A Globally Arbitrated Memory Tree for Mixed-Time-Criticality Systems

Manil Dev Gomony, Jamie Garside, Benny Akesson, Neil Audsley, and Kees Goossens

Abstract—Embedded systems are increasingly based on multi-core platforms to accommodate a growing number of applications, some of which have real-time requirements. Resources, such as off-chip DRAM, are typically shared between the applications using memory interconnects with different arbitration polices to cater to diverse bandwidth and latency requirements. However, traditional centralized interconnects are not scalable as the number of clients increase. Similarly, current distributed interconnects either cannot satisfy the diverse requirements or have decoupled arbitration stages, resulting in larger area, power and worst-case latency.
The four main contributions of this article are: 1) a Globally Arbitrated Memory Tree (GAMT) with a distributed architecture that scales well with the number of cores, 2) an RTL-level implementation that can be configured with five arbitration policies (three distinct and two as special cases), 3) the concept of mixed arbitration policies that allows the policy to be selected individually per core, and 4) a worst-case analysis for a mixed arbitration policy that combines TDM and FBSP arbitration. We compare the performance of GAMT with centralized implementations and show that it can run up to four times faster and have over 51% and 37% reduction in area and power consumption, respectively, for a given bandwidth.

Index Terms—Real-time systems, Globally Arbitrated Memory Tree, GAMT, Shared memory, Latency-rate Servers, Mixed-Time-Criticality, Scalability.

1 Introduction

The complexity of embedded systems is growing, as more and more applications are being integrated into modern systems [1]. In consumer electronics, this trend is caused by a digital convergence of application domains, which has led to highly integrated devices like smart phones and connected ultra-high definition television sets. Some of the applications have *real-time (RT) requirements* and must always finish their computations before a pre-defined deadline. Missing a deadline may result in unacceptable distortions of audio and video or in failure to implement a given standard, which may result in considerable loss of business. In contrast, other non-real-time (NRT) applications do not have explicit deadlines and only try to execute as fast as possible to feel responsive. A particular challenge is that RT and NRT applications are deployed and executed on the same platform, resulting in *mixed-time-criticality systems*, even though they benefit from different design choices with respect to predictability and performance.

To satisfy the computational requirements of an increasing number of applications with low size, weight and power consumption, (heterogeneous) multi-core platforms with shared resources, such as off-chip memory, are used [2], [3]. Sharing off-chip memory between cores, which we refer to as *memory clients*, is typically done using a memory interconnect that multiplexes requests from different clients using some *arbitration policy*. The choice of arbitration policy depends on the requirements of the clients, which may be diverse in terms of bandwidth and/or latency [4], [5]. Time-Division Multiplexing (TDM) is a good policy for important and less dynamic RT clients, such as display controllers, since it provides temporal isolation when used in a non-work-conserving manner (empty TDM slots are not used). This makes their temporal behavior independent from other applications, which enables incremental verification and reduces the costly verification effort [6]. However, non-work-conserving TDM provides poor average-case performance, since slack (empty slots) is not used. This is a problem in the context of mixed-time-criticality systems, where work-conserving arbitration that exploits slack to improve average performance is beneficial for NRT clients. A work-conserving Round-Robin (RR) arbiter may hence be suitable in case bandwidth and latency requirements of NRT clients are fairly homogeneous and a priority-based arbiter in case they are diverse. A great deal of flexibility is hence required from the memory interconnect in a reusable platform for mixed-time-criticality systems.

Existing memory interconnects struggle with these trends and requirements for at least one of the following three reasons: 1) they are not scalable in terms of area and power with the increasing number of clients, 2) they do not synthesize at the ever higher clock frequencies of off-chip memories [7], 3) they only support a single arbitration policy and cannot provide sufficient flexibility for a reusable platform targeting mixed-time-criticality systems.

This article addresses this innovation gap by proposing a *Globally Arbitrated Memory Tree* (GAMT) for complex mixed-time-criticality systems. The four main highlights of this article are: 1) a distributed architecture that scales

- *Manil Dev Gomony and Kees Goossens are with the Eindhoven University of Technology, The Netherlands.*
  Email: {m.d.gomony, k.g.w.goossens}@tue.nl
- *Jamie Garside and Neil Audsley is with University of York, United Kingdom.*
- *Benny Akesson is with CISTER/INESC TEC and ISEP, Polytechnic Institute of Porto, Portugal.*

Manuscript received Month Date, Year; revised Month Date, Year.

well in terms of area, power consumption and maximum frequency as the number of clients increases. 2) the architecture supports five well-known arbitration mechanisms (three distinct and two as special cases) in either work-conserving or non-work-conserving mode, 3) the choice of arbitration policy can be configured *per client* instead of for all clients, further increasing arbitration flexibility, 4) a worst-case analysis for a mixed arbitration policy, where some clients use non-work-conserving TDM and others work-conserving Frame-Based Static Priority (FBSP) [8]. We experimentally verify that our hardware implementation of GAMT behaves identically to the arbitration policies it is configured to mimic and compare our distributed architecture in terms of area, power, and maximum frequency to centralized implementations. We also show that our derived bound for the mixed arbitration policy is conservative and that the clients under non-work-conserving TDM arbitration enjoy temporal isolation.

The remainder of this article is organized as follows. Related work is first discussed in Section 2, after which Section 3 presents essential background material. Our Globally Arbitrated Memory Tree is then introduced in Section 4, followed by a worst-case analysis for a novel mixed arbitration policy combining TDM and FBSP in Section 5. Experimental results are then presented in Section 6, before we draw conclusions in Section 7.

2 Related work

There is a large body of work focusing *mixed-criticality systems* [9], which is a term referring to safety-critical systems, e.g. in the automotive and avionics domains, featuring a mix of *safety levels*. In this context, hardware components like networks-on-chips (NoCs) [10] and memory controllers [11], [12] have been developed around the common theme that tasks or transactions of non-critical clients are dropped when critical clients overrun their execution bounds. In contrast, our work does not consider safety levels, but distinguishes clients with a mix of real-time and best-effort requirements and focus on satisfying their different needs for worst-case and average-case performance.

Existing memory interconnect architectures with predictable arbitration policies can be classified into *centralized* and *distributed* architectures based on their implementation. In a centralized implementation, the arbitration policy is implemented in a single physical location. Centralized architectures are easy to implement as the arbitration decision is made at a central location using a single arbiter for all clients. The centralized implementations in [13], [14], [15], [16] consist of a tree of multiplexer stages for priority resolution among the clients and are not scalable in terms of clock frequency. This is because the number of logic gates in the critical path for multiplexing increases with the number of clients, restricting their maximum synthesizable frequency. This issue can be resolved by pipelining the multiplexer stages, although this introduces additional arbitration delays that must be considered both in performance analysis and to guarantee functional correctness of the arbitration. This is not discussed in the existing work mentioned above, but is covered in this article.

In distributed architectures, arbitration of memory clients is performed in a distributed manner using multiple arbitration nodes [17], [18], [19], [20], [21]. This deals with the scalability problem with clock frequency by breaking up arbitration into multiple smaller steps with less clients. Distributed memory interconnects can be further classified as either *locally arbitrated* or *globally arbitrated* depending on whether the arbitration nodes work independently or in a coordinated manner. Distributed memory interconnects with local arbitration are presented in [17], [18], [21]. These architectures consist of multiple arbitration stages connected in a tree-like structure, where each arbitration stage uses RR arbitration in [17], [21] and First-Come First-Serve (FCFS) in [18]. In [22], a combination of non-work-conserving TDM and work-conserving RR are used with the root arbiter coordinating the scheduling decisions from the other arbitration stages. Another example of distributed interconnects with local arbitration is NoCs using priority-based packet switching [10], [23]. The main problem with distributed arbitration is that multiple independent arbitration stages leads to larger area and power usage due to the buffering of memory requests at every arbitration stage [24]. This problem can be avoided by means of back-pressure between the arbitration stages, as proposed in [18], although at expense of reduced performance in terms of latency and throughput.

The problem with distributed interconnects with local arbitration is addressed by global arbitration, where the scheduling decisions in different arbitration steps are coordinated to eliminate the need for expensive intermediate buffers. The most prominent example of this type of arbitration is found in TDM-based NoCs [19], [20], [24], where coordination is provided by a statically computed global schedule. However, TDM arbitration is not suitable when clients have diverse bandwidth and latency requirements. For example, clients with low latency and low bandwidth requirements must be allocated more than their required bandwidth to meet their latency requirements, which is not desirable when memory bandwidth is scarce. Although [25] presents a NoC with global arbitration while providing differential treatment to the clients using resource managers, the synchronization of scheduling with control messages reduces overall memory access performance.

From this review, we conclude that there is currently no memory interconnect that scales well in terms of area, frequency, and power consumption, while addressing diverse memory client requirements. This article addresses this by proposing a Globally Arbitrated Memory Tree with a scalable distributed implementation. It supports five well-known arbitration policies, which can even be selected per client to further increase the arbitration options. GAMT was previously briefly introduced in [26]. This article extends this work by explaining the architecture and operation in more detail, as well as introducing the novel concept of selecting arbitration algorithm per client. We demonstrate this feature by proposing to mix non-work-conserving TDM arbitration for RT clients and work-conserving FBSP arbitration for NRT clients, a combination that is relevant for mixed-time-criticality systems with diverse requirements. We present the worst-case analysis for this combination and experimentally show that the bound is conservative and that TDM clients are temporally isolated.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2016.2595581, IEEE Transactions on Computers

3 Background

This section provides the necessary background information to understand the contributions of this article. First, Section 3.1 explains the concept of latency-rate servers, which is the framework we use as a base for analysis of GAMT. Section 3.2 then presents the five different existing arbitration policies that the memory tree supports and describes how they fit with the general latency-rate framework. Lastly, we explain our assumptions on the real-time memory controller connected to the memory tree in Section 3.3.

3.1 Latency-Rate Servers

Latency-rate ($\mathcal{LR}$) [27] servers is a shared resource abstraction that guarantees a client c_i sharing a resource a minimum allocated rate (bandwidth), ρ_i, after a maximum service latency (interference), Θ_i. The guaranteed service provided to a client is independent of the actual behavior of others and is based on resource reservations, combining a notion of *accounting* (budgeting) and *enforcement* (e.g. no more service when budget is depleted). The values of Θ_i and ρ_i of each client depend on the particular choice of arbiter and the reservations, as later explained in Section 3.2.

Figure 1 illustrates the requested service of a client over time from a shared resource (upper solid line) and the provided service from the resource (lower solid line). The $\mathcal{LR}$ service guarantee, the dashed line labeled 'service bound' in the figure, provides a lower bound on the data that can be transferred to a client during any interval of time. This makes the $\mathcal{LR}$ server abstraction suitable for performance analysis of streaming applications, such as audio and video encoders/decoders [28], [29], [30], and wireless radios [30], that are more concerned with the time to serve *sequences of requests* rather than just a single request.

The main advantage of the $\mathcal{LR}$ abstraction is that it captures the behavior of many different resources arbiters and configurations in a unified manner. It can furthermore be integrated with well-known performance analysis frameworks, such as network calculus [31], data-flow analysis [32], or worst-case execution time (WCET) estimation [33], allowing techniques from these frameworks to be applied for any arbiter belonging to the class. Examples of the $\mathcal{LR}$ abstraction being used to verify a system in the contexts of network calculus and data-flow analysis are provided in [30] and [34], respectively.

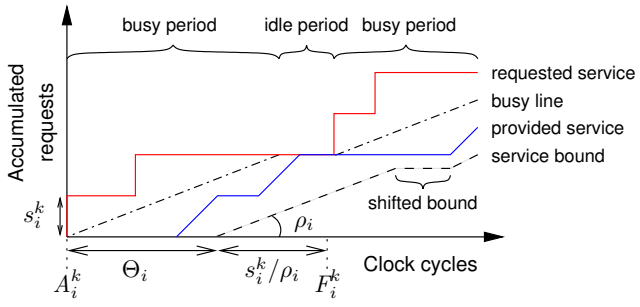


Fig. 1. A $\mathcal{LR}$ server and associated concepts.

The $\mathcal{LR}$ service guarantee is conditional and only applies if the client requests enough service to keep the server busy. This is captured by the concept of *busy periods*, which are periods in which a client requests at least as much service as it has been allocated (ρ_i) on average. This is illustrated in Figure 1, where the client is in a busy period when the requested service curve is above the dash-dotted line with slope ρ_i that we refer to as the *busy line*. The figure also shows how the service bound is shifted when the client is not in a busy period (idle period). We have now introduced all the necessary concepts to formally define a $\mathcal{LR}$ server in Definition 1.

Definition 1 ($\mathcal{LR}$ server). *A server is a $\mathcal{LR}$ server if and only if a non-negative service latency Θ_i can be found such that the provided service, w_i^l, of a client c_i is bounded by Equation (1) during a busy period with duration l. The minimum non-negative constant Θ_i satisfying the equation is the service latency of the server.*

$$w_i^l \geq \max(0, \rho_i \cdot (l - \Theta_i)) \tag{1}$$

Based on the $\mathcal{LR}$ service guarantee, it has been shown [35] that the worst-case finishing time of the k^{th} request from a client i can be bounded by Equation (2), where s_i^k is the request size, A_i^k is the arrival time, and F_i^{k-1} is the worst-case finishing time of the previous request from the client. This bound is visualized for the k^{th} request in Figure 1. Note that this bound is slightly pessimistic and that it has been shown that a reduced service latency $\Theta_i' = \Theta_i - 1/\rho_i + 1$ can be used in Equation (2) [36].

$$F_i^k = \max(A_i^k + \Theta_i, F_i^{k-1}) + s_i^k/\rho_i \tag{2}$$

3.2 Predictable Arbitration Policies

After introducing $\mathcal{LR}$ servers, this section presents five existing arbitration policies that belong to the class, Time-Division Multiplexing (TDM) [8], [30], Round Robin, Frame-Based Static Priority (FBSP) [8], Priority-Based Budget Scheduler (PBS) [37], [38] and Credit-Controlled Static-Priority (CCSP) [16]. For each of these policies, we first explain their basic operation, followed by a brief description of their corresponding bounds on service latency and allocated rate.

3.2.1 TDM and Round Robin

A TDM arbiter operates by periodically repeating a schedule, or frame, with a fixed number of slots, f, each corresponding to a single resource access. Every client c_i is statically allocated a number of slots ϕ_i in the schedule at design time, resulting in an allocated rate according to Equation (3). GAMT assumes that the slots allocated to a client appear consecutively in the schedule, as shown in Figure 2. This assumption reduces both the hardware cost of the implementation and the complexity of latency analysis compared to arbitrary slot allocations. For a consecutive slot allocation, the service latency of a client (in slots), Θ^{tdm}, can simply be computed according to Equation (4) [8]. This corresponds to the worst-case scenario where the busy period of a client starts just after the last slot allocated to the client to maximize the number of interfering slots. More complex methods for determining the service latency for TDM arbiters with arbitrary slot allocations are presented

in [39], [40]. Note that RR arbitration is a special case of TDM, where each client is assigned a single slot in the frame, and is hence covered by the same worst-case analysis.

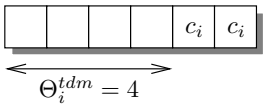


Fig. 2. TDM schedule with frame size $f = 6$ and $\phi_i = 2$ allocated slots to client c_i in a continuous manner. The allocated rate of c_i is $\rho_i = 2/6$.

$$\rho_i = \phi_i / f \tag{3}$$

$$\Theta_i^{tdm} = f - \phi_i = f \cdot (1 - \rho_i) \tag{4}$$

3.2.2 FBSP and PBS

Similarly to TDM, FBSP [8] is also a frame-based arbiter with a fixed frame size and each client c_i is allocated a *budget* of slots, ϕ_i. However, unlike in TDM, there is no static assignment of clients to the slots. Instead, each client is assigned a unique static priority and the (backlogged) client with the highest priority and one or more remaining budget slots is granted service. When a client is granted service, its budget is reduced by one. Note that in this article, we refer to the clients with sufficient budget to get scheduled as *eligible clients*. The budgets of the clients are reset to the number of allocated slots at the end of each frame, making the frame size the *replenishment interval* of the clients. Left-over budget is not preserved between frames to prevent high-priority clients from building up large budgets when they are idle and later starve low-priority clients.

Since FBSP is frame-based just like TDM, it follows that the allocated rate is determined by Equation (3). The worst case for a client c_i under FBSP arbitration is if a request arrives starting a busy period at the same time as all clients in the set of higher priority clients, HP_i. If this happens $\sum_{\forall c_j \in HP_i} \phi_{c_j}$ slots from the end of the frame, the higher priority clients interfere maximally just before the frame repeats, replenishing their budgets and enabling them to interfere maximally again before c_i gets access to the resource. The service latency hence considers two times the maximum interference from higher priority clients. This is expressed in Equation (5) [8] and illustrated in Figure 3. Note that PBS [37], [38] can be seen as a special-case of FBSP where there is a single high-priority client. This is covered by the same worst-case analysis if all low-priority clients assume they have the lowest priority.

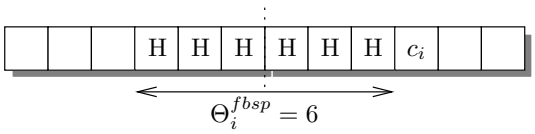


Fig. 3. FBSP arbiter with frame size $f = 6$ and $\phi_H = 3$ allocated slots to higher priority clients.

$$\Theta_i^{fbsp} = 2 \cdot \sum_{\forall c_j \in HP_i} \phi_{c_j} \tag{5}$$

3.2.3 CCSP

The distinguishing feature of the CCSP accounting mechanism is that it does not base budget replenishment on a common frame. Instead, it uses a *continuous replenishment* strategy that gives every client their allocated fraction of a service unit, ρ, before every arbitration decision. In terms of frame-based mechanisms, this can be understood as having a frame size of one and allocate fractional slots to the clients. A hardware implementation of this mechanism is presented in [41].

The allocated service in a CCSP arbiter consists of two parameters. In addition to the allocated rate used by both TDM and FBSP, a client is associated with an *allocated burstiness*, σ. The allocated burstiness determines the initial budget of a client when it starts an active period (similar to a busy period). For every arbitration decision, the budget is incremented by the fractional allocated rate ρ and decremented by one when the client is granted service. When the client is not backlogged, it is only allowed to build up its budget until its initial budget value to bound the maximum budget it can accumulate, making the arbiter predictable.

It has been shown in [16] that the maximum interference in a CCSP arbiter occurs when all higher priority clients start their active periods at the same time. The service latency for this case is computed according to Equation (6).

$$\Theta_i^{ccsp} = \frac{\sum_{\forall c_j \in HP_i} \sigma_j}{1 - \sum_{\forall c_j \in HP_i} \rho_j} \tag{6}$$

3.2.4 Work-conservation

All arbitration policies described in this section can be implemented either in a work-conserving or non-work-conserving manner. The definition of a work-conserving arbiter is that it is never idle as long as there is a client with pending requests. In case there is no eligible client when an arbitration decision has to be made, a non-eligible backlogged client is scheduled according to a certain *slack management policy*. For priority-based arbiters like FBSP or CCSP, the same static priorities can be used to distribute slack bandwidth. However, it is also possible to assign slack bandwidth to NRT clients that benefit from improving the average-case performance. Note that the budget of the scheduled client in a work-conserving arbiter is not deducted, since slack capacity is distributed for free. Whether or not work-conservation is used has no impact on the $\mathcal{LR}$ guarantees discussed in this article.

3.3 Real-time Memory Controllers

The bounds previously presented in this section are all expressed in terms of abstract arbitration decisions (slots) and need to be converted to clock cycles based on the worst-case execution time (WCET) of a request in the considered resource to apply in a real platform. In this work, GAMT is assumed to be connected to a state-of-the-art real-time memory controller, such as [42], [43], [44], that bound the WCET of memory transactions by fixing the memory access parameters, such as burst size and page policy, at design time. For simplicity, we assume the same constant WCET for read and write requests by taking the maximum of both. This is not a very restrictive assumption as the WCET

for read and write transactions can be made similar with negligible loss in the guaranteed bandwidth [45]. Hence, all memory requests are scheduled periodically at time intervals of fixed duration called the *scheduling interval* (SI), which is larger than or equal to the WCET of the requests.

4 Globally Arbitrated Memory Tree

This section presents our proposed Globally Arbitrated Memory Tree (GAMT) that can be configured with five different arbitration policies and supports work conservation for improving average-case performance by distributing slack. Before we present the scalable distributed architecture and operation of GAMT, we first discuss the novel concept by which we achieve scalability, global arbitration and support different arbitration policies.

To achieve scalability, we propose a *distributed architecture*, shown in Figure 4, with dedicated *Atomizer (AT)*, *Accounting* and *Priority Assignment* logic for each client and *Priority resolution* among the N clients using a tree consisting of N-1 pipelined multiplexer stages. The Atomizer first splits incoming requests into equal-sized smaller requests, called *service units*, with a service cycle duration of SC^{cc} by the Atomizer according to the fixed access size of the real-time memory controller. Ensuring that all requests have the same size reduces the complexity of the arbitration and make the timing behavior of clients independent of each others actual request sizes. The Accounting logic keeps track of the eligibility of a client to receive service and uses a *global scheduling interval* (SI) (i.e. global arbitration) of fixed duration, typically equal to SC^{cc}, between scheduling decisions. The Priority Assignment logic assigns a unique priority to the client based on the arbitration policy and whether or not the client is eligible, and the Priority resolution grants service to the client with the highest priority. Once a client is granted service, a feedback signal from the output of the Priority resolution logic updates the client's eligibility status in its Accounting logic. Also, the eligibility status of all clients is updated every scheduling interval. Note that there is *no communication* between the dedicated Accounting blocks to ensure that the complexity of the arbitration logic does not increase with the number of clients. Furthermore, the use of pipelined multiplexer stages for priority resolution breaks the critical path and enables the logic to be synthesized at higher clock frequencies. Since the arbitration decision is made by the Accounting logic at the leaves of the tree, the pipeline registers in the multiplexer tree are simple registers of width equal to the data-path width, unlike the flit-sized buffers at every arbitration stage in most existing distributed implementations. Moreover, there is no back-pressure required between the arbitration stages.

Five well-known arbiters belonging to the class of $\mathcal{LR}$ servers can currently be mimicked by configuring the Accounting and Priority Assignment logic. In TDM and RR, the responsibility of the Accounting logic is to keep track of the current slot, which essentially is the deciding factor for a client to get service. In FBSP, PBS, and CCSP the Accounting logic keeps track of the budget of the client. The priority level assigned to an eligible client by the Priority Assignment logic is based on the arbiter configuration, which guarantees a minimum bandwidth and/or a maximum latency according to the $\mathcal{LR}$ abstraction. In TDM and RR, there can only be one eligible client at a time, and hence, the highest priority is assigned to the client that is statically assigned to the slot. For FBSP, PBS and CCSP, the priority levels that are computed at design time to meet a certain bandwidth/latency requirement [8] are assigned to the eligible clients. At run time, multiple clients may be eligible and enter the tree. Note that for slack management in work-conserving mode, i.e. when none of the eligible clients are backlogged, the backlogged non-eligible clients are assigned with unique priorities that are lower than the lowest priority level assigned to an eligible client. The priority levels in the work-conserving mode depends on the slack management policy, which could be the same or different from the regular arbitration policy.

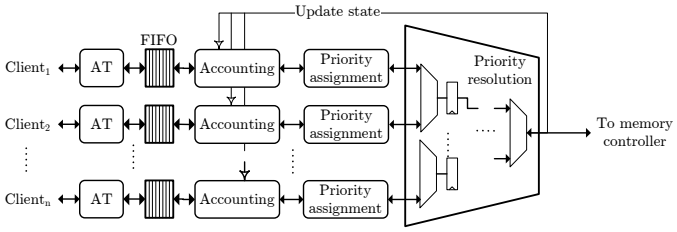


Fig. 4. High-level architecture of the Globally Arbitrated Memory Tree.

4.1 GAMT Architecture and Operation

After presenting the high-level concepts of GAMT, we proceed by discussing its distributed architecture and operation in more detail.

4.1.1 GAMT Architecture

Figure 5 shows the detailed architecture of GAMT in which the clients are at the leaves of the tree and the memory controller (MC) and DRAM at the root. The Accounting and Priority Assignment (APA) logic for each client is located in the network interface (NI) to which the client is attached. The 2-to-1 multiplexers (Mux) implementing the priority resolution are interconnected in a tree-like structure with a NI (NI_d) at the root of the tree, which interfaces with the memory controller.

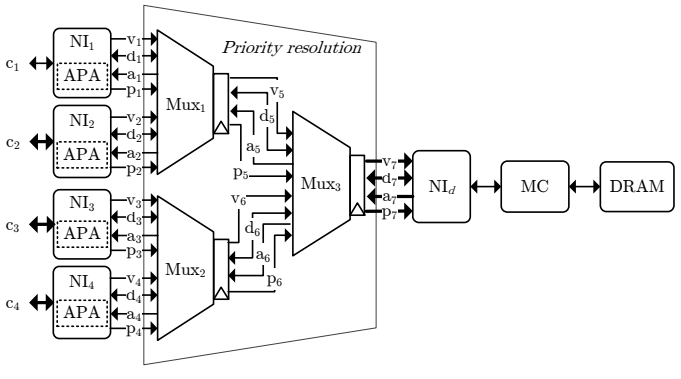


Fig. 5. Detailed architecture of GAMT along with the memory shared by four clients $c_1 - c_4$. The valid signal used to indicate a new request is denoted by v, the data/command lines *d*, priority lines *p* and the acknowledgment signal *a*.

When an eligible request is scheduled, the request valid (v) signal is asserted and the data/command (d) and the

priority (p) of the client are transmitted over the bus. When two valid inputs of the multiplexer stage arrive at the same clock cycle, the one that carries the highest priority is granted access and the other is dropped. Note that competing requests are guaranteed to arrive on the same clock cycle because of the global scheduling interval. When a service unit arrives at the root, NI_d generates an acknowledgment (a) signal that is sent back to the client, which removes the request from the head of its request queue and the current state of the Accounting logic is updated (details are presented later in Section 4.2). The dropped service units are not removed from their request buffers (no acknowledgment) and they are re-scheduled during the next SI. Note that if the request of an eligible client is dropped during a scheduling interval, the client remains eligible in the next scheduling interval. One drawback of this approach is that dropping and rescheduling requests could increase the switching activity, and hence, the power consumption.

It can be seen that the minimum SI duration (SI_{min}) must at least be equal to or greater than the total time from a request is scheduled until its acknowledgment arrives back at the source NI. Otherwise, the eligibility status of the clients in the Accounting logic will be outdated for the next service unit, resulting in incorrect functional behavior. The minimum SI hence depends on the number of multiplexer stages in the tree, which in turn depends on the number of clients in the system. For a balanced tree, this constraint is given by $SI_{min} \geq 2 \times \log_2(N)$, where N is the number of memory clients since each multiplexer stage introduces one cycle delay in both the request and response paths.

The WCET for a memory read and write request is given by $2 \times \log_2(N) + SC^{cc}$ and $\log_2(N) + SC^{cc}$, respectively. For a 16-bit IO DDR3-800 memory device, the SC^{cc} for the smallest request size of 16 Bytes is 25 clock cycles [46] assuming a *close-page policy* [8]. If we assume that GAMT runs at the same clock frequency as the memory, the minimum SI of 12 cycles for up to 64 clients is less than the SC^{cc} for the smallest request size. This ensures that requests are scheduled fast enough to ensure that there is always a request for the memory to serve and hence that the pipeline delays in GAMT are not a performance bottleneck. Moreover, with larger request sizes and faster memories, the WCET of requests in clock cycles increases making this constraint insignificant. However, note that GAMT may not be suitable for SRAMs where data can be accessed in just a few clock cycles.

For a read request, the response arrives back at the source on a pipelined response path. In this section, we assume the same clock domain and data-path width for both GAMT and the memory controller to ensure that their SIs are of the same duration. Hence, the buffer in the memory controller does not overflow as the service unit (if any) scheduled by the tree will be consumed by the memory during the same SI. However, it is possible to have different data-path widths for the memory tree and the controller and run them at different speeds by coupling the GAMT and memory controller, as proposed in [24]. During the periodic DRAM refresh operation in the memory controller, the service unit arriving at the root is dropped and rescheduled again. The refresh duration need not be an integer multiple of service cycle duration and the pending request will be served immediately in the first service cycle after the refresh operation is finished. Note that the impact of refresh needs to be taken into account for the worst-case memory bandwidth and latency computation [8].

4.1.2 Operation

Figure 6 shows an example timing behavior of the GAMT instance in Figure 5 when there are pending read/write requests to be scheduled in the FIFOs of NI_1 and NI_3 from clients c_1 (read) and c_3 (write), respectively (irrelevant signals are omitted for clarity). We consider an SI duration of 7 clock cycles and write payloads and read responses with a size of four words in this example. At the beginning of the first SI (grey vertical lines), the APA logic in NI_1 and NI_3 assert the valid signals, v_1 and v_3, and the data/command of the requests are issued on d_1 (req) and d_3 (req), respectively. We assume that client c_1 has higher priority than c_3 and their priorities are sent over p_1 and p_3, respectively (not shown in Figure 6). Since there are no pending requests in NI_2 and NI_4, the multiplexers Mux_1 and Mux_2 grant access to both requests arriving from NI_1 and NI_3, respectively. The requests arrive at Mux_3 after a delay of one clock cycle introduced by the first multiplexer stage. However, Mux_3 grants access to the request arriving from NI_1 since it has the highest priority, and the request from NI_3 is dropped. Once the root NI receives the valid signal on v_7, it sends back an acknowledgment on a_7 after one clock cycle delay as shown. The acknowledgment is sent back to the source NI_1 over a fully-pipelined response path and arrives back at NI_1 after three clock cycles. The request is then removed from the head of the FIFO in NI_1 and the APA status is updated. In the next scheduling interval, NI_3 reschedules the dropped request as it did not receive an acknowledgment. The response for the read request arrives from the memory after the WCET of the read request when the memory controller issues the response data on d_7 (resp), which is sent back to c_1 over a non-blocking response path.

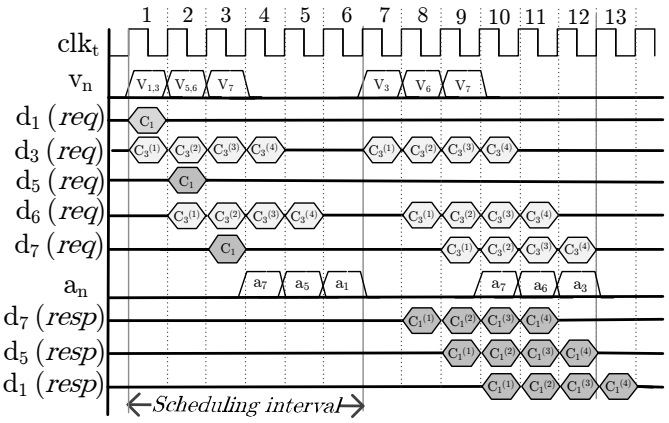


Fig. 6. Example timing diagram showing scheduling of read and write requests (req) from clients c_1 and c_3, respectively, and read responses (resp) from memory. All valid and accept signals are combined and shown together as (v_n) and (a_n), respectively.

4.2 APA Architecture and Configuration

In this section, the generic RTL architecture of the Accounting and Priority Assignment (APA) logic is first presented

and then we show how it can be configured to operate as either TDM, RR, FBSP, PBS or CCSP.

The RTL architecture of the proposed generic APA logic is shown in Figure 7. In the NI, the Atomizer splits an incoming request into smaller service units (corresponding to the fixed request size assumed by most real-time memory controllers) and the FIFO buffer stores all pending service units from a memory client. Work-conserving mode of the arbitration policy is enabled by setting the register WC to one, which enables the data valid signal (v) to be asserted whenever there is a request pending in the FIFO. Note that in work-conserving mode, the priority level of the memory client will be lower than any priority level in the non-work-conserving mode. Work conservation is disabled by setting WC to zero, which means the valid signal is asserted only when a client is eligible (has enough budget) to get service and is backlogged.

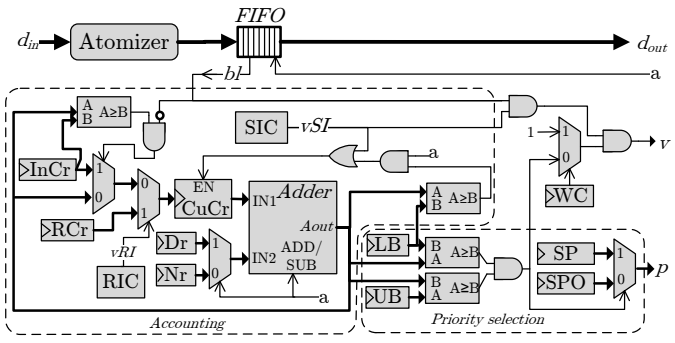


Fig. 7. Generic APA architecture that can be configured to operate as either TDM, RR, FBSP, PBS or CCSP arbitration.

Algorithm 1 shows the logical operation of the Accounting and Priority Assignment blocks[1]. In Accounting, the SI counter (SIC) asserts a valid signal, *vSI*, indicating the start of every new SI. At every SI (line 2), the value in register Current credits (CuCr), which is used to determine the eligibility status of a client, is updated according to the following different scenarios: When there are no backlogged requests, i.e. *bl* is not asserted, the value in CuCr is set to the initial budget stored in register Initial credits (InCr) using the multiplexer logic that selects InCr when the output of the Adder is greater than or equal to the initial budget (lines 3-4). The RI counter (RIC) is used by frame-based arbitration policies to replenish the budget every replenishment interval by asserting *vRI*, which causes CuCr to be reset to the value in RCr (Reset credits) (lines 5-6). When the above two conditions are not satisfied, CuCr is incremented by the value in the register Numerator (Nr) (lines 7-9). Addition is performed using a full-adder, Adder, with one of its inputs connected to CuCr and the second input to Nr when it is in addition (ADD) mode. The Adder is in the ADD mode by default and the subtract (SUB) mode is enabled when the acknowledgment (a) signal is valid. As explained in Section 3.2, accumulating a large budget in CCSP mode is not allowed when the client is not backlogged. On a valid acknowledgment signal, CuCr is decremented by the value in register Denominator (Dr) (lines 11-13). Note that the value of *Aout* returned (line 15) at the end of the procedure is passed to the Priority Assignment block and also used as the next state value of CuCr during the next iteration of the Accounting procedure.

1. For clarity in presentation, the pseudo code is split into two procedures, Accounting and Priority Assignment. Accounting is triggered by signals acknowledgment (a) and backlogged (bl) signals, whereas Priority Assignment is purely combinatorial logic.

Algorithm 1 Accounting and Priority Assignment logic

Input signals: Acknowledgment (a), Backlogged (bl)
Output signal: Priority (p)

```
1:  procedure ACCOUNTING(a, bl)
2:      if vSI then
3:          if (!bl) & (Aout ≥ InCr) then
4:              CuCr ← InCr
5:          else if vRI then
6:              CuCr ← RCr
7:          else
8:              Aout ← CuCr + Nr
9:              CuCr ← Aout
10:         end if
11:     else if (a) & (Aout ≥ LB) then
12:         Aout ← CuCr - Dr
13:         CuCr ← Aout
14:     end if
15:     return Aout
16: end procedure

17: procedure PRIORITY ASSIGNMENT(Aout)
18:     if LB ≤ Aout ≤ UB then
19:         p ← SP
20:     else
21:         p ← SPO
22:     end if
23:     return p
24: end procedure
```

The Priority Assignment logic selects a priority level stored in the register Static priority (SP) when the value of the Adder output, *Aout*, falls in between the values stored in registers Lower bound (LB) and Upper bound (UB) (lines 18-19). A different priority level with a constant offset, configured in the register SP plus offset (SPO), is selected (lines 20-21) for non-eligible clients in work-conserving mode to ensure that they are not provided service in the current SI while there are still eligible clients. The value of the offset needs to be selected according to the slack management policy, discussed in Section 3.2. When a client is scheduled in work-conserving mode, no credits are deducted from its budget, and to ensure this its current budget level is checked against the sufficient budget limit in LB before enabling CuCr (line 11).

4.3 APA Configurations

After presenting the functionality of the Accounting and Priority Assignment block, we continue by showing how to configure it to mimic the five supported arbiters in the class of $\mathcal{LR}$ servers. A summary of the different programmable registers in APA and the initial values that need to be configured to implement the supported arbitration policies are shown in Table 1. This section discusses these configurations in more detail and relate them to the arbitration policies they correspond to. Note that many other configurations of the APA logic are possible, although many of them are not

likely to be useful as they may correspond to policies that are starvation-prone or impossible to analyze. However, it may be possible to configure the logic to correspond to other well-known algorithms.

TABLE 1
APA programmable registers and their configuration for TDM/RR, FBSP/PBS and CCSP arbitration policies.

	Arbitration policy		
Register	*TDM/RR*	*FBSP/PBS*	*CCSP*
InCr	f	$f \cdot \rho$	$\sigma \cdot dr$
CuCr	0	$f \cdot \rho$	$\sigma \cdot dr$
RCr	0	$f \cdot \rho$	Not used
Nr	1	0	nr
Dr	0	1	dr
SP	Unique for each client	Unique for each client	Unique for each client
SPO	SP + Offset	SP + Offset	SP + Offset
UB	Last slot in frame	$> f \cdot \rho$	High value
LB	First slot in frame	1	$nr - dr$
SIC	SI	SI	SI
RIC	$f \cdot SI$	$f \cdot SI$	Not used

4.3.1 TDM and RR

When configured in TDM mode, the Accounting logic keeps track of the progress of the current frame in terms of number of slots and the Priority Assignment logic sets the priority of a client to the highest value available during its continuously allocated slots in the frame. In Accounting, CuCr is initialized to zero and is incremented by one every SI by configuring Nr with a value of one, which keeps track of the current slot in the frame. To identify the start of a new frame, the RIC is configured to assert *vRI* every frame by setting it to count down from $f \cdot SI$ clock cycles. This resets CuCr to zero by loading the value from RCr, which needs to be initialized to zero to restart counting the slots for the new frame. In TDM, there is no budgeting required, and hence, the value in Dr is initialized to zero so that *ack* does not affect the value in CuCr as it switches the Adder to SUB mode. Note that RR is a special case of TDM where only one slot is allocated to each client, i.e. the frame length is equal to the total number of clients.

In Priority selection, LB needs to be configured with the starting slot number of the client in the frame and UB with the ending slot number according to the continuous number of slots allocated to the client in the frame. In non-work-conserving mode, the priority level of clients configured in SP does not matter as only a single client schedules its request in a scheduling interval. For operation in work-conserving mode, we need to assign unique priority to each client in SP such that there is no conflict of priorities when SPO is selected, i.e. the priority levels of all clients in SPO must be less than in SP. For example, when the slack management policy is such that the average-case performance of bandwidth-demanding NRT clients needs to be increased, as explained in Section 3.2, SPO can be assigned priority levels in descending order starting from the client with largest bandwidth requirement. Note that InCr is not used in TDM mode, but is configured to the maximum value of f to ensure that CuCr is not updated from InCr.

4.3.2 FBSP and PBS

In FBSP and PBS modes, the Accounting logic keeps track of the current budget of a client in terms of number of slots in a frame of size f, and the Priority Assignment logic sets the priority level of the client on the priority lines as long as sufficient budget is available. At the start of every frame, CuCr is initialized with $f \cdot \rho_i = \phi_i$, which corresponds to the number of slots allocated to client c_i in a frame, i.e, the maximum budget. The current budget needs to be decremented by one whenever a service unit gets scheduled, i.e. when an acknowledgment arrives back, and hence, Dr is configured with one. To replenish the budget at the start of every new frame, the RIC enables the multiplexer logic to update the initial budget from RCr to CuCr at the end of every frame. Note that Nr is set to zero as it is not used for budget replenishment. SP needs to be configured with the priority (determined at design time to meet the latency requirements) of the client and SPO with a constant offset. LB needs to be configured with a value of one and UB with a value greater than $f \cdot \rho_i$ such that the priority in SP is selected for a number of service units equal to $f \cdot \rho_i$ in a frame. Note that InCr is not used in FBSP mode, but is set to a maximum value of $f \cdot \rho$ to avoid initialization of CuCr from InCr.

4.3.3 CCSP

In CCSP mode, the Accounting logic keeps track of the current budget level of a client based on a continuous replenishment policy and the Priority Assignment logic sets a higher priority for the client on the priority lines based on its current budget. Each client is initialized with an initial budget of $\sigma \cdot dr$ in CuCr and InCr. The budget stored in CuCr is replenished by incrementing at a rate of nr, configured in Nr, every SI and depleted by subtracting dr, configured in Dr, when an acknowledgment arrives back, where nr and dr are integers used to represent the allocated rate, $\rho = nr/dr$ [41]. In the Priority Assignment logic, SP and SPO are configured with the client's priority level and with a constant offset, respectively, just like for FBSP and PBS. LB is set to dr as $dr - nr$ is the minimum budget required to select SP and that *Aout* is CuCr+nr at the beginning of every SI, which determines the priority level. UB needs to be set to a sufficiently large value such that it is larger than the maximum budget that can ever built up, which is bounded in [16].

5 Mixed Arbitration Policies

Having presented the architecture and operation of GAMT and shown how to configure the Accounting and Priority Assignment (APA) logic, this section continues by introducing the concept of mixed arbitration policies. The key idea is that the APA logic allows any of the five arbitration policies to be selected *per client*, as opposed to jointly for all clients, greatly increasing the arbitration options *without any additional hardware cost.*

Examples of interesting mixed arbitration policies currently supported by GAMT are combinations of either TDM or Round Robin with a priority-based policy like CCSP, PBS or FBSP. These policies are largely motivated by the latency requirements that they can satisfy. Priority-based arbiters result in high-priority clients getting very low latencies at

the expense of latencies of low-priority clients getting very high [8] (see also Equations (5) and (6)). In contrast, Round Robin arbitration treats all clients equally and gives the same latency to all of them. Intuitively, a mixed policy is useful in cases where the latency requirements of the clients are somewhere in the middle and neither policy can efficiently satisfy their requirements. In that case, some clients can be served in a Round Robin manner, getting equal treatment, while others use priorities to efficiently cater to more diverse latency requirements. Replacing the Round Robin policy with non-work-conserving TDM in this scenario gives freedom to allocate different bandwidth to each client, but more importantly, non-work-conserving TDM provides temporal isolation between clients and ensure that both the actual and the guaranteed bandwidth and latency are independent of others. This is a useful property for verification e.g. in the context of certification for safety-critical systems, such as airplanes [47], [48]. In the following section, we proceed by analyzing one of these interesting policies, namely the combination of TDM and FBSP.

5.1 TDM+FBSP Arbitration

This section proposes a novel arbitration policy where some clients use non-work-conserving TDM arbitration to achieve temporal isolation, while others use work-conserving FBSP arbitration to reduce their average latency and satisfy diverse requirements in terms of bandwidth and latency. This is a compelling and intuitive combination of policies, since they are both based on periodically repeating frames, making their combined behavior easy to reason about. Throughout this article, we will refer to this mixed arbitration policy as TDM+FBSP.

To use this policy, the clients are individually configured according to the TDM or FBSP column in Table 1, respectively. However, to achieve temporal isolation, the TDM clients must be configured to have higher priority than the FBSP clients. This configuration guarantees that a TDM client always has the highest priority in the system during its allocated slots. As an example, consider two TDM clients, c_1 and c_2, allocated to the first three slots in a frame of size five with the remaining two slots reserved for FBSP clients c_3 and c_4 with rates rates $\rho_i = 1/5$ each. Table 2 shows the initial values that need to be set to the different programmable registers for the four clients. The TDM clients are assigned higher priority (SP) than FBSP clients and LB and UB parameters are used to allocate them consecutively in the beginning of the frame. We have used a constant offset value of 4 to configure SPO, which is the minimum value that makes sure all clients have unique priorities at all times.

TABLE 2
Example initialization of the programmable registers for TDM+FBSP.

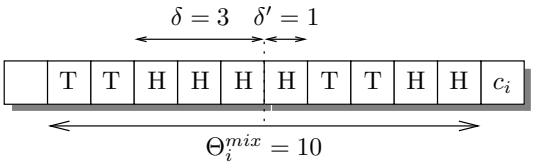
Register	c_1	c_2	c_3	c_4
CuCr	0	0	1	1
RCr	0	0	1	1
Nr	1	1	0	0
Dr	0	0	1	1
SP	1	2	3	4
SPO	5	6	7	8
UB	1	3	2	2
LB	1	2	1	1

5.2 Analysis of TDM+FBSP

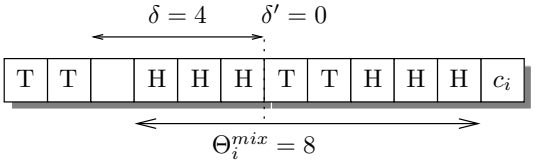
After defining TDM+FBSP arbitration, we continue by deriving its worst-case analysis according to the $\mathcal{LR}$ framework. Since TDM clients are guaranteed to always have the highest priority, nothing changes in their analysis and they can be analyzed independently from the FBSP clients (and each other). This means that the bounds on allocated rate and service latency from Equations (3) and (4) are still valid for the TDM clients under TDM+FBSP arbitration.

While the high priority of TDM clients make them oblivious of their FBSP counterparts, the reverse does not hold. This is because all FBSP clients have lower priorities than the TDM clients and must wait for them during their allocated slots. The most general way to incorporate this into the worst-case analysis is by adding all TDM clients to the set of higher priority clients for all FBSP clients and continue to use Equations (3) and (5) to compute the allocated rate and service latency, respectively. This bound on service latency may intuitively seem pessimistic, since the FBSP analysis assumes that all higher priority clients interfere twice with the client under analysis, first at the end of a frame and then in the beginning of the following frame when budgets have been replenished. While this is possible for FBSP clients where slots are dynamically assigned to clients based on their priority and availability of pending requests, it may not be obvious that it can happen for TDM clients that are statically assigned to slots. Figure 8a illustrates that this is indeed possible by considering a TDM+FBSP arbiter with frame size $f = 6$, repeated twice. The frame in the figure has $\phi_T = 2$ total slots allocated to TDM clients and $\phi_H = 3$ total allocated slots to FBSP clients with higher priority than FBSP client c_i under analysis. In the worst case, client c_i starts a busy period just before the allocated TDM slots in the first frame in the figure. It then first suffers 2 slots of interference from the TDM clients, followed 3 slots of interference from higher priority FBSP clients before the end of the frame. Once the second frame starts, the budgets of higher priority clients have been fully replenished, allowing them to interfere during another 3 slots. Before this interference ends, we are back at the statically assigned TDM slots, enabling the TDM clients to interfere a second time before client c_i receives service. This example results in a maximum service latency $\Theta_{c_i}^{mix} = 10$ slots.

A reduced bound on service latency can be obtained by constraining the slot assignment for the TDM clients to be consecutive in the beginning (or end) of the frame, as shown in Figure 8b. This assignment prevents the TDM clients from interfering twice by ensuring that the FBSP clients cannot continuously interfere between the TDM allocations in the two frames, as shown in Lemma 1. Note that concentrating the TDM slots of all TDM clients in the beginning or end of the frame does not impact their service latency in any way, since the bound in Equation (4) is independent of where the continuous allocation of each client is in the TDM table. Based on Lemma 1, Theorem 1 then proceeds by showing that our proposed TDM+FBSP arbitration policy belongs to the class of $\mathcal{LR}$ servers.



(a) The TDM clients interfere twice in the worst-case scenario with unconstrained slot assignments.



(b) The TDM clients only interfere once in the worst-case scenario when assigned slots in the beginning of the frame.

Fig. 8. TDM+FBSP arbiter with frame size $f = 6$, repeated twice, and $\phi_H = 3$ allocated slots to higher priority FBSP clients and $\phi_T = 2$ to TDM clients.

Lemma 1. *TDM clients only interfere once with FBSP clients in the worst-case scenario under TDM+FBSP arbitration if their slots are assigned consecutively in the beginning or end of the frame.*

Proof. We prove the lemma by contradiction by assuming that the TDM clients $\forall c_j \in TDM$ interfere twice with an FBSP client c_i under analysis. This means that the higher priority FBSP clients are able to continuously interfere between two sets of continuous slots assigned to TDM clients in consecutive frames.

We define δ as the distance from the end of the collective TDM allocation until the end of the frame. Similarly, we define δ' as the distance from the start of the frame until the start of the TDM allocation. These definitions are illustrated in Figure 8. To continuously interfere, Equation (7) must hold to allow the set of higher priority FBSP clients, HP_i, to bridge the gap between two sets of TDM allocations.

$$\sum_{\forall c_j \in HP_i} \phi_j \geq \max(\delta, \delta') \tag{7}$$

Since the allocated TDM slots are continuously assigned either in the beginning or end of the frame, we maximize δ or δ', respectively, and get

$$\max(\delta, \delta') = f - \sum_{\forall c_j \in TDM} \phi_j$$

Combining these two inequalities and rearranging gives us

$$\sum_{\forall c_j \in HP_i} \phi_j + \sum_{\forall c_j \in TDM} \phi_j \geq f$$

However, this means that the slots allocated to the higher priority FBSP clients and the TDM clients must be greater than or equal to the frame size, which is not possible since at least one slot must be assigned to the FBSP client c_i under analysis. This contradicts the initial assumption that the TDM clients interfere twice, which concludes the proof. □

Theorem 1. *TDM+FBSP arbitration belongs to the class of $\mathcal{LR}$ servers. Assuming all TDM clients have continuous slot assignments in the beginning of the frame, the service latency of the clients is given by Equations* (4) *and Equation* (8) *for TDM and FBSP clients, respectively.*

$$\Theta_i^{mix} = 2 \cdot \sum_{\forall c_j \in HP_i} \phi_j + \sum_{\forall c_j \in TDM} \phi_j \tag{8}$$

Proof. Since TDM clients have higher priority than FBSP clients, nothing changes in their analysis and existing service latency results hence hold without modification. To prove the theorem, it hence suffices to show that Equation (9) holds for an FBSP client c_i under analysis during a busy period of length l. We are only interested in values of $l > 2 \cdot \sum_{\forall c_j \in HP_i} \phi_j + \sum_{\forall c_j \in TDM} \phi_j$, since these are the only values for which $w_i^l > 0$.

$$w_i^l \geq \max(0, \rho_i \cdot (l - 2 \cdot \sum_{\forall c_j \in HP_i} \phi_j + \sum_{\forall c_j \in TDM} \phi_j)) \tag{9}$$

The worst-case interference from TDM clients equals $\sum_{\forall c_j \in TDM} \phi_j$ by Lemma 1, since they can only interfere once, and $2 \cdot \sum_{\forall c_j \in HP_i} \phi_j$ for FBSP clients by Equation (5). Client c_i has hence received maximum interference from other clients when $l = \sum_{\forall c_j \in TDM} \phi_j + 2 \cdot \sum_{\forall c_j \in HP_i} \phi_j$. This implies that it has already received maximum interference from both TDM clients and higher priority FBSP clients in the current frame and that it receives its ϕ_i consecutively allocated slots without interruption during $l \in [\Theta_i^{mix} + 1, \Theta_i^{mix} + \phi_i]$ (see e.g. second frame in Figure 8b). After this, lower priority FBSP clients are served, potentially followed by unallocated slots until the end of the frame. The following frames are identical to this frame in the worst case, since it features maximum interference from all clients. Client c_i hence receives ϕ_i of service for every f slots from this point onwards.

The provided service to c_i is minimum just before the allocated service is provided in a frame. It is hence sufficient to show that the $\mathcal{LR}$ guarantee is satisfied for all time instants $l' = \Theta_i^{mix} + n \cdot f$, where $n \in \mathbb{N}$. At these time instants, Equation (9) becomes

$$w_i^{l'} \geq \rho_i \cdot (\Theta_i^{mix} + n \cdot f - \Theta_i^{mix}) = \rho_i \cdot n \cdot f \tag{10}$$

Knowing that the provided service equals $w_i^{l'} = n \cdot \phi_i$ at these points, we arrive at

$$w_i^{l'} = n \cdot \phi_i \geq \rho_i \cdot n \cdot f \tag{11}$$

The proof is concluded by exploiting that $\rho_i = \phi_i / f$ for a frame-based arbiter by Equation (3), which implies that the inequality holds for all l'. This shows that the $\mathcal{LR}$ guarantee holds for an FBSP client c_i with $\Theta_i^{mix} = 2 \cdot \sum_{\forall c_j \in HP_i} \phi_j + \sum_{\forall c_j \in TDM} \phi_j$. □

The proof that TDM+FBSP belongs to the class of $\mathcal{LR}$ servers is a very useful theoretical result that enables the proposed mixed arbitration policy to be used for WCET or throughput estimation of real-time applications using several well-known performance analysis frameworks, as previously explained in Section 3.1.

6 Experiments

In this section, we present the functionality verification of GAMT and evaluation of its real-time guarantees in mixed arbitration mode. Also, we show the performance comparison of GAMT with respect to centralized implementations.

6.1 Experimental Setup

The experimental setup consists of the RTL implementation of GAMT [49] and centralized implementations of two different arbitration policies, TDM [50] FBSP [41], and CCSP [8], with a 32-bit data-path. We used Cadence Encounter RTL compiler and the 40 nm nominal Vt CMOS standard cell technology library from TSMC with the worst-case process corner for logic synthesis to determine the power and area usage and the maximum synthesizable frequency.

6.2 Verification of Functional Correctness

We ensured the functional correctness of GAMT by comparing the scheduling decisions made by the FPGA implementation of GAMT (with 16 clients) at every scheduling interval with C++ reference models of centralized implementations of TDM, FBSP, and CCSP. Note that the other supported arbitration policies are special cases of these three arbiters and do not require additional verification. We used synthetic traffic generators for the clients to generate random traffic to cover both backlogged and non-backlogged conditions and verified the functionality (for several thousands of scheduling decisions) in both work-conserving and non-work-conserving modes of all the three arbitration policies. We found that all scheduling decisions made by both GAMT and the centralized implementations were the same, which *strongly suggests that GAMT correctly implements the different arbitration policies.* Since all decisions in GAMT are made identically to the centralized reference implementations, the timing analyses of the original arbiters can be used for GAMT as well with an only addition of the constant propagation delay in GAMT.

6.3 Real-time performance evaluation

To evaluate the conservativeness of the analytically computed latency bounds, we performed experiments using the FPGA implementation of GAMT with eight memory clients configured using TDM arbitration policy and eight clients using FBSP. We selected a frame size of 16, with each TDM client allocated to one slot and a rate of 1/16 allocated to each FBSP client. As required by our analysis in Section 5, the slots of the TDM clients are all consecutively assigned in the beginning of the TDM frame. Unique priorities are furthermore assigned to all the clients with the highest priority to the TDM clients. To measure the service latency introduced by the arbitration in GAMT alone (excluding self-interference), we configured the traffic generators to generate traffic with a single outstanding request.

Figure 9 shows the average latencies of all the TDM (1-8) and FBSP (9-16) clients computed over 1500 memory requests. The solid vertical line above the average latency indicates the measured maximum latency and the (red) horizontal line the worst-case latency bound of the different clients. We computed the reduced worst-case latency bound (in service cycles) using Equation (2), as explained in Section 3.1. Note that the service latencies for TDM and FBSP clients are based on Equations (4) and (8), respectively. The computed bound in service cycles is converted to clock cycles by multiplying with the service cycle length and the pipeline delay of four cycles (corresponding to four stages) is then added to it. We can see that the maximum observed latencies of all requests from all clients are smaller than the derived bounds, *suggesting the bounds are conservative*. Also, both the average and maximum latency of FBSP clients increase with decreasing priority, which shows that mixing arbitration policies provides us with sufficient flexibility to shape the latency distributions according to client requirements.

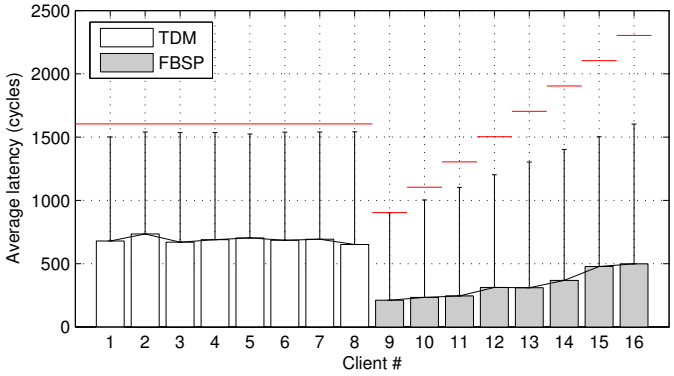


Fig. 9. Average service latency of requests of all TDM and FBSP clients in a system consisting of 16 clients. The solid vertical line indicates the measured maximum latency for each client and the (red) horizontal line shows the worst-case latency bound.

To verify whether the clients with TDM arbitration policy are temporally isolated from other clients, we repeated the same experiment with the FBSP clients turned off. We then compared the observed latencies of all the requests of all TDM clients with and without the presence of FBSP clients. We found that all requests of the TDM clients have identical observed latency both when run independently and in the presence of other clients. *This evidence strongly suggests that there is complete timing isolation for TDM clients*, i.e. that their execution is not affected by other clients even by a single clock cycle.

Finally, we evaluate the gain in average-case performance due to the efficient slack utilization in the mixed arbitration mode of GAMT, i.e. possibility of exploiting slack across multiple arbitration policies. With the same number of TDM and FBSP clients as before, and with maximum two outstanding requests[2], experiments were performed with the FBSP clients configured both in work conserving and non-work-conserving modes. Our experimental results suggest that the FBSP clients in work-conserving mode has *over 32% reduction in average-latency* than the TDM clients compared to non-work-conserving mode.

2. The rate allocated to all the clients is 1/16, and hence, a second request is required (in the current frame) to utilize the slack (if any) before the credits are replenished.

6.4 Comparison with Centralized Implementations

We synthesized the design of GAMT and the centralized implementations of TDM, FBSP and CCSP for different number of clients, i.e. 4, 8, 16, 32 and 64 to determine the maximum synthesizable frequency. For TDM and FBSP, we have used a frame size such that each client gets four slots and for CCSP, we used registers with 16-bit precision for the configuration registers (Numerator, Denominator, and Current Credits). Table 3 shows the area, power and maximum clock frequency of the GAMT and the centralized implementations of TDM, FBSP and CCSP. In general, it can be seen that the maximum clock frequency, f_{max}, of centralized arbiters do not scale with the number of clients. With 64 clients, GAMT can be run up to a clock frequency of 1.2 GHz, whereas the centralized implementations are limited to around 0.3 GHz.

In general, the area and power consumption of all different designs increase linearly with the number of clients due to the additional logic added. The area and power of centralized TDM increases and the f_{max} scales down with increasing number of clients due to large look-up-table size and complexity in the priority resolution in work-conserving mode. For CCSP, the area and power consumption increases significantly with the number of clients due to its complex accounting logic. Also, the f_{max} of CCSP scales down with increasing number of clients due to the critical path in the priority resolution. Although centralized FBSP has similar priority resolution as CCSP, it has lower area and power because of simpler accounting logic. However, the f_{max} of FBSP with 64 clients is lower than that of CCSP due to a longer critical path in priority resolution. On the other hand, GAMT has better scalability in f_{max} with the number of clients, since its critical path in the APA logic remains constant irrespectively of the number of clients as it is simply duplicated. However, it is worthwhile to note that GAMT consumes more power compared to the centralized implementations in most cases. This is primarily due to the addition of extra priority lines on the bus and the dedicated APA logic for each client. One limitation of GAMT is that it can support only TDM with continuous slot allocation strategy, whereas the centralized implementation of TDM using a Look-up-Table (LUT) can support distributed allocations [8].

To efficiently compare the centralized designs and GAMT in terms of frequency, area and power consumption, we define two cost-efficiency metrics, *bandwidth/area* and *bandwidth/power (bits/Watts)*. Bandwidth is computed by multiplying data-path width (in Bytes) with the clock frequency (f_{max}). Figures 10 & 11 show the ratio of bandwidth (Bytes/s) to area usage (mm^2) and bandwidth (Bytes/s) to power consumption (mW), respectively, of centralized implementations of TDM, FBSP and CCSP normalized to GAMT. It can be seen that for all configurations of clients, GAMT has over 51% and 37% performance gain in terms of area and power consumption, respectively, compared to traditional centralized implementations. Hence, *we can conclude that GAMT is suitable when there are a large number of memory clients in the system that requires the arbiter to be clocked at higher speed or when the platform requires different arbiters for different sets of applications.*

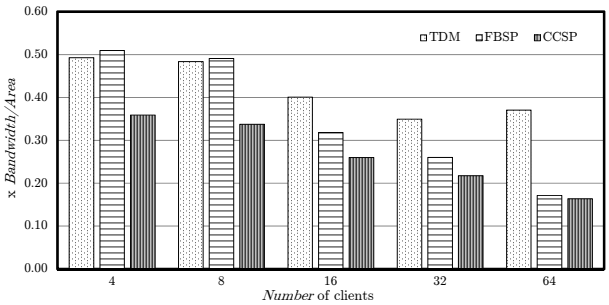


Fig. 10. Bandwidth/Area performance of centralized TDM, FBSP and CCSP arbiter implementations normalized to GAMT.

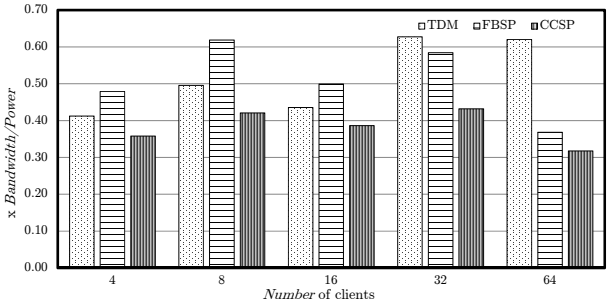


Fig. 11. Bandwidth/Power performance of centralized TDM, FBSP and CCSP arbiter implementations normalized to GAMT.

7 Conclusions

The increasing number of memory clients in mixed-time-criticality systems requires a scalable memory interconnect supporting multiple arbitration policies. However, existing centralized architectures are not scalable in terms of clock frequency with the increasing number of clients and locally arbitrated distributed implementations suffer from long latencies and large area and power usage due to the buffers in the local arbitration stages. On the other hand, existing distributed memory interconnects using global arbitration are limited to TDM, which is not suitable for clients with diverse bandwidth and latency requirements.

This article addresses this problem by proposing Globally Arbitrated Memory Tree, GAMT, with a distributed architecture that can be configured with five well-known arbitration policies (two of which are special cases of the other). We introduce the novel concept of mixed arbitration policies, where the choice of arbiter is done per client instead of jointly for all clients, to further increase arbitration flexibility without impacting cost. A novel mixed arbitration policy targeting mixed-time-criticality systems that combines non-work-conserving Time-Division Multiplexing to achieve temporal isolation with work-conserving Frame-Based Static-Priority to address diversity and reduce average latency was then proposed. A worst-case analysis was then performed of the mixed policy in the context of the latency-rate framework. Experimental results suggest that GAMT behaves identically to the arbiters it is configured to emulate and that bounds for the novel mixed policy are conservative. Moreover, a reduction of over 32% in average latency was achieved for all FBSP clients due to the efficient utilization of slack in the mixed arbitration mode TDM+FBSP. Synthesis results using a 40 nm process shows that GAMT runs four times faster with 64 clients than the

TABLE 3
Area usage and power consumption at maximum clock frequency (f_{max}) of GAMT and centralized implementations of TDM, FBSP and CCSP.

	Area (mm^2)				Power (mW)				f_{max} (MHz)			
# Clients	*TDM*	*FBSP*	*CCSP*	*GAMT*	*TDM*	*FBSP*	*CCSP*	*GAMT*	*TDM*	*FBSP*	*CCSP*	*GAMT*
4	0.02	0.02	0.02	0.02	5.19	4.47	5.35	4.55	588	588	526	1250
8	0.03	0.03	0.04	0.03	7.88	6.32	8.07	9.77	500	500	435	1250
16	0.06	0.06	0.08	0.07	16.13	11.56	14.94	20.20	435	357	357	1250
32	0.11	0.14	0.17	0.14	17.46	18.75	25.36	41.07	333	333	333	1250
64	0.20	0.37	0.42	0.28	35.60	49.98	63.18	82.81	333	278	303	1250

corresponding centralized architectures and have over 51% and 37% savings in terms of area and power consumption for a given bandwidth, respectively.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology), co-financed by ERDF (European Regional Development Fund) under PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2) and 621353 (DEWI) and EU FP7 288008 T-CREST, 318763 JUNIPER, 288248 Flextiles, CA505 BENEFIC, CA703 OpenES.

REFERENCES

[1] "International Technology Roadmap for Semiconductors (ITRS)," 2013, http://www.itrs.net.

[2] P. Kollig, C. Osborne, and T. Henriksson, "Heterogeneous multi-core platform for consumer multimedia applications," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. European Design and Automation Association, 2009, pp. 1254–1259.

[3] C. H. K. van Berkel, "Multi-core for mobile phones," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. European Design and Automation Association, 2009, pp. 1260–1265.

[4] P. van der Wolf and J. Geuzebroek, "SoC infrastructures for predictable system integration," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1–6.

[5] L. Steffens, M. Agarwal, and P. Wolf, "Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach," in *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, 2008, pp. 255–265.

[6] B. Akesson, A. Molnos, A. Hansson, J. A. Angelo, and K. Goossens, "Composability and predictability for independent application development, verification, and execution," in *Multiprocessor System-on-Chip — Hardware Design and Tool Integration*. Springer, 2010, ch. 2.

[7] "JEDEC Solid State Technology Association," http://www.jedec.com, JEDEC Solid State Technology Association, 2015.

[8] B. Akesson and K. Goossens, *Memory Controllers for Real-Time Embedded Systems*, first edition ed., ser. Embedded Systems Series. Springer, 2011.

[9] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[10] L. S. 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*. IEEE, 2015, pp. 47–56.

[11] H. Kim, D. Broman, E. Lee, M. Zimmer, A. Shrivastava, J. Oh *et al.*, "A predictable and command-level priority-based DRAM controller for mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. IEEE, 2015, pp. 317–326.

[12] L. Ecco, S. Tobuschat, S. Saidi, and R. Ernst, "A mixed critical memory controller using bank privatization and fixed priority scheduling," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*. IEEE, 2014, pp. 1–10.

[13] E. S. Shin, I. V. J. Mooney, and G. F. Riley, "Round-robin arbiter design and generation," in *Proceedings of the 15th International Symposium on System Synthesis*, ser. ISSS '02. ACM, 2002, pp. 243–248.

[14] G. Dimitrakopoulos, C. Kachris, and E. Kalligeros, "Scalable Arbiters and Multiplexers for On-FGPA Interconnection Networks," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 90–96.

[15] M. Weber, "Arbiters: design ideas and coding styles," in *Synopsys Users Group (SNUG)*, 2001.

[16] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on*, Aug 2008, pp. 3–14.

[17] J. Garside and N. Audsley, "Prefetching across a shared memory tree within a Network-on-Chip architecture," in *System on Chip (SoC), 2013 International Symposium on*, Oct 2013, pp. 1–4.

[18] J. H. Rutgers, M. J. G. Bekooij, and G. J. M. Smit, "Evaluation of a Connectionless NoC for a Real-Time Distributed Shared Memory Many-Core System," in *Proceedings of the 2012 15th Euromicro Conference on Digital System Design*, ser. DSD '12, 2012, pp. 727–730.

[19] A. Hansson, M. Coenen, and K. Goossens, "Channel trees: Reducing latency by sharing time slots in time-multiplexed networks on chip," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, Sept 2007, pp. 149–154.

[20] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A Time-Predictable Memory Network-on-Chip," in *14th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2014, pp. 53–62.

[21] A. Rahimi, I. Loi, M. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for Shared-L1 processor clusters," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[22] K. Goossens, O. P. Gangwal, J. Roever, and A. P. Niranjan, "Interconnect and memory organization in SOCs for advanced set-top boxes and TV — Evolution, analysis, and trends," in *Interconnect-Centric Design for Advanced SoC and NoC*. Kluwer, 2004, ch. 15.

[23] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, April 2008, pp. 161–170.

[24] M. D. Gomony, B. Akesson, and K. Goossens, "Coupling TDM NoC and DRAM Controller for Cost and Performance Optimization of Real-Time Systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6.

[25] A. Kostrzewa, S. Saidi, and R. Ernst, "Dynamic control for mixed-critical networks-on-chip," pp. 317–326, Dec 2015.

[26] M. D. Gomony, J. Garside, B. Akesson, N. Audsley, and K. Goossens, "A Generic, Scalable and Globally Arbitrated Memory Tree for Shared DRAM Access in Real-time Systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 193–198.

[27] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE Trans. Netw*, vol. 6, no. 5, 1998.

[28] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 21, no. 2, pp. 151–166, 1999.
[29] S. Stuijk, M. Geilen, and T. Basten, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," *Computers, IEEE Transactions on*, vol. 57, no. 10, pp. 1331–1345, 2008.
[30] J. P. Vink, K. Van Berkel, and P. Van Der Wolf, "Performance analysis of soc architectures based on latency-rate servers," in *Design, Automation and Test in Europe, 2008. DATE'08*. IEEE, 2008, pp. 200–205.
[31] R. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, 1991.
[32] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*. CRC press, 2009.
[33] V. Rodrigues, B. Akesson, M. Florido, S. M. de Sousa, J. P. Pedroso, and P. Vasconcelos, "Certifying execution time in multicores," *Science of Computer Programming*, vol. 111, pp. 505–534, 2015.
[34] A. Nelson, K. Goossens, and B. Akesson, "Dataflow formalisation of real-time streaming applications on a Composable and Predictable Multi-Processor SOC ," *Journal of Systems Architecture*, 2015.
[35] M. H. Wiggers, M. J. Bekooij, and G. J. Smit, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proceedingsof the 10th international workshop on Software & compilers for embedded systems*. ACM, 2007, pp. 11–22.
[36] H. Shah, A. Knoll, and B. Akesson, "Bounding sdram interference: detailed analysis vs. latency-rate analysis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE, 2013, pp. 308–313.
[37] M. Steine, M. Bekooij, and M. Wiggers, "A Priority-Based Budget Scheduler with Conservative Dataflow Model," in *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, 2009, pp. 37–44.
[38] J. Staschulat and M. Bekooij, "Dataflow models for shared memory access latency analysis," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009, pp. 275–284.
[39] B. Akesson, A. Minaeva, P. Sucha, A. Nelson, and Z. Hanzalek, "An Efficient Configuration Methodology for Time-Division Multiplexed Single Resources," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2015.
[40] A. Minaeva, P. Sucha, B. Akesson, and Z. Hanzalek, "Scalable and efficient configuration of time-division multiplexed resources," *Journal of Systems and Software*, vol. 113, pp. 44–58, 2016.
[41] B. Akesson, L. Steffens, and K. Goossens, "Efficient service allocation in hardware using credit-controlled static-priority arbitration," in *Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA'09. 15th IEEE International Conference on*. IEEE, 2009, pp. 59–68.
[42] B. Akesson and K. Goossens, "Architectures and modeling of predictable memory controllers for improved system integration," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1–6.
[43] Y. Li, B. Akesson, and K. Goossens, "Architecture and analysis of a dynamically-scheduled real-time memory controller," *Real-Time Systems*, pp. 1–55, 2015.
[44] M. Paolieri, E. Quiñones, and F. J. Cazorla, "Timing Effects of DDR Memory Systems in Hard Real-time Multicore Architectures: Issues and Solutions," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1s, pp. 64:1–64:26, Mar. 2013.
[45] S. Goossens, K. Chandrasekar, B. Akesson, and K. Goossens, "Power/performance trade-offs in real-time sdram command scheduling," *Computers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.
[46] "MICRON DDR3 SDRAM Specification," 2006, Rev. M 08/14 EN.
[47] I. RTCA, *RTCA/DO-178C*. U.S. Dept. of Transportation, Federal Aviation Administration, 2012.
[48] Certification authorities in North and South America, Europe, and Asia, *Certification authorities software team (CAST), position paper (CAST-32) multi-core processors*, 2014.
[49] J. Garside, https://github.com/RTSYork/GAMT/.
[50] S. Goossens, J. Kuijsten, B. Akesson, and K. Goossens, "A reconfigurable real-time SDRAM controller for mixed time-criticality systems," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, Sept 2013, pp. 1–10.

Manil Dev Gomony received his PhD and Masters in Electrical Engineering from Eindhoven University of Technology in 2015 and Linköping University in 2010, respectively. Currently, he is a Researcher at the Bell Laboratories of Alcatel-Lucent. His research interests are confined to different aspects of embedded systems such as application-specific processors, memory subsystems, scheduling algorithms, and embedded software.

Jamie Garside received his PhD in Real-Time Systems from the University of York in 2016, and a MEng in Computer Systems and Software Engineering in 2011. His research interests are predictable memory interconnects and prefetching within real-time systems.

Benny Akesson received his MSc degree at Lund Institute of Technology, Sweden in 2005 and a PhD from Eindhoven University of Technology, the Netherlands in 2010. Since then, he has been employed as a Researcher at Eindhoven University of Technology, Czech Technical University in Prague, and CISTER/INESC TEC Research Unit in Porto. Currently, he is working as a Research Fellow at TNO-ESI. His research interests include memory controller architectures, real-time scheduling, performance modeling, and performance virtualization. He has published more than 50 peer-reviewed conference papers and journal articles, as well as two books about memory controllers for real-time embedded systems.

Neil Audsley received a BSc (1984) and PhD (1993) from the Department of Computer Science at the University of York, UK. In 2013, he received a Personal Chair from the University of York, where he leads a team researching Real-Time Embedded Systems. Specific areas of research include high-performance real-time systems, real-time operating systems, real-time architectures, scheduling, and model-driven development. Professor Audsley's research has been funded by both national (EPSRC) and European (EU) grants. He has published upwards of 150 publications in peer reviewed journals, conferences and books.

Kees Goossens received his PhD in Computer Science from the University of Edinburgh in 1993. He worked for Philips/NXP Research from 1995 to 2010 on networks-on-chips for consumer electronics, where real-time performance, predictability, and costs are major constraints. He was part-time professor at Delft University from 2007 to 2010, and is now full professor at the Eindhoven University of Technology, where his research focuses on composable (virtualized), predictable (real-time), low-power embedded systems, supporting multiple models of computation. He published 4 books, 100+ papers, and 24 patents.