Coupling TDM NoC and DRAM Controller for Cost and Performance Optimization of Real-Time Systems

Manil Dev Gomony*, Benny Akesson†, and Kees Goossens*
*Eindhoven University of Technology, The Netherlands
†Czech Technical University in Prague, Czech Republic

***Abstract*—Existing memory subsystems and TDM NoCs for real-time systems are optimized independently in terms of cost and performance by configuring their arbiters according to the bandwidth and/or latency requirements of their clients. However, when they are used in conjunction, and run in different clock domains, i.e. they are decoupled, there exists no structured methodology to select the NoC interface width and operating frequency for minimizing area and/or power consumption. Moreover, the multiple arbitration points, one in the NoC and the other in the memory subsystem, introduce additional overhead in the worst-case guaranteed latency. These makes it hard to design cost-efficient real-time systems.**

The three main contributions in this paper are: (1) We present a novel methodology to couple any existing TDM NoC with a real-time memory controller and compute the different NoC interface width and operating frequency combinations for minimal area and/or power consumption. (2) For two different TDM NoC types, one a packet-switched and the other circuit-switched, we show the trade-off between area and power consumption with the different NoC configurations, for different DRAM generations. (3) We compare the coupled and decoupled architectures with the two NoCs, in terms of guaranteed worst-case latency, area and power consumption by synthesizing the designs in 40 nm technology. Our experiments show that using a coupled architecture in a system consisting of 16 clients results in savings of over 44% in guaranteed latency, 18% and 17% in area, 19% and 11% in power consumption for a packet-switched and a circuit-switched TDM NoC, respectively, with different DRAM types.

I. Introduction

In multi-processor platforms for real-time systems, main memory (off-chip DRAM) is typically a shared resource for cost reasons and to enable communication between the processing elements [1]–[3]. Such platforms run several applications with real-time requirements on main memory performance in terms of bandwidth and/or latency [4], [5]. These real-time requirements must be guaranteed at design time to reduce the cost of verification effort. This is made possible using real-time memory subsystems [6]–[8] that bound the memory access time by fixing the memory access parameters at design-time and employing predictable arbitration, such as Time Division Multiplexing (TDM) and Round-Robin, using bus-based interconnects for resource sharing. On the other hand, statically scheduled TDM NoCs [9]–[15] solve the scalability issues with the traditional bus-based approaches [16] for accessing shared resources in larger multi-processor platforms. To provide end-to-end guarantees on latency in TDM NoCs, a dedicated virtual circuit is allocated to each client from a source network interface (NI) to a destination. Both real-time memory controllers and statically scheduled TDM NoCs can be analyzed using shared resource abstractions, such as the Latency-Rate ($\mathcal{LR}$) server model [17], which can be used in formal performance analysis based on e.g., network calculus or data-flow analysis.

Currently, real-time memory subsystems and TDM NoCs are optimized independently by allocating each client with its required worst-case bandwidth and/or latency in the arbiters of the NoC and the memory subsystem for minimum resource utilization. In larger multi-processor platforms for real-time systems, TDM NoCs and real-time memory subsystems need to be used in conjunction, with the NoC in a tree topology and the memory subsystem at the root of the tree [18], as shown in Figure 1. Since they run in different clock domains, the virtual circuit for each client in the NoC needs to be decoupled using dedicated buffers in the memory subsystem. Moreover, the NI, which interfaces with the memory subsystem, requires a separate port for each virtual circuit increasing its area and power consumption, as well as the worst-case latency of a memory transaction due to the introduction of multiple independent arbitration points. However, for a decoupled architecture, there exists no structured methodology to select the NoC interface width and/or operating frequency for minimal area and/or power consumption. This makes it difficult to design cost-efficient real-time systems.

The three main contributions in this paper are: (1) We propose a novel methodology to couple any existing TDM NoC with a real-time memory controller and compute the different NoC interface width and operating frequency combinations for minimal area and/or power consumption. (2) For two different NoC types, one a packet-switched and the other circuit-switched, we show the trade-off between area and power consumption with the different NoC configurations, and for different DRAM devices across three memory generations. (3) We compare the performance of the coupled and decoupled architectures with the two NoCs in terms of worst-case latency, area and power consumption by synthesizing designs in 40 nm technology, for the different DRAM devices.

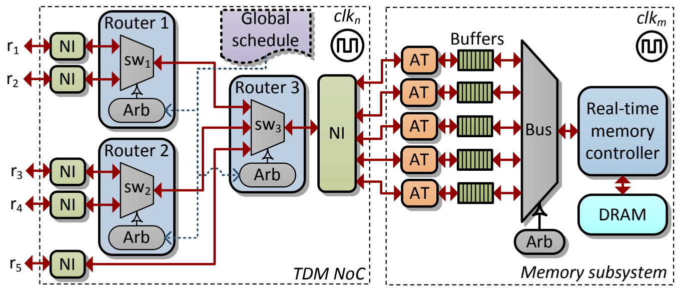

Fig. 1. High-level architecture of a decoupled TDM NoC tree and a memory subsystem shared by five memory clients r_1 to r_5. The NoC and the memory subsystem run in different clock domains using clock sources clk_n and clk_m, respectively. Every router in the NoC has a switch (sw) and an arbiter (Arb), which arbitrates according to a global TDM schedule. The *atomizer* (AT) splits a larger transaction in to smaller sized transactions according to the fixed access size of the real-time memory controller.

In the remainder of this paper, Section II reviews the related work, while Section III gives an introduction to the $\mathcal{LR}$ server model, state-of-the-art real-time memory controllers and TDM NoCs. In Section IV, we introduce our methodology to couple a TDM NoC with a real-time memory controller and select the NoC parameters. By experimentation, we analyze the impact of NoC parameter selection on its area and power consumption and compare the coupled and decoupled architectures in Section V, and finally we conclude in Section VI.

II. Related work

Related work on the co-design of NoC and memory subsystem can be classified into those that provide guarantees on real-time requirements to the clients and those that focus on improving the average-case performance of the system. To provide real-time guarantees while accessing a shared memory, power/performance optimized tree topologies using bus-based interconnects and predictable arbiters [16], [19] were traditionally used. Since, bus-based interconnects were not scalable in larger SoCs, statically scheduled TDM [9]–[15] and priority-based NoCs [20] were introduced. Although, a tree topology using TDM NoC, *channel trees* [18], has been proposed for accessing shared resources reduces the worst-case latency using a fully pipelined response path, it was not optimized for area or power consumption. Optimization of a TDM NoC in terms of area and power consumption was done in [21] by coupling with a memory interface and removing the buffers and flow control by implementing a Direct Memory Access (DMA) access table inside a NI. However, the approach is specific for DMA clients and is not applicable for clients with diverse requirements on bandwidth and/or latency. At the application level, DRAM-aware mapping of application tasks to the processing nodes exploits bank-level parallelism [22] has been proposed, and memory-centric scheduling approach in which statically computed TDMA schedules are made for each core to access the memory system to avoid memory access contention, allowing applications to be verified in isolation [23]. However, those were not optimized in terms of area and/or power consumption.

Other related work that focus on improving the average-case performance of the clients include a memory-centric NoC design that explores the benefits of a dedicated NoC for shared DRAM access by funneling the traffic from different clients to the memory with the right width converters [24]. A connectionless NoC for shared memory access with a binary arbitration tree that multiplexes multiple clients to one bus master are proven to reduce average latency and hardware cost as opposed to a connection-oriented NoC [25]. At the architecture level, DRAM traffic-aware NoC routers [26] and network interfaces [27] exploit bank-level parallelism and minimize bus turnaround time by grouping read and write transactions to improve memory utilization. Memory controllers that interacts with the NoC and make command scheduling decisions based on the congestion information from the NoC [28] improves the average-case performance, but do not provide guarantees on bandwidth and/or latency to the clients.

To the best of our knowledge, there exists no previous work that optimizes the co-design of a TDM NoC and a shared memory subsystem in terms of area, power consumption and performance, while providing real-time guarantees on bandwidth and/or latency to the clients.

III. Background

This section first introduces the $\mathcal{LR}$ server model since we use it as the shared resource abstraction to derive bounds on latency provided by predictable arbiters. We then introduce state-of-the-art real-time memory subsystems and TDM NoCs.

A. $\mathcal{LR}$ servers

Latency-Rate ($\mathcal{LR}$) servers [17] is a general shared resource abstraction model that provides a lower linear bound on the service provided by various scheduling algorithms or arbiters, such as TDM and Round-Robin, to a client. The $\mathcal{LR}$ abstraction helps to formally verify applications using shared resources by using a variety of formal analysis frameworks, such as data-flow analysis and network calculus.

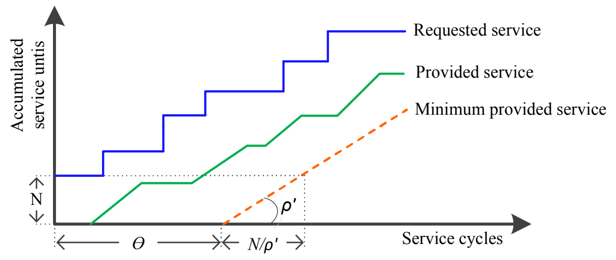

Fig. 2. Example service curves of a $\mathcal{LR}$ server showing service latency (Θ) and completion latency (N/ρ').

Figure 2 shows example requested and provided service curves of a $\mathcal{LR}$ server. The requested service by a client at a time consists of one or more *service units* (indicated on the y-axis) corresponding to data. According to the $\mathcal{LR}$ abstraction, the minimum service provided to the client depends on two parameters, namely the *service latency*, Θ, and the *allocated rate*, ρ', (bandwidth). The *service latency* is intuitively the worst-case time taken by the arbiter to schedule a request at the head of the request queue of a client because of interfering clients and depends on the arbiter and its configuration, e.g. the allocated rate. Once a request consisting of N service units is scheduled, its service units will be served at a guaranteed rate ρ' and it hence takes N/ρ' *service cycles* to finish serving the request. The *worst-case latency*, $\hat{L}$, (in service cycles) of a client is the total time taken by a request of size N service units at the head of the client's request queue to get served in the worst case, and is given by $\hat{L} = \Theta + \lceil N/\rho' \rceil$.

B. Real-time memory subsystems

Existing real-time memory controllers [6]–[8] bound the execution time for a memory transaction by fixing the memory access parameters, such as burst length and number of read/write commands per request, at design-time. Hence, the amount of data accessed in the memory device while serving a single transaction is always fixed and we refer to these transactions of fixed size as *service units* with size (in Bytes) SU^{bytes}. The service unit size of DRAMs is typically in the range of 16-256 Bytes. The time (in ns) taken by the memory controller to finish the execution of a service unit is called a *memory service cycle* and is given by SC_m^{ns}. For a given memory device with operating frequency f_m, the memory service cycle SC_m^{cc} of a service unit size of SU^{bytes}can be computed according to [29]. Also, the worst-case bandwidth, b^{gross}, offered by a memory for a fixed access granularity can be computed. It is shown in [30] that the memory service cycle for read and write transactions in DRAMs can be equally long with negligible loss in the guaranteed memory bandwidth.

For resource sharing between multiple memory clients, bus-based interconnects with an arbiter belonging to the class of $\mathcal{LR}$ servers are typically employed to provide real-time guarantees to the memory clients [7], as shown in Figure 1. The *atomizer (AT)* splits every transaction of a memory client into smaller service units of size equal to SU^{bytes}, according to the fixed transaction size of the memory controller. The worst-case latency of the memory subsystem (in ns), $\hat{L}_m$, to serve a memory request from a client consisting of N service units is given by Equation (1), where Θ_m is the service latency of the memory arbitration, ρ'_m the rate allocated to the client, and δ_{mem} the internal pipeline delay of the memory subsystem and depends on the number of pipeline stages in the RTL implementation of the memory subsystem.

$$\hat{L}_m = \frac{(\Theta_m + \lceil N/\rho'_m \rceil) \cdot SC_m^{cc} + \delta_{mem}}{f_m} \tag{1}$$

C. Statically scheduled TDM NoCs

State-of-the-art TDM NoCs are either packet switched [9]–[11], [13], [15] or circuit switched [12], [14], according to a statically computed global TDM schedule. The routers in these NoCs are non-blocking, and hence, the Network Interface (NI) performs end-to-end flow control to avoid overflow of the buffers in the routers. The NI converts a memory transaction into one or more smaller units, called *flits*, and transports them to a destination NI according to the global TDM schedule. Typically, the service unit size of the NoC is in the order of few words (4-12 Bytes) for a smaller buffer area in the NI [18]. The time taken to transport a complete service unit over a NoC link is called *NoC service cycle* and is given by SC_n^{ns} (in ns) and SC_n^{cc} (in clock cycles). For a service unit of size SU^{bytes}, SC_n^{cc} can be computed using Equation (2), where, IW_n is the interface width of the NoC (in bits) and δ_{ov} is the overhead cycles to carry header, such as address and command that is specific to a NoC architecture.

$$SC_n^{cc} = \lceil (SU^{bytes} \times 8)/IW_n \rceil + \delta_{ov} \tag{2}$$

We use the TDM NoC in a tree topology with the memory subsystem at the root of the tree and a fully pipelined response path without arbitration as in a channel tree [18]. The worst-case latency of a client in the NoC (in ns), $\hat{L}_n$, to transport a read request from a source to a destination NI and read back the response consisting of N service units is given by Equation (3) [1], where f_n is the NoC operating frequency (in MHz), Θ_n is the service latency in the global TDM arbitration, ρ'_n the rate allocated to the client in the NoC, n_{hops} the number of hops between the source and destination NI, and δ_{sw} the fixed pipeline delay of a NoC router. We assume no flow control because of the buffers in the memory subsystem, and request and response paths of equal length.

$$\hat{L}_n = \frac{(\Theta_n + \lceil N/\rho'_n \rceil) \cdot SC_n^{cc} + 2n_{hops} \cdot \delta_{sw}}{f_n} \tag{3}$$

For the decoupled architecture, previously shown in Figure 1, the worst-case latency of a memory request can be computed as the sum of worst-case latencies of the NoC, $\hat{L}_n$, and the memory subsystem, $\hat{L}_m$. It can be seen that this approach could be pessimistic in terms of worst-case latency, since there are multiple arbitration points for the same request and each of them introduces a service latency in the worst-case latency estimation. In the next section, we present a novel methodology to couple any statically scheduled TDM NoC with a real-time memory subsystem that contains only a single arbitration point for each client.

IV. Coupling TDM NoC and Memory Subsystem

We present our proposed methodology to couple any existing TDM NoC with a real-time memory controller and compute the NoC parameters in this section. Also, we show the worst-case guarantees provided by the coupled architecture.

A. Architecture and operation

The basic idea is to use a single clock domain and generate different clock frequencies for both the memory subsystem and NoC. This helps to align the clock edges at the service cycle boundaries, which enables us to schedule the service units to the memory controller using the single global TDM schedule without the memory arbiter, bus and the decoupling buffers in between. This implies that the service unit size must be of equal size in the NoC and the memory subsystem and the memory service cycle must be equal to the NoC service cycle (in ns), i.e., $SC_m^{ns} = SC_n^{ns}$.

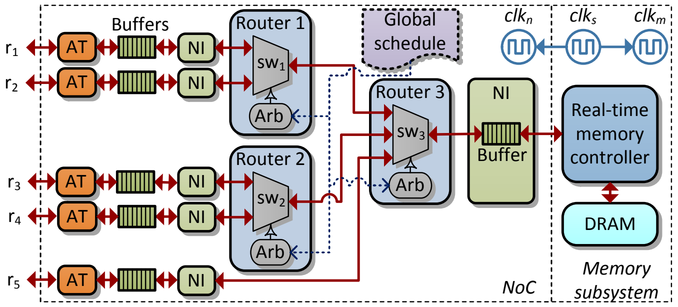

Fig. 3. High-level architecture of a coupled interconnect with a memory subsystem shared by five memory clients r_1 to r_5.

Figure 3 shows the high-level architecture of a TDM NoC coupled with a memory subsystem shared by five memory clients $r_1 - r_5$. The clocks for the NoC and the memory subsystem represented by clk_n (f_n) and clk_m (f_m), respectively, are derived from a single clock source, clk_s, by dividing in different ratios, m and n, respectively. The memory subsystem consists of a real-time memory controller, such as [6]–[8], attached to a memory device operating at a frequency (in MHz) given by f_m. Compared to the decoupled architecture previously shown in Figure 1, the decoupling buffers and the atomizer in the memory subsystem are moved to the top of the NoC tree from the memory subsystem. Since the service units are scheduled using the global TDM arbitration in the NoC, the bus and the arbiter in the memory subsystem are not required anymore. The NI attached to the memory controller needs only a single port instead of a number of ports equal to the number of clients in the decoupled architecture. However, since there are no decoupling buffers in the memory subsystem, the NI requires a buffer of size equal to the service unit as opposed to the buffer-less NI in the decoupled architecture. This is because the memory controller cannot serve the transaction until the complete service unit (including the payload of a write transaction) is available in its input as a DRAM device needs a complete burst before it can perform a read/write operation, and typically, reads/writes two words in a single clock cycle.

Consider the TDM slot allocation for the five clients $r_1 - r_5$, as shown in Figure 4, which corresponds to the global TDM schedule. Figure 5 shows the clock-cycle-level behavior of the interconnect. For simplicity, we assume $SC_m^{cc} = 4$ cycles and $SC_n^{cc} = 6$ cycles and $f_n = \frac{3}{2} f_m$, i.e., three clocks of clk_n corresponds to two clocks of clk_m. In the first service cycle ($SC1$), Router 1 schedules the service unit of client r_1 and the first word arrives at Router 3 after a pipeline delay of Router 1 equal to δ_{sw} (one clock cycle in this example). Router 3 immediately forwards the service unit to the memory controller (mc) after introducing another pipeline delay, and hence, the service unit of r_1 arrives after $2\delta_{sw}$ cycles. The memory controller has to wait for one service cycle for the complete service unit to be delivered by the NoC. Once the memory controller issues a read/write command, it takes δ_{mem} cycles for the command to reach the memory device.

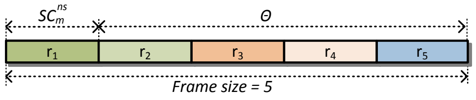

Fig. 4. Global TDM allocation for clients $r_1 - r_5$.

The response path is fully pipelined and there is no flow control as in the channel trees. Hence, the memory controller sends back a read response as soon as it buffers the complete data from the memory device. As shown in Figure 5, the

[1] For a write request, only the request path need to be considered, and hence, there will be a single $n_{hops} \cdot \delta_{sw}$ in Equation (3).

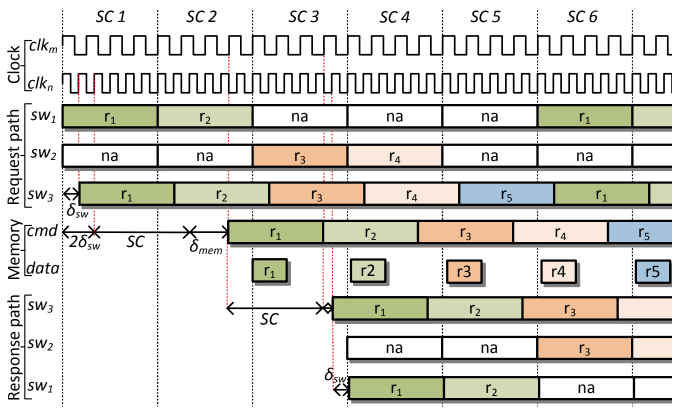

Fig. 5. Timing diagram showing the coupled architecture operation.

response for the service unit of r_1 is completely buffered in the memory controller after one service cycle. However, at this point the end of the service cycle may not be aligned with the NoC clock and hence, it has to wait for the next rising edge of the clock before the response can be forwarded back to the client through the routers. Similarly, clients $r_2 - r_5$ are scheduled during the successive service cycles $SC2 - SC5$, respectively, as shown in the figure.

To summarize, a TDM NoC can be coupled with a real-time memory controller by configuring its flit size equal to the service unit size, selecting a buffer of size equal to the service unit size for the NI attached to the memory controller, and configuring the response path to be fully pipelined. Hence, any existing TDM NoC can be coupled with a real-time memory controller without significant hardware modifications.

B. Computation of NoC parameters

To ensure that the NoC delivers a complete service unit in a service cycle for a given memory device of operating frequency f_m and a service unit size of SU^{bytes} with a service cycle duration SC^{cc}_m, the NoC link bandwidth must be same as the memory subsystem in a service cycle as given by Equation (4).

$$f_n \times (IW_n/8) \times \frac{SC^{cc}_n - \delta_{ov}}{SC^{cc}_n} = f_m \times \frac{SU^{bytes}}{SC^{cc}_m} \tag{4}$$

Given f_m, SC^{cc}_m , SU^{bytes} and δ_{ov} , we need to determine all valid (f_n, IW_n) combinations. Since we use a single clock source for both the NoC and the memory subsystem, the possible f_n and f_m are integer divisions of the system clock frequency. Hence, at first all possible values of f_n that are integer multiples and common fractions of f_m need to be computed. However, we need to make sure that SC^{ns}_n= SC^{ns}_m and the NoC and the memory subsystem clocks are aligned at the boundary of the service cycle. Hence, those values of f_n that satisfy Equation (5) need to be selected which ensures that the number of cycles in SC^{cc}_n (previously defined in Equation (2)) corresponding to an f_n will also be the integer multiple or common fraction (same used for computing the f_n from f_m) of SC^{cc}_m, i.e., the clocks will be aligned at the edge of the service cycle. Finally, the values of IW_n corresponding to the different f_n can be computed by substituting the values of the known and the computed parameters in Equation (4).

$$(SC^{cc}_m \times f_n) \bmod f_m = 0 \tag{5}$$

C. Real-time guarantees

In the coupled architecture, the worst-case latency (in ns), $\hat{L_c}$, for a read request consisting of N service units at the head of a client's request queue is given by Equation (6), where, ρ_c is the rate allocated to the client in the arbiter of the coupled architecture and Θ_c is the service latency caused by interfering clients. It can be seen that the coupled architecture introduces only a single service latency component in the worst-case latency expression. The worst-case latency for a read request given by Equation (6) consists of an additional memory clock cycle since the response from the memory device may not be aligned with the NoC clock and hence it has to wait for one memory clock. It can be seen in Equation (6) that the coupled architecture introduces only a single service latency component in the worst-case latency as opposed to the two in the decoupled architecture, one by the memory subsystem (Equation (1)) and the other by the NoC (Equation (3)).

$$\hat{L_c} = \frac{(\Theta_c + \lceil N/\rho_c \rceil) \cdot SC^{cc}_n + 2n_{hops} \cdot \delta_{sw}}{f_n} + \frac{\delta_{mem} + SC^{cc}_m + 1}{f_m} \tag{6}$$

V. Experiments

In this section, we first present the trade-off between area and power consumption for different frequency and interface width configurations for the different NoC and memory types. Then we compare the coupled and decoupled architectures in terms of area, power and guaranteed latency.

A. Experimental setup

Our experimental setup consists of RTL-level implementations of the following modules: (1) Router and NI of two different TDM NoC types, one is packet switched *Aelite* [10] and the other circuit switched *Daelite* [12]. (2) TDM arbiter. (3) Bus-based interconnect using the Device Transaction Level (DTL) protocol, which is comparable to functionality and complexity of AXI and OCP protocols [31]. For logic synthesis and for power/area estimation of the designs, we used the Cadence Encounter RTL compiler and the 40 nm nominal V_t CMOS standard cell technology library from TSMC.

B. Impact of NoC parameters on area and power consumption

We analyze the impact of different operating frequency and interface width (f_n, IW_n) combinations for the coupled NoC on its area and power consumption in this section. First, we computed the memory service cycle SC^{cc}_m and worst-case guaranteed bandwidth b^{gross} for different DRAM devices (of 16-bit IO) with different service unit sizes (16 B, 32 B, 64 B, 128 B and 256 B) according to [29]. For all those memories with different service unit sizes, we then computed the different (f_n, IW_n) combinations for both Aelite and Daelite NoCs according to the methodology presented in Section IV-B. The values of δ_{ov} for Aelite and Daelite is 3 and 2 cycles, respectively. Table I shows the different NoC operating frequencies and interface widths for different memories with a service unit size of 64 B. In general, it can be seen that the interface width and operating frequency requirement of the NoCs increase with the gross bandwidth of the memory device, as expected. Both NoCs may have different interface width requirements because of their different overhead, δ_{ov}. Due to lack of space, we do not show the (f_n, IW_n) combinations for all service unit sizes. However, we found that increasing service unit size increases the gross bandwidth of the memory because of more efficient memory accesses and hence a faster and/or wider NoC is required.

To analyze the trade-off between area and power consumption with the different (f_n, IW_n) combinations in Table I, we synthesized the RTL design of a single router-NI combination of both Aelite and Daelite NoCs with the different (f_n, IW_n)

TABLE I. MEMORY SERVICE CYCLE AND GROSS BANDWIDTH FOR DIFFERENT DRAMS WITH A SERVICE UNIT SIZE OF 64 B

Memory	f_m (MHz)	SC_m^{cc} (cc)	b^{gross} (MB/s)	(f_n, Aelite IW_n, Daelite IW_n) combinations
LPDDR-266	133	19	448.0	(133,32,31),(266,15,15),(399,10,10),(532,8,7),(665,6,6),(798,5,5),(931,4,4)
LPDDR-416	208	19	700.6	(208,32,31),(416,15,15),(624,10,10),(832,8,7),(1040,6,6),(1248,5,5)
LPDDR2-667	333	25	853.8	(133.2,74,64),(199.8,43,40),(266.4,31,29),(333,24,23),(399.6,19,19),(466.2,16,16), (532.8,14,14),(599.4,13,12),(666,11,11),(999,8,8)
LPDDR2-1066	533	39	874.7	(177.6,52,47),(355.3,23,22),(533,15,14),(710.6,11,11),(888.3,9,9), (1066,7,7),(1243.6,6,6)
DDR3-800	400	25	1024.0	(160,74,64),(240,43,40),(320,31,29),(400,24,23),(480,19,19),(560,16,16),(640,14,14), (720,13,12),(800,11,11),(1200,8,8)
DDR3-1600	800	44	1163.6	(200,65,57),(400,27,26),(600,18,17),(800,13,13),(1000,10,10),(1200,9,8)

combinations. Figure 6 shows the trade-off between area and power consumption for both Aelite (solid line) and Daelite (dash-dotted line) NoCs coupled with the different memories with a service unit size of 64 B. It can be seen that Daelite, which is a circuit-switched NoC has about 20% higher area and 35% power consumption compared to the packet-switched Aelite because of the additional logic required by the slot table in the Daelite router. For both NoCs, the power consumption increases with the gross bandwidth of the memory device because of higher operating frequency and/or interface width requirements. For lower area usage (smaller interface widths and higher operating frequency), the power consumption is higher because of the dynamic power consumption at higher operating frequencies. The power consumption reduces with operating frequency, however, the area increases because of increase in interface width. The area requirement increases linearly with the interface width and hence the leakage power consumption. In addition to leakage power, increase in area increases the dynamic power as well due to increase in the added logic that increases the switching in the circuit.

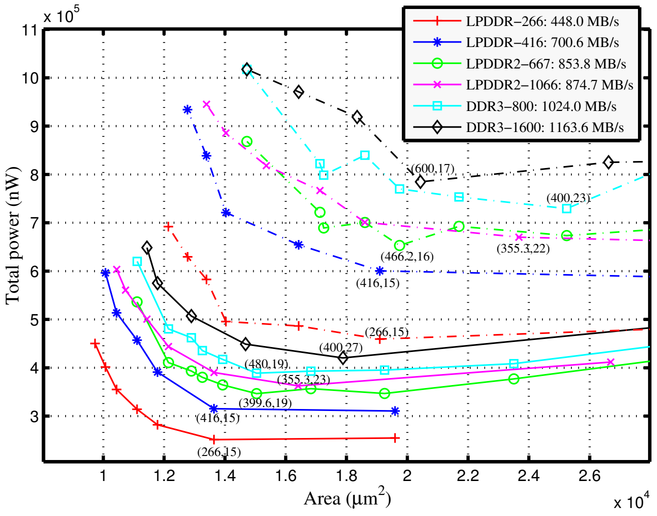

Fig. 6. Area vs. power of Aelite (solid lines) and Daelite (dash-dotted lines) NoCs coupled with different memories with a service unit size of 64 B. The best (f_n, IW_n) combinations for minimal power consumption are shown.

Considering power consumption as the primary optimization criteria, we select the points shown with the (f_n, IW_n) combinations in Figure 6 as the best since further increase in area beyond these points does not reduce the power consumption. We use these configurations for the comparison of the coupled and decoupled architectures in terms of area and power consumption presented in the next section.

C. Comparison between coupled and decoupled NoCs

This section compares the performance of the coupled and decoupled NoC architectures in terms of area, power consumption and guaranteed latency. Since the architectural differences between the decoupled and coupled architectures are in the destination NI, the memory subsystem bus and arbiter, we synthesized these modules independently to find the differences in area and power consumption [2]. For a fair comparison in terms of area and power consumption, we selected the same optimal (f_n, IW_n) combinations from Figure 6 for both architectures. We consider a system consisting of 16 memory clients, and hence, we configured the NI and the DTL bus with 16 ports for the decoupled architecture.

For the coupled architecture, we configured the NI with a single port and a buffer size of one service unit i.e., 64 B, for both request and response paths. For each of the (f_n, IW_n) combinations, we configured both the NIs with IW_n bits and synthesized for a target frequency of f_n MHz. The DTL bus was configured with a width of 32-bits assuming a 16-bit DDR memory device that reads/writes 32-bits every clock cycle, and synthesized with a TDM arbiter for a target frequency of the memory device f_m. For the decoupled architecture, the area and power consumption include the NI, DTL bus and arbiter, and only the NI and buffer in the coupled. The synthesis results of both coupled and decoupled architectures with Aelite and Daelite NoCs and with different memory types are shown in Table II. It can be seen that the coupled architecture consumes much less power and area compared to the decoupled architecture. The savings in area are over 0.068 mm^2 and 0.060 mm^2 and in power are over 1.1 mW and 0.9 mW for Aelite and Daelite NoCs, respectively, with all memory types. The larger savings in Aelite compared to Daelite is because of the larger area usage of its NI when a new port is added compared to the NI of Daelite. To determine the impact of the savings in area and power in a real system, we considered a four-stage NoC tree consisting of 15 routers and NIs. The savings in area and power consumption of the NoC are shown in Figure 7. It can be seen that the savings of 18% and 17% in area, 19% and 11% in power consumption for Aelite and Daelite, respectively, can be achieved by using the coupled architecture.

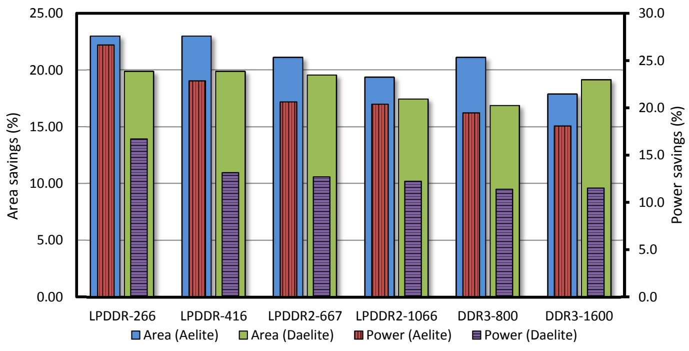

Fig. 7. Area and power savings of the coupled architecture with respect to the decoupled for Aelite and Daelite NoCs and different DRAM types.

To compare the worst-case guaranteed latency of both decoupled and coupled architectures, we assume the same TDM allocation for the NoC in both architectures. For a system consisting of 16 clients with a worst-case bandwidth requirement of each client equal to $1/16^{th}$ of the gross memory

[2]For simplicity, we do not consider the power consumption of the clock sources. However, we assume that the clock source of the coupled architecture consume less power than the multiple sources in the decoupled architecture.

TABLE II. AREA, POWER AND LATENCY COMPARISON OF COUPLED AND DECOUPLED ARCHITECTURES USING AELITE AND DAELITE NOCS WITH DIFFERENT MEMORY TYPES FOR A SERVICE UNIT SIZE OF 64 B

		IW_n (bits)		*Decoupled architecture*						*Coupled architecture*					
				Aelite			*Daelite*			*Aelite*			*Daelite*		
Memory	f_n (MHz)	*Aelite*	*Daelite*	Power (mW)	Area (mm^2)	$\hat{L}_d$ (μs)	Power (mW)	Area (mm^2)	$\hat{L}_d$ (μs)	Power (mW)	Area (mm^2)	$\hat{L}_c$ (μs)	Power (mW)	Area (mm^2)	$\hat{L}_c$ (μs)
LPDDR-266	266.0	15	15	1.38	0.09	4.81	1.21	0.07	4.81	0.26	0.02	2.65	0.29	0.01	2.65
LPDDR-416	416.0	15	15	1.59	0.09	3.08	1.36	0.07	3.08	0.38	0.02	1.70	0.39	0.01	1.70
LPDDR2-667	399.6	19	16	1.62	0.09	2.52	1.44	0.07	2.51	0.44	0.02	1.39	0.43	0.01	1.38
LPDDR2-1066	355.3	23	22	1.61	0.09	2.45	1.36	0.08	2.45	0.43	0.02	1.35	0.45	0.02	1.35
DDR3-800	480.0	19	23	1.75	0.09	2.10	1.43	0.08	2.11	0.49	0.02	1.16	0.49	0.02	1.17
DDR3-1600	400.0	27	17	1.69	0.09	1.85	1.60	0.07	1.83	0.50	0.02	1.02	0.52	0.01	1.00

bandwidth, we consider a TDM wheel of frame size 16 with one slot allocated to each client. Hence, the service latency of a client, Θ_n is 15 service cycles. We consider a four-stage NoC tree, i.e., n_{hops} = 4. The value of δ_{sw} of Aelite and Daelite are 3 and 2 clock cycles, respectively, according to the number of pipeline stages in their routers. Since we consider the worst-case latency for a read, we use a value of 20 cycles for δ_{mem} based on the pipeline stages for a read operation in the RTL implementation of our memory controller [30].

For the decoupled architecture, we assume the same TDM slot allocation in the memory subsystem as in the NoC for simplicity, and hence, $\Theta_m = \Theta_n = 15$ service cycles. Table II shows the guaranteed (read) latency of a client of the decoupled ($\hat{L}_d = \hat{L}_m + \hat{L}_n$) and coupled ($\hat{L}_c$) architectures for different memories and for both NoCs. It can be seen that the guaranteed latency with the coupled architecture is over 44% lower than the decoupled architecture with all memory types. This is due to the double service latency in the decoupled architecture because of its two decoupled arbitration points. However, there is flexibility in selecting any arbiter type in the memory subsystem of the decoupled architecture. Hence, by using a priority-based arbiter and assigning the highest priority to one of its clients, its service latency in the memory subsystem can be reduced to zero as in the case of coupled architecture. However, this will affect the guaranteed latencies of the low-priority clients in the system adversely.

VI. CONCLUSION

Existing NoC and memory subsystems for real-time systems are optimized independently by configuring their arbiters according to the real-time requirements of the clients. However, there exists no structured methodology to select NoC parameters for minimizing area/power consumption when they are used in conjunction. Moreover, the decoupled multiple arbitration points increase the worst-case latency bounds. We proposed a novel methodology to couple any existing TDM NoC with a real-time memory controller and configure the NoC parameters for minimal area and/or power consumption. Coupling the NoC and memory controller using our approach saves over 44% in guaranteed latency, 18% and 17% in area, 19% and 11% in power consumption, for two different NoC types and with different DRAM generations, for a system consisting of 16 memory clients.

ACKNOWLEDGMENT

This research is partially supported by EU grants FP7 288008 T-CREST, FP7 288248 FlexTiles, CA104 Cobra, CA505 BENEFIC, CA703 OpenES, NL grant STW10346 NEST, and Ministry of Education of the Czech Republic under the project number 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] D. Melpignano *et al.*, "Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications," in *In Proc. DAC*, 2012.

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

[5] L. Steffens *et al.*, "Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach," *Proc. ECRTS*, 2008.

[6] 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.

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

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

[9] K. Goossens *et al.*, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. DAC*, 2010.

[10] A. Hansson *et al.*, "aelite: A flit-synchronous network on chip with composable and predictable services," in *Proc. DATE*, 2009.

[11] M. Millberg *et al.*, "The Nostrum backbone-a communication protocol stack for Networks on Chip," in *In Proc. VLSI Design*, 2004.

[12] R. Stefan *et al.*, "dAElite: A TDM NoC Supporting QoS, Multicast, and Fast Connection Set-up," *IEEE Transactions on Computers*, 2012.

[13] M. Schoeberl *et al.*, "A Statically Scheduled Time-Division-Multiplexed Network-on-Chip for Real-Time Systems," in *Proc. NOCS*, 2012.

[14] D. Wiklund *et al.*, "SoCBUS: switched network on chip for hard real time embedded systems," in *Proc. IPDPS*, 2003.

[15] Tilera Corporation, in *http://www.tilera.com/*.

[16] S. Dutta *et al.*, "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems," *IEEE Des. Test. Comput.*, vol. 18, no. 5, 2001.

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

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

[19] J. Nurmi *et al.*, *Interconnect-Centric Design for Advanced SoC and NoC*. Springer US, 2005.

[20] Z. Shi and A. Burns, "Priority Assignment for Real-Time Wormhole Communication in On-Chip Networks," in *Proc. RTSS*, 2008.

[21] J. Sparso *et al.*, "An area-efficient network interface for a TDM-based Network-on-Chip," in *Proc. DATE*, 2013.

[22] X. Jin *et al.*, "Memory Access Aware Mapping for Networks-on-Chip," in *Proc. RTCSA*, 2011.

[23] Y. Gang *et al.*, "Memory-centric Scheduling for Multicore Hard Real-time Systems," *Real-Time Syst.*, vol. 48, no. 6, 2012.

[24] C. Seiculescu *et al.*, "A DRAM Centric NoC Architecture and Topology Design Approach," in *Proc. ISVLSI*, 2011.

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

[26] W. Jang and D. Pan, "Application-Aware NoC Design for Efficient SDRAM Access," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 10, 2011.

[27] M. Daneshtalab *et al.*, "A Low-Latency and Memory-Efficient On-chip Network," in *Proc. NOCS*, 2010.

[28] D. Kim *et al.*, "A network congestion-aware memory subsystem for manycore," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 4, 2013.

[29] B. Akesson *et al.*, "Automatic Generation of Efficient Predictable Memory Patterns," in *Proc. RTCSA*, 2011.

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

[31] H.W.M. van Moll *et al.*, "Fast and accurate protocol specific bus modeling using TLM 2.0," in *Proc. DATE*, 2009.