A Generic, Scalable and Globally Arbitrated Memory Tree for Shared DRAM Access in Real-Time Systems

Manil Dev Gomony*, Jamie Garside‡, Benny Akesson†, Neil Audsley‡, and Kees Goossens*
*Eindhoven University of Technology, The Netherlands
†Czech Technical University in Prague, Czech Republic
‡University of York, United Kingdom

***Abstract*—Predictable arbitration policies, such as Time Division Multiplexing (TDM) and Round-Robin (RR), are used to provide firm real-time guarantees to clients sharing a single memory resource (DRAM) between the multiple memory clients in multi-core real-time systems. Traditional centralized implementations of predictable arbitration policies in a shared memory bus or interconnect are not scalable in terms of the number of clients. On the other hand, existing distributed memory interconnects are either globally arbitrated, which do not offer diverse service according to the heterogeneous client requirements, or locally arbitrated, which suffers from larger area, power and latency overhead. Moreover, selecting the right arbitration policy according to the diverse and dynamic client requirements in reusable platforms requires a generic re-configurable architecture supporting different arbitration policies.**

The main contributions in this paper are: (1) We propose a novel generic, scalable and globally arbitrated memory tree (GSMT) architecture for distributed implementation of several predictable arbitration policies. (2) We present an RTL-level implementation of Accounting and Priority assignment (APA) logic of GSMT that can be configured with five different arbitration policies typically used for shared memory access in real-time systems. (3) We compare the performance of GSMT with different centralized implementations by synthesizing the designs in a 40 nm process. Our experiments show that with 64 clients GSMT can run up to four times faster than traditional architectures and have over 51% and 37% reduction in area and power consumption, respectively.

I. Introduction

In heterogeneous multi-core platforms for real-time systems, Dynamic Random Access Memory (DRAM) is typically used as a shared resource to reduce cost and enable communication between memory clients, i.e. tasks of an application running on multiple cores [1], [2]. The number of memory clients is ever increasing with more applications being integrated in such platforms. Moreover, real-time guarantees on memory performance in terms of bandwidth and/or latency need to be provided to these clients to meet the firm real-time requirements of the applications, which may be quite diverse [3]. Current real-time memory controllers [4]–[8] provide real-time guarantees to the clients and use a predictable arbitration policy in the interconnect in front of them to multiplex requests from different clients. Several predictable arbitration policies, such as Time Division Multiplexing (TDM), Round Robin [9], Frame-Based Static Priority (FBSP) [10], Priority-Based Scheduler (PBS) [11], [7] and Credit-Controlled Static-Priority (CCSP) [12], has been proposed accordingly to suit diverse client requirements. Current interconnect architectures implementing predictable arbitration policies can be classified into three categories: *centralized* (with local arbitration) and *distributed with local* and *global* arbitration, respectively.

Centralized architectures are easy to implement as the arbitration decision is made locally at a central location using a single global schedule for all clients. However, they suffer from poor scalability with respect to the number of clients. This is because in traditional centralized implementations of priority-based arbitration policies, such as FBSP and CCSP, the priorities of all clients are compared using combinatorial logic consisting of a tree of multiplexers [13]. The main drawback of this approach is that the critical path of the multiplexer tree increases with the number of clients, which reduces the maximum clock frequency at which the logic can be synthesized [10], [14]. Moreover, the implementation of slack management in any of the predictable arbitration policies requires implicit priority resolution as one client needs to be selected out of many based on the *slack management policy*, which again is not scalable using centralized architectures [15].

Distributed memory interconnects with global and local arbitrations were proposed to address the scalability issue in shared DRAM access. However, current distributed interconnects with local arbitration [16], [17] suffer from poor performance in terms of latency, area usage and power consumption due to the buffering of memory requests at every arbitration stage. On the other hand, distributed interconnects with global TDM arbitration [18]–[20] are not suitable in systems where the clients have diverse bandwidth and/or latency requirements. To summarize, currently there is no *scalable* distributed architecture with *global arbitration* implementing different predictable arbitration policies and no *generic* configurable architecture that can be configured with an arbitration policy according to the diverse client requirements.

The main contributions in this paper are: (1) We propose a novel generic, scalable and globally arbitrated memory tree (GSMT) (shown in Figure 1) for distributed implementation of several predictable arbitration policies. (2) We present a generic RTL-level implementation of Accounting and Priority assignment (APA) logic of GSMT that can be configured to operate as one of the five different arbitration policies, TDM, RR, FBSP, PBS and CCSP, which are typically used for shared memory access in real-time systems. (3) We compare the performance of GSMT in terms of area, power and maximum clock frequency with the traditional centralized implementations by synthesizing the designs in a 40 nm process.

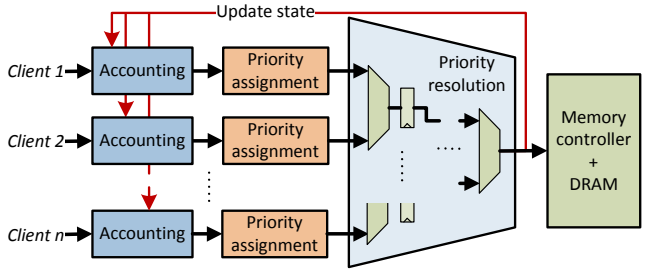

Fig. 1. High-level architecture of GSMT. *Accounting* keeps track of eligibility status of a client to get service. *Priority assignment* assigns a unique priority to each client and the fully-pipelined implementation of *Priority resolution* grants access to the client with highest priority.

In the remainder of this paper, Section II reviews the related work and Section III introduces the state-of-the-art real-time memory controllers and predictable arbitration policies. In Section IV, we introduce the generic GSMT architecture and its operation. Section V then introduces our proposed RTL-level design of configurable GSMT and its configuration as different arbitration policies. We present our experimental results in Section VI, and finally we conclude in Section VII.

II. Related work

Centralized implementations of predictable arbitration policies consisting of a tree of multiplexer stages for priority resolution among the clients presented in [12], [13], [15], [21] are not scalable as the number of logic gates in the critical path for multiplexing increases with the number of clients, restricting its maximum synthesizable frequency. Although the scalability issue is addressed using distributed implementation with global arbitration using TDM in Network-On-chip (NoC) [18]–[20], TDM is not suitable when the clients have diverse bandwidth and latency requirements. For example, the clients with low latency and bandwidth requirements often need to be allocated with more than their required bandwidth to meet their latency requirements, which is not desirable when the memory bandwidth is scarce. NoC using priority-based packet switching [22] provides real-time guarantees, but the buffering of packets in every router stage increases the memory access latency, area and power consumption [19].

Memory trees with distributed implementation of several local arbiters consisting of 2-to-1 multiplexer stages connected in a tree-like structure and each stage having a RR arbiter are presented in [23], [24], and a similar binary arbitration tree using a First Come First Serve (FCFS) policy in [17]. In the hybrid arbitration tree [16], a combination of RR and TDM arbitration policies were used in which the latency-sensitive clients are scheduled with TDM and bandwidth-demanding clients with RR to improve their average-case performance. However, cascading multiple arbitration stages leads to longer access latencies as the memory requests might experience the worst-case interference in each arbitration stage and also larger area and power usage due to the buffering of memory requests at every arbitration stage.

To summarize, existing centralized implementation of arbitration policies are not scalable in terms of clock frequency with number of clients and the 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 TDM arbitration (bufferless) are not suitable for clients with diverse requirements. Also, there exist no re-configurable architecture supporting different arbitration policies. In this paper, we present a generic distributed architecture with global arbitration (GSMT) for scalable implementation of several predictable arbitration policies and a configurable RTL-level design of GSMT that can be configured with five different arbitration policies.

III. Background

We assume real-time memory controllers to be used in conjunction with our proposed memory tree (GSMT) and hence, we introduce them in this section. Also, we introduce the predictable arbitration policies supported by GSMT.

A. Real-time memory controllers

State-of-the-art real-time memory controllers [4]–[8] bound the execution time of DRAM transactions by fixing the memory access parameters, such as burst size and page policy, at design time. For simplicity, we assume a constant execution time (WCET) for read and write requests by taking the maximum of both. This is not a restrictive assumption as the WCET for read and write transactions can be made similar with negligible loss in the guaranteed bandwidth [25]. Hence, all the memory requests are scheduled at time intervals of fixed duration called *scheduling interval (SI)*, which is larger than or equal to the WCET of the requests.

B. Predictable arbitration policies

In this work, we consider five different arbitration policies, Time Division Multiplexing (TDM), Round Robin [9], Frame-Based Static Priority (FBSP) [10], Priority-Based Scheduler (PBS) [7], [11] and Credit-Controlled Static-Priority (CCSP) [12], which have been proposed for shared memory access in real-time systems.

TDM is a frame-based arbitration policy with a fixed frame size, f, consisting of one or more TDM slots and each slot is of size equal to the SI. Each client is statically assigned to one or more slots and the fraction of number of assigned slots to the total number of slots in the frame is the *rate*, ρ, allocated (corresponds to the fraction of total memory bandwidth). With the progress of time, the clients are served every SI according to the static order in the TDM schedule and the frame repeats itself at the end of every frame. RR is a special case of TDM where the frame size is equal to the number of clients and each client is assigned to exactly one slot such that the clients are served one after the other.

Similar to TDM, FBSP [10] is also frame-based with a fixed frame size and each client is assigned a *budget* of slots corresponding to its rate, ρ. However, unlike in TDM there is no static assignment of clients to the slots. Instead, each client is assigned a unique static priority and at every SI, the (backlogged) client with the highest priority and one or more budget slots is granted service. When a client is granted service during an SI, its budget slot count is reduced by one. At the beginning of every new frame, the budgets of all clients are reset to their initial values. PBS is a special case of FBSP where only one of the clients is assigned with the highest priority [11] and every other client has equal priority.

Unlike frame-based arbitration policies, CCSP [12] does not use the notion of frames for the replenishment of the budgets of the clients. Instead, the budget for each client is *replenished continuously*, i.e. for every SI. This means that the replenishment interval in CCSP is a lot less than the frame-based arbitration policies. The service provided to a client depends on allocated *burstiness (σ)*, rate (ρ), and its static priority. To start, a client is credited with initial budget, which depends on σ. During every SI, the budget level is incremented at a constant fractional rate ρ and decremented by one when it is granted service. When the client is not backlogged, it is only allowed to build up its budget until its initial budget.

In all the arbitration policies that we discussed above, *work conservation*, i.e. assigning the unused time slot of a client to another client with pending request(s), can be implemented according to a certain *slack management policy*. For example, higher priorities can be given to the bandwidth demanding clients to improve their average-case performance or using the same static-priority levels in priority-based arbitration policies in work-conserving mode as well. Note that the budget of the scheduled client in work conservation mode is not deducted. One more important aspect of the various arbitration policies is that each of them comes with different properties. TDM is suitable for providing temporal isolation among the clients and RR when all clients need fair treatment. FBSP, PBS and CCSP are priority-based with different benefits [10] and suitable when differentiated treatment needs to be provided to the clients. This means that an arbitration policy need to be selected according to the requirements of clients running in the system.

IV. Generic Scalable Memory Tree (GSMT)

Given that we have presented the concept behind real-time memory controllers and different predictable arbitration

policies, we proceed by introducing our proposed scalable memory tree (GSMT) in this section. Before we present the detailed architecture and operation of GSMT, we first discuss the novel concept by which we achieve scalability, global arbitration and genericness in GSMT.

Scalability: We propose a distributed implementation, as shown in Figure 1, with dedicated *Accounting* and *Priority assignment* logic for each client with *Priority resolution* among the clients using a tree consisting of 2-to-1 pipelined multiplexer stages. The Accounting logic keeps track of the eligibility status of a client to get service, the Priority assignment logic assigns a unique priority to the client based on 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. The use of pipelined multiplexer stages for priority resolution breaks the critical path and enables the logic to be synthesized at higher clock frequencies.

Global arbitration: The Accounting logic of all clients use a *global scheduling interval* of fixed duration determined by the fixed access duration of the memory controller and the pipeline depth of the multiplexer stages in Priority resolution. Since the scheduling decision is made at the Accounting logic which is at the leaves of the tree, the pipeline registers in the multiplexer tree are simple latches of width equal the data-path width unlike the huge buffers at every local arbitration stage in the existing distributed implementations.

Generalization: Several predictable arbiters can be realized by configuring the Accounting logic. In TDM, 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. In TDM, 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 [10] are assigned to the eligible clients. 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 which are lower than their priority levels when they are eligible. The priority levels in the work-conserving mode depends on the slack management policy.

A. Detailed architecture

Figure 2 shows the detailed architecture of GSMT in which the clients are at the leaves of the tree and the memory controller 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 real-time memory controller.

The incoming memory requests from a client is split into equal sized service units at the NI according to the fixed access size of the memory controller. Each service unit is then scheduled by the arbitration policy at fixed scheduling intervals (SI). When a request is scheduled, the request valid (v) signal is asserted and the data/command (d) and the priority (p) of the client is transmitted over the bus. When two inputs of the

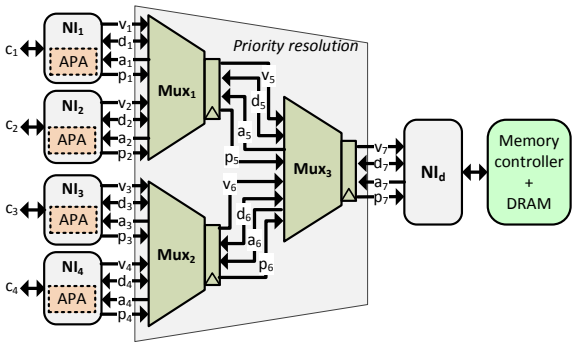

Fig. 2. Detailed architecture of GSMT along with the memory controller and the DRAM shared by four memory clients $c_1 - c_4$.

multiplexer stage arrive at the same clock cycle, the one which carries the highest priority is granted access and the other is dropped. When a service unit arrives at the root, NI_d generates an acknowledgement (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 in Section V). The dropped service units are not removed from their request buffers (no acknowledgement) and they are re-scheduled during the next SI. For a read request, the response arrives back at the source on a dedicated response path. We assume the same clock domain and data-path width for both GSMT and the memory controller to ensure that their SIs are of the same duration. Hence, the buffer in the memory controller will not overflow as the service unit (if any) scheduled by the tree will be consumed by the memory during the same SI. On the other hand, it is still possible to have different data-path widths for the memory tree and the controller and run them at different speeds by coupling the GSMT and memory controller as proposed in [19]. Note that during the periodic DRAM refresh operation by the memory controller, the service unit arriving at the root is dropped and rescheduled again.

B. Timing behavior

Figure 3 shows an example timing behavior of the GSMT when there are pending requests to be scheduled in the FIFO of NI_1 and NI_3 from clients c_1 and c_3, respectively (irrelevant signals are omitted for clarity). At the beginning of the first SI (red 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 and d_3, 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 3). 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 acknowledgement on a_7 after one clock cycle delay as shown. The acknowledgement 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 SI, NI_3 reschedules the dropped request.

It can be seen that for functional correctness, the minimum SI duration (SI_{min}), must at least be equal to or greater than the total time when a request is scheduled until its acknowledgement arrives back at the source NI. This depends on the number of multiplexer stages in the tree, which in turn depends on the number of clients in the system. This constraint is given by $SI_{min} \geq 2 \times \log_2(C)$, where C is

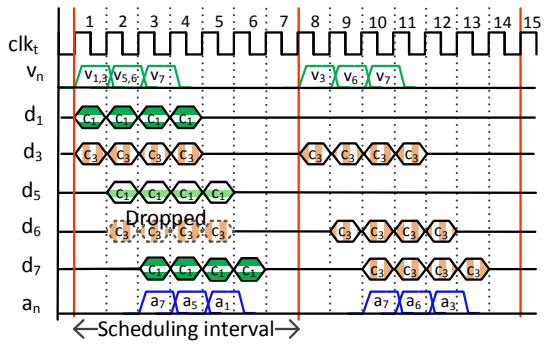

Fig. 3. Example timing diagram showing scheduling of write requests from clients c_1 and c_3. All valid (v) and accept (a) signals are combined and shown together as (v_n) and (a_n), respectively.

the number of memory clients and note that each multiplexer stage introduces one cycle delay each in the request and response paths. For a 16-bit IO DDR3-800 memory device, the WCET for the smallest request size of 16 Bytes is 25 clock cycles [26] assuming a *close-page policy* [10]. If we assume that GSMT runs at the same clock frequency as the memory device, the minimum SI of 12 cycles with up to 64 clients is less than the WCET for the smallest request size. Hence, the memory bandwidth is not negatively impacted due to the pipeline delays in GSMT. Moreover, with larger request sizes and faster memories the WCET increases making this constraint insignificant. Note that GSMT may not suitable for SRAMs when the data can be accessed in a single clock cycle.

V. Generic APA architecture and configuration

In this section, first we present our proposed generic RTL-level architecture of the Accounting and Priority assignment (APA) logic and then show how it can be configured to operate as either TDM, RR, FBSP, PBS or CCSP, which typically are used for sharing DRAM resource in real-time systems.

A. APA architecture

The RTL-level architecture of our proposed generic APA logic is shown in Figure 4. In the NI, the Atomizer splits an incoming request into smaller requests (corresponding to the fixed transaction size in real-time memory controller) and the FIFO buffer stores all pending requests 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. Work conservation is disabled by setting WC to zero, which enables asserting *v* only when a client is eligible (has enough budget) to get service and is backlogged.

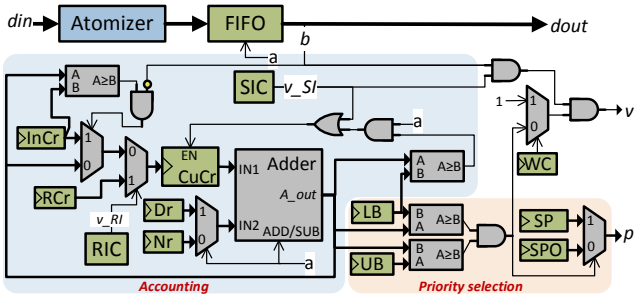

Fig. 4. Generic APA architecture that can be configured to operate as either TDM, RR, FBSP, PBS or CCSP.

Algorithm 1 shows the logical operation of the Accounting and Priority assignment blocks[1]. In Accounting, the value in register Current credits (CuCr) is incremented by the value in the register Numerator (Nr) at every SI (line 8). The SI counter (SIC) asserts a valid signal, *v_SI*, indicating the start of every new SI. 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 acknowledgement (a) signal is valid. Note that CuCr is updated once every SI, since it is enabled (EN) when *v_SI* is asserted. As explained in Section III, the building up of budget in CCSP mode is not allowed when the client is not backlogged. Hence, when b is not asserted, the value in CuCr is restricted to the initial budget stored in register Initial credits (InCr) using the multiplexer logic which selects InCr when the output of the Adder (*A_out*) is greater than or equal to the initial budget (lines 3-4). On a valid acknowledgement, CuCr is decremented by the value in register Denominator (Dr) (lines 10-11). The RI counter (RIC) is used by frame-based arbitration policies to replenish budget every replenishment interval by asserting *v_RI* when CuCr is reset to the value in RCr (line 6).

Algorithm 1 Accounting and Priority assignment logic

Input signals: Acknowledgement (a), Backlogged (b)
Output signal: Priority (p)

```
1:  procedure ACCOUNTING(a, b)
2:      if v_SI then
3:          if ((!b) & (A_out ≥ InCr)) then
4:              CuCr ← InCr
5:          else if v_RI then
6:              CuCr ← RCr
7:          else
8:              CuCr ← CuCr + Nr
9:          end if
10:     else if ((a) & (A_out ≥ LB)) then
11:         CuCr ← CuCr - Dr
12:     end if
13: end procedure
14: procedure PRIORITY ASSIGNMENT(A_out)
15:     if LB ≤ A_out ≤ UB then
16:         p ← SP
17:     else
18:         p ← SPO
19:     end if
20:     return p
21: end procedure
```

The Priority assignment logic selects a priority level stored in the register Static priority (SP) when the value of the Adder output, *A_out*, falls in between the values stored in registers Lower bound (LB) and Upper bound (UB) (lines 15-16). A different priority level with a constant offset as configured in the register SP offset (SPO) is selected (lines 17-18) in work conservation mode, i.e, when the client is not eligible to get service in the current SI. The value of the offset needs to be selected according to the slack management policy as discussed in Section III. When a client is scheduled in work-conserving mode, its budget is not deducted. Hence, its current budget level is checked against the sufficient budget limit in LB before enabling CuCr (line 10). Note that in FBSP and CCSP, all non-eligible clients, i.e. with budget less than LB, are in work-conserving mode by definition, and TDM do not have the notion of budget for the clients.

B. APA configurations

A summary of the different programmable registers and counters in APA and the initial values need to be configured to implement the different arbitration policies are shown in Table I, which we will discuss in detail in this section.

TDM and RR: When configured in TDM or RR mode, the Accounting logic keeps track of the progress of the current

[1]For clarity in presentation, the pseudo code is split into two procedures, Accounting and Priority assignment. Accounting is triggered by signals *a* and *b*, whereas Priority assignment is purely combinatorial logic.

TABLE I. APA PROGRAMMABLE REGISTERS/COUNTERS AND THEIR CONFIGURATION FOR DIFFERENT ARBITRATION POLICIES

	Arbiter		
Register	*TDM*	*FBSP*	*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	End position in TDM frame	$> f \cdot \rho$	High value
LB	Start position in TDM frame	1	$nr - dr$
SIC	SI	SI	SI
RIC	$f \cdot SI$	$f \cdot SI$	Not used

frame in terms of number of slots and the Priority assignment logic sets the highest priority for a client during its allocated slot(s) 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 basically counts the progress of the current frame. To identify the start of a new frame, the RIC is configured to assert *v_RI* every frame. This resets the value in 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 or RR, 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. 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. We need to assign unique priorities to each client such that there is no conflict of priorities when SPO is selected. Here, the actual SP does not matter as it is unique. Note that InCr is not used in TDM mode but it is configured to the maximum value of f to ensure that CuCr is not updated from InCr.

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 a higher priority for the client on the priority lines as long as sufficient budget is available. At the start of every frame, CuCr is initialized with $f \cdot \rho$, which corresponds to the number of slots allocated to the client in a frame, i.e, maximum budget. The current budget needs to be decremented by one whenever a service unit gets scheduled i.e, successfully arrives at the root NI, 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 bandwidth and/or latency requirements) of the client and SPO with a constant offset. Note that SP is used while within budget and otherwise SPO is used, which ensures that the allocated bandwidth is guaranteed to the clients before the slack bandwidth is distributed among them. LB needs to be configured with a value of one and UB with a value greater than $f \cdot \rho$ such that the priority in SP is selected for a number of service units equal to $f \cdot \rho$ in a frame. Note that InCr is not used in FBSP mode but it is set to a maximum value of $f \cdot \rho$ to avoid initialization of CuCr from InCr.

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 acknowledgement arrives back, where nr and dr are integers used to represent the allocated rate, $\rho = nr/dr$. In the Priority assignment logic, SP and SPO are configured with the client's priority level and with a constant offset, respectively. LB is set to dr as $dr - nr$ is the minimum budget required to select SP and nr needs to be added to it since *A_out* 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 [12] .

To summarize, the APA logic in GSMT schedules requests exactly the same way as in the centralized implementations and the existing analyses [10] can hence be used for the computation of worst-case latency of a memory transaction for the demonstrated arbitration policies. However, the constant pipeline delay introduced by the number of multiplexer stages in the tree, discussed in Section IV-B, needs to be added to the worst-case latency.

VI. EXPERIMENTS

In this section, we present our experimental setup, verification of functional correctness of GSMT and its performance comparison with respect to centralized implementations of two different arbitration policies.

Our experimental setup consists of the RTL-level implementation of GSMT and centralized implementations of two different arbitration policies, TDM [25] and CCSP [10], 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 of the designs.

We ensured the functional correctness of GSMT by comparing the scheduling decisions made by the FPGA implementation of GSMT (with 16 clients) at every scheduling interval with the C++ models of centralized implementations of TDM, FBSP, and CCSP. Note that the other 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 GSMT and the centralized implementations were the same, and hence, *we conclude that GSMT correctly implements the different arbitration policies.*

We repeatedly synthesized the designs of GSMT and central implementation of TDM and CCSP[2] for different number of clients, i.e. 4, 8, 16, 32 and 64 to determine the maximum synthesizable frequency. Table II shows the area, power and maximum clock frequency of the GSMT and the centralized implementation of TDM, and CCSP arbitration policies. In general, it can be seen that the maximum clock frequency, f_{max}, of centralized TDM and CCSP do not scale with the number of clients. With 64 clients, GSMT can be run up to a clock frequency of 1.2 GHz, whereas CCSP and TDM are limited to 0.3 GHz.

In general, the area and power consumption of all different designs increase linearly with the number of clients due to the

[2]We selected frame-based (TDM) and priority-based (CCSP) arbiters to compare GSMT with different types of hardware implementations.

TABLE II. AREA, POWER AND MAXIMUM CLOCK FREQUENCY (f_{max}) OF GSMT AND CENTRALIZED IMPLEMENTATIONS OF TDM AND CCSP

	Area (mm^2)			Power (mW)			f_{max} (MHz)		
# Clients	*TDM*	*CCSP*	*GSMT*	*TDM*	*CCSP*	*GSMT*	*TDM*	*CCSP*	*GSMT*
4	0.016	0.020	0.017	5.19	5.35	4.55	588	526	1250
8	0.029	0.036	0.035	7.88	8.07	9.77	500	435	1250
16	0.061	0.077	0.070	16.13	14.94	20.20	435	357	1250
32	0.107	0.172	0.141	17.46	25.36	41.07	333	333	1250
64	0.203	0.417	0.282	35.60	63.18	82.81	333	303	1250

additional logic added. The area usage of centralized CCSP increases significantly with increasing number of clients due to the extra registers added to break the critical path, resulting in a low f_{max}. The f_{max} for centralized TDM scales down as well with increasing number of clients due to the critical path in the priority resolution of work-conserving mode. On the other hand, GSMT has better scalability in f_{max} with the number of clients since its critical path in the APA logic remains constant irrespective of the number of clients as it is dedicated for each client. However, it is worthwhile to note that GSMT consumes more power compared to the centralized implementations in most cases. This is primarily due to the addition of extra priority lines in the bus and the dedicated APA logic for each client. One limitation of GSMT 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 [10].

To efficiently compare the centralized designs and GSMT in terms of frequency, area and power consumption, we define two cost-efficiency metrics, *bandwidth/area* and *bandwidth/power*. Bandwidth is computed by multiplying data-path width (in Bytes) with the clock frequency (f_{max}). (Figure 5 shows the ratio of bandwidth (Bytes/s) to area usage (mm^2) and bandwidth (Bytes/s) to power consumption (mW) of centralized CCSP and TDM normalized to GSMT. It can be seen that for all configurations of clients, GSMT has over 51% and 37% performance gain in terms of area and power consumption, respectively, compared to traditional centralized implementations of CCSP and TDM. Hence, *we can conclude that GSMT is suitable when there are a large number of memory clients in the system that requires the arbiter to be clocked at higher speed.*

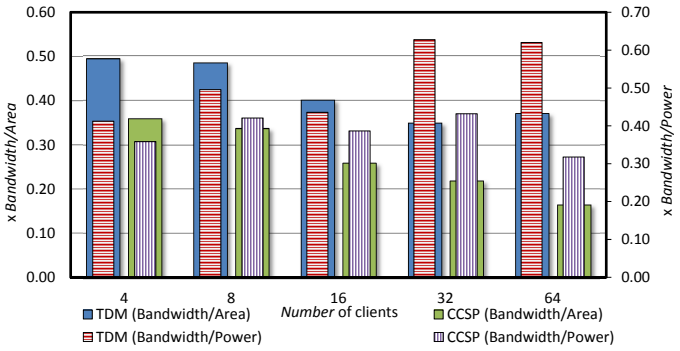

Fig. 5. Bandwidth/Area and Bandwidth/Power performance of centralized CCSP and TDM arbiters normalized to GSMT.

VII. CONCLUSIONS

Sharing DRAM between multiple memory clients in a real-time system requires implementation of predictable arbitration policies. Traditional centralized implementations of predictable arbitration policies are not scalable in terms of clock frequency with increasing number of clients due to the long critical path in the logic for priority resolution among the clients. On the other hand, existing distributed implementations either cannot provide differential treatment to the clients or have poor performance in terms of area, power consumption and latency. In this paper, we presented a novel generic scalable memory tree, GSMT, for distributed implementation of several predictable arbitration policies. Moreover, we presented a configurable RTL-level design of GSMT that can be configured to operate as five different arbitration policies proposed for shared memory access in real-time systems. Our experimental results show that GSMT outperforms the centralized implementations by more than four times in terms of clock speed and over 51% and 37% in terms of area and power consumption, respectively.

ACKNOWLEDGMENT

This work was partially funded by projects EU FP7 288008 T-CREST and 288248 Flextiles, CA505 BENEFIC, CA703 OpenES, ARTEMIS-2013-1 621429 EMC2, 621353 DEWI, and Czech Ministry of Education CZ.1.07/2.3.00/30.0034.

REFERENCES

[1] P. Kollig *et al.*, "Heterogeneous Multi-Core Platform for Consumer Multimedia Applications," in *Proc. DATE*, 2009.

[2] C. van Berkel, "Multi-core For Mobile Phones," in *Proc. DATE*, 2009.

[3] P. van der Wolf *et al.*, "SoC Infrastructures for Predictable System Integration," in *Proc. DATE*, 2011.

[4] M. Paolieri *et al.*, "Timing Effects of DDR Memory Systems in Hard Real-time Multicore Architectures: Issues and Solutions," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1s, 2013.

[5] B. Akesson *et al.*, "Architectures and Modeling of Predictable Memory Controllers for Improved System Integration," in *Proc. DATE*, 2011.

[6] J. Reineke *et al.*, "PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation," in *Proc. CODES+ISSS*, 2011.

[7] H. Shah *et al.*, "Bounding WCET of applications using SDRAM with Priority Based Budget Scheduling in MPSoCs," in *Proc. DATE*, 2012.

[8] Y. Li *et al.*, "Dynamic Command Scheduling for Real-Time Memory Controllers," in *Proc. ECRTS*, 2014.

[9] M. Katevenis *et al.*, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, 1991.

[10] B. Akesson and K. Goossens, *Memory Controllers for Real-Time Embedded Systems*, 1st ed., ser. Embedded Systems. Springer, 2011.

[11] M. Steine *et al.*, "A priority-based budget scheduler with conservative dataflow model," in *Proc. DSD*, 2009.

[12] B. Akesson *et al.*, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *Proc. RTCSA*, 2008.

[13] G. Dimitrakopoulos *et al.*, "Scalable Arbiters and Multiplexers for On-FGPA Interconnection Networks," in *Proc. FPL*, 2011.

[14] K. Chapman, "Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources. Application Note," in *http://www.xilinx.com*, 2012.

[15] E.S. Shin *et al.*, "Round-robin Arbiter Design and Generation," in *Proc. ISSS*, 2002.

[16] K. Goossens *et al.*, "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.

[17] J.H. Rutgers *et al.*, "Evaluation of a Connectionless NoC for a Real-Time Distributed Shared Memory Many-Core System," in *Proc. DSD*, 2012.

[18] A. Hansson *et al.*, "Channel trees: reducing latency by sharing time slots in time-multiplexed networks on chip," in *CODES+ISSS*, 2007.

[19] M.D. Gomony *et al.*, "Coupling TDM NoC and DRAM Controller for Cost and Performance Optimization of Real-Time Systems," in *Proc. DATE*, 2014.

[20] M. Schoeberl *et al.*, "A Time-Predictable Memory Network-on-Chip," in *Proc. WCET*, 2014.

[21] M. Weber, "Arbiters: design ideas and coding styles," in *SNUG*, 2001.

[22] Z. Shi and A. Burns, "Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching," in *Proc. NOCS*, 2008.

[23] J. Garside and N. Audsley, "Prefetching across a shared memory tree within a Network-on-Chip architecture," in *Proc. SoC*, 2013.

[24] A. Rahimi *et al.*, "A fully-synthesizable single-cycle interconnection network for Shared-L1 processor clusters," in *Proc. DATE*, 2011.

[25] S. Goossens *et al.*, "A Reconfigurable Real-Time SDRAM Controller for Mixed Time-Criticality Systems," in *In Proc. CODES+ISSS*, 2013.

[26] "MICRON DDR3 SDRAM Specification," 2006, Rev. M 08/14 EN.