transmission on the NoC when no contention and maximal contention are considered respectively. Therefore, the time during which these packets are physically present in the network can be bounded by $[C_i^-; C_i^+]$. These bounds can be obtained with the analysis from [9] and capture the characteristic of the underlying hardware depending on the source/destination routing distance, the packet size and the link bandwidth.

In order to capture the dynamics of requesting the RM, we use event arrival functions [11], which provide abstractions of the worst-case possible requests behavior.

Definition 2 (Requests Arrival Functions)**.** Let $\eta_i^-(\Delta t)$ and $\eta_i^+(\Delta t)$ constitute the minimum and maximum number of requests to the RM *issued* from an HRT sender i, within a time window of size Δt. Their pseudo-inverse counterparts $\delta_i^-(n)$ and $\delta_i^+(n)$, return the minimum/maximum time interval between the first and the last request in any sequence of n requests arrivals.

Transmission requests arrive to the RM with a worst-case propagation jitter $J_{i,ctrl}$ resulting from the propagation delay in the NoC. This jitter is captured using the method from [12] and is denoted as the difference between minimum $C_{i,ctrl}^-$ and maximum $C_{i,ctrl}^+$ network latency for a control packet,

$$J_{i,ctrl} = C_{i,ctrl}^+ - C_{i,ctrl}^- \tag{1}$$

Based on this definition, we derive the event models of request arrivals to the RM from a transmission i based on [12], which also contains proof for Lemma 1.

Lemma 1. *The input event/request model from an HRT transmission i to the RM, is defined as:*

$$\delta_{i,RM}^-(n) = \max\{\delta_i^-(n) - J_{i,ctrl}\ ,\ (n-1)\cdot C_{i,ctrl}^-\} \tag{2}$$

where, $\delta_i^-(n)$ denotes the minimum time interval necessary for n activations of a transmission i. Note that $\delta_{i,RM}^-$ can be straightforwardly converted to $\eta_{i,RM}^+$ with the method in [11].

Busy window analysis. In the first step, we construct a critical instant which considers the worst-case arrival sequence of events, events' duration and the scheduling policy to compute the maximum delay that a transmission (task) may suffer, from all interfering transmissions. It maximizes the response time, denoting the duration between the activation of a transmission and its completion. We assume conservatively that requests from all HRTTs arrive at the same moment to the RM. Moreover, transmissions cannot be nested i.e. in order to send a new request previous one must be finished and released, this must be ensured by clients.

Let the maximum q-event busy windows $\omega_i^+(q)$ describe the maximum time interval required to complete q consecutive activations of an HRTT i belonging to a synchronization scenario protected by a RM.

Theorem 2. *The maximum q-event busy windows $\omega_i^+(q)$ is bounded by:*

$$\omega_i^+(q) \leq q\cdot C_i^+ + 3q\cdot C_{i,ctrl}^+ + B_{i,DI}(\omega_i^+(q)) + B_{i,IND}(\omega_i^+(q)) \tag{3}$$

Proof: $(q \cdot C_i^+)$ is the mimimum network latency to serve q requests from transmission i, $(3q \cdot C_{i,msg}^+)$ is the latency of control messages ($reqMsg$, $ackMsg$, $relMsg$) required for each transmission, $B_{i,IND}(\omega_i^+(q))$ is the maximum blocking resulting from the preemption of HRTTs with a lower-priority than i and $B_{i,DI}(\omega_i^+(q))$ is the maximum blocking time due to HRTTs with a higher priority than i. ■

Note that the term $\omega_i(q)$ appears on both sides of Eq. 3 which constitutes an integer-fixed point problem. It can be solved iteratively starting with $\omega_i(q) = q \cdot C_i^+ + 3q \cdot C_{i,ctrl}^+$.

Lemma 3. *Direct blocking (DI) that an HRTT i experiences during a time Δt due to preemptions from the set $hp(i)$ of higher-priority HRTTs in the same synchronization scenario can be bounded by:*

$$B_{i,DI}(\Delta t) = \sum_{\forall j \in hp(i)} \eta_{j,RM}^+(\Delta t) \cdot (C_{i,ctrl}^+ + C_j^+ + 2 \cdot C_{j,ctrl}^+) \quad (4)$$

Proof: The direct blocking considers both the execution time of higher priority HRTTs and the additional protocol overhead required to manage the preemtions. Following the assumed SPP arbitration policy, each transmission $j \in hp(i)$ may block/preempt the considered transmission $\eta_{j,RM}^+(\Delta t)$ times, for the duration C_j^+ of maximum network latency. Additionally, each preemption requires to send $ackMsg$ and receive $relMsg$ $(2 \cdot C_{j,ctrl}^+)$ from the higher-priority HRTT, plus a restitution message $resMsg$ $(C_{i,ctrl}^+)$. ■

Lemma 4. *Indirect blocking (IND) that an HRTT i experiences during a time Δt due to the preemptions of the set $lp(i)$ of lower priority HRTTs belonging to the same synchronization scenario, can be bounded by:*

$$B_{i,IND}(\Delta t) = \min\{q, \sum_{\forall j \in lp(i)} \eta_{j,RM}^+(\Delta t)\} \cdot [\max_{\forall j \in lp(i)}(C_{j,ctrl}^+) + \max_{\forall j \in lp(i)}(C_{j,pckt}^+)] \quad (5)$$

where, $C_{j,pckt}^+$ denotes the maximum latency of a single packet belonging to the transmission j.

Proof: Each q transmissions i can maximally preempt q transmissions belonging to $lp(i)$. Because in the worst-case, the maximal number of activations of tasks from $lp(i)$ may be lower than q, we select the minimum from both. Each preemption request introduces an additional delay because the higher-priority request must wait until the control message (*blckMsg*) arrives to the appropriate client. We conservatively assume the longest possible time of *blckMsg* for all transmissions belonging to $lp(i)$. Moreover, RM must ensure that all packets from the preempted transmission leave the NoC before i starts. This is conservatively guaranteed by selecting the maximum latency time of a single packet (i.e. the last) from all transmissions belonging to $lp(i)$. ■

The *worst-case response time (WCRT)*, required to complete the transmission of q activations of an HRTT i in the worst case is denoted R_i. It is defined as the difference between the busy window $\omega_i(q)$ and the earliest possible activation $\delta^-(q)$.

$$R_i = \omega_i(q) - \delta_i^-(q) \quad (6)$$

B. Slack Constraints

After considering the maximum interference between HRTTs to compute the worst-case response time, we derive the constraints on the slack values required for a safe applicability of the mechanism. The *initial slack* budget of an HRTT$_i$ is the difference between its deadline and the WCRT:

$$S_i = D_i - R_i \quad (7)$$

The initial slack is computed offline and known to the RM. Note that, the constraint $R_i \leq D_i$ must be satisfied otherwise the system is not schedulable.

Whenever an HRTT is activated i.e. a $reqMsg$ is sent to the RM, the RM updates the slack budget of this HRTT with the *initial slack* value. We derive for each synchronization scenario λ the *minimum* (S_{min}) and *maximum* (S_{max}) initial slack budgets:

$$S_{min} = \min_{\forall j \in \lambda_{HRTT}} (D_j - R_j) \;\; ; S_{max} = \max_{\forall j \in \lambda_{HRTT}} (D_j - R_j) \quad (8)$$

where, λ_{HRTT} denotes the subset of all HRTTs belonging to λ. These budgets define the margins of performance improvement provided for SRTTs.

During runtime, the RM must constantly monitor the *available slack* and update its value while scheduling SRTTs and protecting HRTTs. Recall that an HRTT can be delayed only if i) it has enough available slack and ii) all other lower priority HRTTs have enough available slack. Therefore, the *available slack* provided for SRTTs at time t is equal to:

$$S(t) = \min\left(S_i(t) \;, \min_{\forall j \in lp(i)} S_j(t)\right) \quad (9)$$

where i is the highest priority HRTT *active* at time t.

Due to the synchronization protocol inducing additional time overhead, the RM can allow the execution of SRTT$_i$ after the preemption of an ongoing HRTT$_j$ only when:

$$S(t) \geq C_{j,ctrl}^+ + \psi + C_{i,ctrl}^+ \quad (10)$$

where $C_{i,ctrl}^+$ denotes the time necessary to send $ackMsg$ to SRTT and $C_{j,ctrl}^+$ is the restitution message $resMsg$ to the currently ongoing HRTT. The values of $C_{i,ctrl}^+$ and $C_{j,ctrl}^+$ might be different as they depend on the mapping of HRTTs and SRTTs and their distance from the RM. ψ denotes the preemption's latency derived from Lemma 4, for managing the preemptions of SRTTs by the RM when the slack budget is exhausted.

$$\psi = [\max_{\forall j \in \lambda_{SRTT}}(C_{j,ctrl}^+) + \max_{\forall j \in \lambda_{SRTT}}(C_{j,pckt}^+)] \quad (11)$$

where λ_{SRTT} denotes the subset of all SRTTs belonging to λ. This time overhead must also be consumed from the available slack budget.

Therefore, the RM must monitor ongoing SRTTs and conduct preemption early enough so that the delayed HRTTs may start without violating its deadline i.e. it must enforce SRTTs preemption whenever the value of the available slack reaches ψ. Consequently, the value of the *available slack* must vary during runtime between initial slack budgets and ψ.

V. Experimental Results

In this section, we evaluate the proposed mechanism using real-life applications and benchmarks. We focus on network traffic concerning memory transfers as memories constitute the most common hot module in a system-on-chip thereby challenging the performance and scalability of MPSoCs (cf. [13]). We assume that HRTTs are DMA-based predictable transfers whereas SRTTs are cache-based sporadic transfers.

We model HRTTs after the MPEG-4 video decoder application [14]. The decoder's demands with respect to the

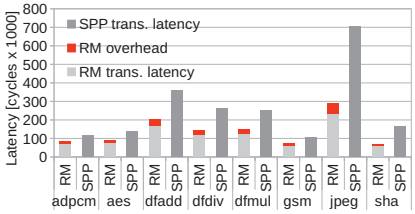


Fig. 3. Performance of CHSTONE benchmarks (SRTTs) synchronized with MPEG-4.

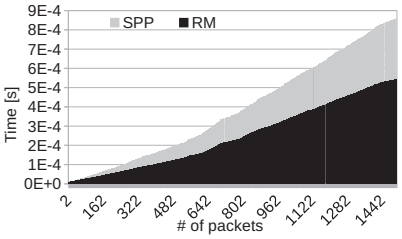


Fig. 4. Trace with completed transmissions from the SRTT *dfmul* benchmark.

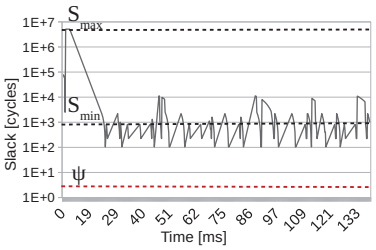


Fig. 5. Slack budget monitored by the DRAM's RM. (Available slack $\geq$ SRTTs requirements).

interconnect resources are presented in Fig 6(a). We identify three memory modules with high communication requirements: DRAM (7 senders), SRAM2 (4 senders) and SRAM1 (2 senders). Note that transmissions to different memory modules constitute independent/disjoint synchronization scenarios, therefore arbitrated using different RMs for increased scalability. In the sequel, we focus on the DRAM memory module with the highest interconnect demand. Each task from MPEG-4 is modeled with a traffic generator conducting 8kB long DMA transfers which allows to maximize the benefit from DDRAM3 2133N [15]. Transmissions are performed periodically and periods are calculated based solely on the required bandwidth including some release jitter (J=10% of P). Priorities of transmissions are assigned using rate monotonic scheme i.e. the shortest period equals to the highest priority. For SRTTs we use activation traces of applications from the CHSTONE benchmark [16]. Traces were extracted using the Gem5 simulator and an ARMv7-a core with a 32 kB L1 cache and 64 Bytes long cache-line. Simulations are carried out with OMNeT++ event-based simulation framework and HNOCS library [17].

The evaluation is done through comparison with SPP-based solutions (cf. Sec. II) where SRTTs are always statically assigned the lowest priority. We do not compare our solution against TDM since it is not work-conserving and would drastically decrease the performance of SRTTs not optimized for this scheme, thereby making the comparison unfair. We compute the slack budgets for HRTTs with analysis from Sec. IV implemented in the pyCPA framework [11]. We use the approach from [9] to derive basic network latencies ($C_i^{+/-}$ and $C_{i,ctrl}^{+/-}$). Finally, Fig 6(b) shows the considered mapping for the DRAM synchronization scenario. HRTTs as well as SRTTs are mapped to the same VC and synchronized with the same RM.

A. Application and Benchmark-Based Results

We start with the evaluation for SRTTs of the performance gain achieved through the proposed mechanism. Fig. 3 presents latencies of CHSTONE benchmarks (SRTTs) synchronized with tasks from MPEG-4 (HRTTs) and belonging to the DRAM's synchronization scenario. Results concern both SPP and RM using slack-based arbitration. We observe that benchmarks synchronized with RM finish faster compared to SPP despite the additional overhead resulting from the synchronization protocol. The improvement varies among applications to reach a speedup value of nearly 60% for jpeg application, This depends on the duration and the frequency of blocking in the NoC resulting from the access patterns of soft and hard real-time applications. The speedup is also proportional to the number of memory accesses, e.g. *jpeg* exposes the highest number of transfers and thereby highly benefits from the improvement in network latency. Finally, it is visible that the protocol overhead remains very low compared to the transmission time (from 5 to 14 %).

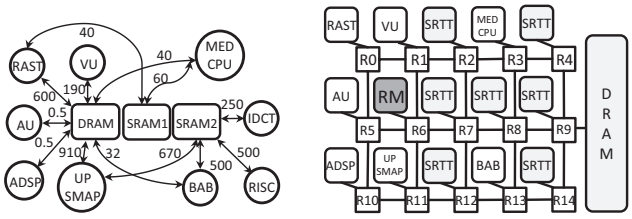


Fig. 6. (a) MPEG-4 (HRTTs) average communication demands specified in MB/s, (b) mapping of MPEG-4 with interfering SRTTs.

For detailed explanation, Fig. 4 presents the evolution over time of the number of completed transmissions from the *dfmul* benchmark. It is visible that the application progresses significantly faster when synchronized with the RM. The slack-based resource allocation allows SRTT to fetch data and start *early* processing therefore accelerating benchmark's execution when compared to SPP.

As explained previously, the slack budget available to SRTTs varies at runtime. Fig. 5 presents the evolution of the available slack over a time window of 200 ms in the considered DRAM scenario. Raising edges in the Figure correspond to the provisioning of the slack budget (between S_{min} and S_{max}) when a new HRTT is activated, and falling edges to the consumption of the slack budget when an SRTT is granted an access to the NoC. It is visible that the value of the available slack varies between initial slack budgets (S_{min}/S_{max}) and ψ. Consequently, as the slack never reaches ψ, SRTTs can be safely accelerated by postponing the execution of HRTTs without causing a deadline miss. Moreover, we may observe that the available slack stays on a relatively high level (compared to ψ). This means that the provided slack budget is larger than the SRTTs requirements. Therefore, the average performance of HRTTs has not decreased significantly.

Scalability. In the next series of experiments, we evaluate the performance of CHSTONE benchmarks (SRTTs) in setups with increasing workloads from synthetic HRTTs benchmarks. We consider synchronization scenarios with x senders, each performing a burst of y hard real-time transmissions per activation. Senders are activated periodically with a period $P = 16 \cdot x \cdot C^+$ and a small jitter J equal to 10% of the period. We assume that all synchronized transmissions are of equal length i.e. they have the same C^+. However, we vary the system's load L defined as the total number of HRTTs per period P i.e. $L = \sum_x (y_x \cdot C^+)/P$. Results are presented in Fig. 7 where the highest performance gain for SRTTs is achieved when the HRTT's load is high but the system is not fully loaded (L between 60% and 80%). This constitutes the sweet spot between the number of HRTTs providing high slack budgets and the overall, *non full*, system load giving a chance to SRTTs to use the provided slack. Indeed, if the system is fully loaded ($L > 90\%$), SRTTs cannot benefit from the available slack since HRTTs are using the NoC more than 90% of the time. Besides, the temporal overhead of the

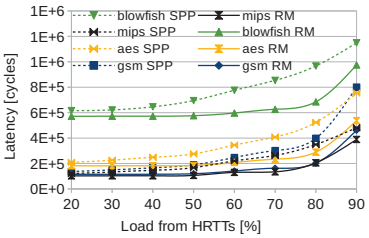


Fig. 7. Performance of SRTTs CHSTONE benchmarks for different levels of interfering HRTTs load.

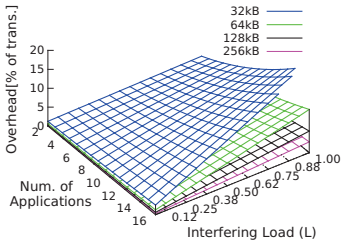


Fig. 8. Temporal overhead resulting from RM.

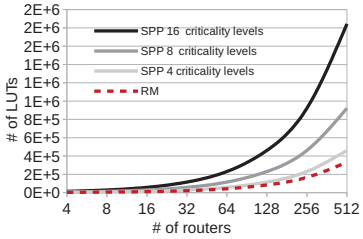


Fig. 9. Area overhead resulting from RM.

required synchronization protocol increases significantly their latencies making SPP a comparable solution. Similarly, if the load from HRTTs is very low then the interference with SRTTs is sporadic and the possible performance gain very limited compared to SPP. However, in such setups using the RM is still justified since it reduces the hardware requirements.

The protocol timing overhead depicted in Fig 3 concerns only the considered DRAM scenario. In order to assess the scalability of the approach, we conduct a series of experiments to measure the temporal overhead by varying the number of synchronized applications and the system load. Fig. 8 summarize the results presented as a percentage of the transmission's length. The overhead increases proportionally to the number of synchronized applications and their activity i.e. load L. Note that this is directly related to the frequency of transmissions. If the system is composed of many senders that are rarely transmitting data, then the overhead remains on a low level. In the considered safety critical systems, the behavior of HRTTs is well specified and tested, and the frequency of HRTTs limited (e.g. periodic in ms). The overhead decreases, as an absolute ratio, with increasing length of transmissions as it is constant w.r.t the transmission length. DMA engines imposing large granularity of transfers combined with scratchpad memories are mostly used for safety requirements as they are predictable compared to caches. The temporal overhead can be mitigated by implementing multiple RMs in the same NoC (for instance as in the described use case, different RMs for different memory modules). Moreover, large synchronization scenarios can be decomposed into smaller ones by mapping transmissions to different VCs supervised by independent RMs.

Implementation Overhead. The evaluation of the hardware overhead was done in the IDAMC platform [18] on a Virtex-6-LX760 Xilinx FPGA. The clients required 200 Look-up tables (LUTs), which corresponds only to 3% of the area of a Network Interface (NI) module whereas RM required approx. 1200 LUTs which corresponds to 16 % of the area of the NI. Although RM introduces area overhead, it simultaneously decreases the number of required VCs. For instance, in the considered DRAM synchronization scenario, RM requires 2 VCs (control messages and transmissions) whereas SPP requires 8 VCs (one for each criticality level). Fig. 9 presents the area overhead resulting from the implementation of RM and SPP in NoCs for different sizes and number of employed criticality levels. Results are based on the implementation and synthesis in the IDAMC platform with 5 flit buffer per VC per ingress port. In case of small NoCs with only few HRTTs, the overhead of SPP and RM are comparable e.g. 3600 vs 3200 LUTs for 4 crticality levels. However, in case of larger designs, RM allows to significantly reduce area overhead, e.g. for a design with 8 crticalities and 32 routers, the area required by RM is three times smaller than in case of SPP.

VI. Conclusion

The efficient and safe support of a broad diversity of traffic requirements in NoCs is crucial for the success of MPSoCs in the market of safety critical and real-time systems. In this work, we proposed a novel mechanism, which achieves this goal through a slack-based, global and dynamic prioritization of data streams. Our solution introduces an *overlay network* which decouples the mechanisms responsible for admission control in NoC from the underlying infrastructure conducting switch arbitration. The proposed approach allows to decrease both temporal and hardware overhead while significantly improving the overall performance of the system and meeting the strict timing constraints.

Acknowledgment

This work was funded within the EMC2 project by the German Federal Ministry of Education and Research with the funding ID 01—S14002O and by the ARTEMIS Joint Undertaking under grant agreement n° 621429. The responsibility for the content remains with the authors.

References

[1] J. W. S. W. Liu, *Real-Time Systems*. 2000.
[2] R. Davis, K. Tindell, and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems," in *RTSS*, 1993.
[3] J. Diemer and R. Ernst, "Back suction: Service guarantees for latency-sensitive on-chip networks," in *NOCS*, 2010.
[4] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *DAC*, 2010.
[5] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "Phasenoc: Tdm scheduling at the virtual-channel level for efficient network traffic isolation," in *DATE*, 2015.
[6] 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.
[7] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," in *Proc. of the IEEE*, 1995.
[8] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *NoCS 2008*, 2008.
[9] E. A. Rambo and R. Ernst, "Worst-case communication time analysis of networks-on-chip with shared virtual channels," in *DATE*, 2015.
[10] G. Durrieu and et al., "Predictable flight management system implementation on a multicore processor," in *ERTS*, 2014.
[11] J. Diemer, P. Axer, and R. Ernst, "Compositional performance analysis in python with pycpa," in *WATERS*, jul 2012.
[12] S. Schliecker and et al, "Providing accurate event models for the analysis of heterogeneous multiprocessor systems," in *CODES+ISSS*, pp. 185–190, ACM, 2008.
[13] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha, "Handling mixed-criticality in soc-based real-time embedded systems," in *EMSOFT*, ACM, 2009.
[14] D. Bertozzi, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *Parallel and Distributed Systems*, vol. 16, pp. 113–129, Feb 2005.
[15] JEDEC, Arlington, Va, USA, *JESD79-3F: DDR3 SDRAM Specification*, July 2012.
[16] Y. H. et al., "Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis," *Journal of Information Processing*, 2009.
[17] Y. Ben-Itzhak and et al., "Hnocs: Modular open-source simulator for heterogeneous nocs," in *SAMOS*, 2012.
[18] S. Tobuschat, P. Axer, R. Ernst, and J. Diemer, "Idamc: A noc for mixed criticality systems," in *RTCSA*, 2013.