A Dual-Layer Bus Arbiter for Mixed-Criticality Systems with Hypervisors

Bekim Cilku, Bernhard Frömel, Peter Puschner
Institute of Computer Engineering
Vienna University of Technology
A1040 Wien, Austria
Email: {bekim, froemel, peter}@vmars.tuwien.ac.at

***Abstract*—In mixed-criticality systems, applications with different levels of criticality are integrated on the same computational platform. Without a proper isolation of the different applications of such a mixed-criticality system certification gets expensive, because it has to be shown that application components of lower criticality do not hamper the correct operation of the critical applications. Therefore, all components – even the less critical ones – have to be certified for the highest criticality level. For single core platforms the use of hypervisors promises to shield applications of different criticality from each other. Timing problems may emerge when the hypervisor is ported to a multicore platform where different cores access the global memory concurrently. We show, that full temporal isolation of applications executing on different cores is only achievable if the hypervisor is run on appropriate hardware. The presented dual-layer bus arbiter enables critical applications to preserve isolation properties and also improves the execution performance of non-critical applications.**

Keywords*—mixed-criticality systems; partitioning; multi-core; hypervisor; time predictability; memory hierarchy;

I. Introduction

A common trend in real-time embedded systems is to integrate applications with different levels of criticality on a single shared platform. Applications that are experiencing concurrency on shared resources are prone to temporal and spatial interference. In such cases the execution time of an application depends on the execution of all other concurrent applications. Without proper isolation of the critical applications, the certification process for such a system becomes complex and expensive [1]. A platform that provides spatial and temporal isolation of applications allows independent applications to be developed and certified in isolation. Spatial isolation protects memory elements of one application so they cannot be accessed by any other application. Temporal isolation preserves the timing behavior of applications such that they are not affected from other applications that execute concurrently on the shared platform [2].

Following this direction, in MultiPARTES [1] we use the hypervisor called XtratuM [3] to establish virtual partitions with configurable temporal and spatial properties. In this approach, applications with different levels of criticality can be placed in different partitions and can be certified in isolation. Spatial isolation between virtual partitions is achieved with the help of a Memory Management Unit (MMU). Different parts of the memory address space are assigned to different partitions. The hypervisor sets up a MMU table such that the virtual addresses of each partition are translated to their own dedicated global address space. In this way the MMU table protects the given memory space of one partition against possible violations from any other partition.

Regarding temporal isolation, the hypervisor can ensure full temporal isolation on a single core by using static cyclic scheduling. This policy divides CPU time into slots and statically assigns the slots to the virtual partitions. In this way each partition is activated at a predefined time and gets a specific amount of processor time [3]. Another issue is interrupt handling. The hypervisor masks all interrupts that are not associated with the currently executing partition. For CPUs with cache memory the hypervisor flushes the cache on each context switch in order to avoid cache interference between virtual partitions [4].

In case of multi-core hardware, the hypervisor can still schedule the available CPU time of now multiple cores among partitions. However, temporal non-interference cannot be guaranteed anymore, because problems emerge when concurrent partitions of different cores compete for access to a shared bus. A tempting way to realize the timing isolation for bus accesses is to use a uniform, time division multiple access (TDMA) scheme for all partitions, i.e., each partition accesses the bus in an a-priori determined time slot. Such a uniform bus-access scheme, however, does not honor the diverging access requirements of applications of different criticality and therefore causes an unnecessary and unfavorable performance degradation of mixed-criticality systems.

In this paper we present a bus arbiter for multi-core mixed-criticality systems that preserves isolation properties for critical virtual partitions while allowing for a better bus utilization when non-critical partitions access the bus. The arbiter is based on a hierarchical, two-layer arbitration scheme and switches between two different modes (critical and non-critical mode): In the *critical mode*, critical partitions are allowed to access the bus only within predefined access slots. In the *non-critical mode*, all non-critical partitions are allowed to compete for the access to the bus, which the arbiter grants in a dynamic manner (e.g., priority-, or round-robin based).

The remainder of the paper is organized as follows. The second section describes the hypervisor and the hardware architecture used in MultiPARTES. In section three we describe the arbiter for mixed-criticality systems. Experiments

[1] www.multipartes.eu

are shown in section four. Section five presents related work. The paper finishes with a conclusion and an outlook on future work (section six).

II. MultiPARTES architecture description

This section provides background information on XtratuM and the hardware used in MultiPARTES (cf. Figure 1).

XtratuM is an open source bare-metal hypervisor intended for embedded real-time systems. It uses a para-virtualization technique to be as close as possible to native hardware [5]. The interface between applications and the hardware is managed by a set of hypercalls that transfer control to the hypervisor. The allocation of the available hardware resources to partitions is defined statically by a configuration file. Resource quotas are assigned according to the needs of each partition. These quotas direct the scheduling and control the memory areas and communication ports that applications use. The configuration file also specifies the scheduling plan. It divides time into time slots and statically assigns these time slots to virtual partitions. This policy creates one major time frame (MAF) which is repeated periodically. During run-time, XtratuM assigns a processor core to the corresponding partition on each time slot and ensures that each partition gets only the pre-specified amount of processor time. For partitions that host multiple applications, the partition is responsible of implementing its own, internal scheduling in a transparent way to the hypervisor (hierarchical scheduling) [5]. XtratuM also provides a Health Monitor as a mechanism to detect and manage unexpected events. The Health Monitor aims at identifying faults and takes a set of actions predefined in the configuration upon the occurrence of a fault.

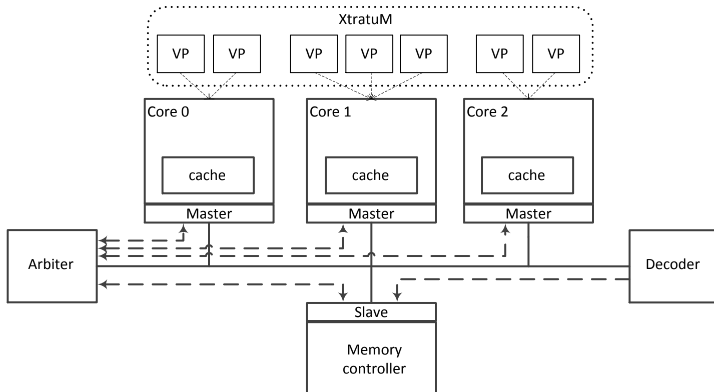

Fig. 1: MultiPARTES architecture

The hardware used in MultiPARTES consists of a multi-core LEON3 processor that is interconnected with a memory controller and I/O resources through a shared AMBA bus. LEON3 is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. Multi-core LEON3 systems are highly customizable with respect to processor core and periphery features. LEON3 has a seven-stage pipeline with Harvard architecture, separate caches for instructions and data, a hardware multiplier and divider, on-chip debug support, an MMU with a configurable TLB and multi-processor extensions [6].

The Advanced Microcontroller Bus Architecture (AMBA) is an open standard specification that defines an on-chip communication standard for designing high-performance embedded systems [7]. It is widely used in network interconnect chips, RAM controllers, Flash memory controllers and SoCs (System on Chips) [8]. It consists of a high-performance system backbone bus (AHB) on which the CPUs, on-chip memories and other DMA devices reside, and a low bandwidth bus (APB) where most of the system peripheral devices are located. The APB is optimized for low power consumption and reduced interface complexity. The AHB and the APB are connected via a bus bridge.

An AMBA AHB system consist of the following main components:

- Master - component that initiates read or write operations;
- Slave - component that responds to read and write operations;
- Arbiter - component that regulates bus accesses;
- Decoder - component that performs a centralized address-decoding function;

All master/slave components are connected through the following physical lines:

- Address bus - used to transmit the address from master to decoder;
- Data bus - to transmit data on separate buses for read and write operations;
- Control bus - to provide control signals and information about the state of a transfer;

To connect master and slave components, AMBA uses a central multiplexer controlled by a centralized arbiter and decoder. The arbiter determines which master gets the right to commence a transfer and also ensures that at any given time at most a single transfer is in progress. The criterion used to determine which master gains access to the bus is called the *arbitration scheme* [9]. Before a master is able to perform a transfer on the bus, the arbiter needs to grant the master the access to the bus. This process is started with an assertion of the *request signal* from the master to the arbiter. When the bus becomes available, based on the arbitration algorithm, the arbiter selects one master and sends a *grant signal* to indicate that the access to the bus is granted. The basic requirements for an arbitration policy are that it should guarantee fairness of accesses between master components and it has to prevent the starvation of a master. The arbitration scheme can be static or dynamic. In a static arbitration scheme, bus access allocation is done in a predetermined way (as planned during design time). In contrast, dynamic arbiters make decisions about bus accesses during runtime.

Figure 2 displays the AMBA arbiter signals. The description of the signals are as follows:

- HBUSREQx - this signal indicates that the bus master *x* requests access to the bus;
- HLOCKx - this signal indicates to the arbiter that the master *x* requires a locked access to the bus;
- HGRANTx - this signal is an output of the arbiter and indicates that the master *x* has been granted access to the bus;
- HMASTER[3:0] - these signals indicate which master is currently performing a transfer;
- HMASTLOCK - this signal indicates that the current master performs a locked transfer;
- HADDR[31:0] - these signals indicate the address where read or write transfers take place;
- HBURST[2:0] - these signals indicate how many transfers are required by the bus;
- HTRANS[1:0] - these signals indicate the type of the current transfer;

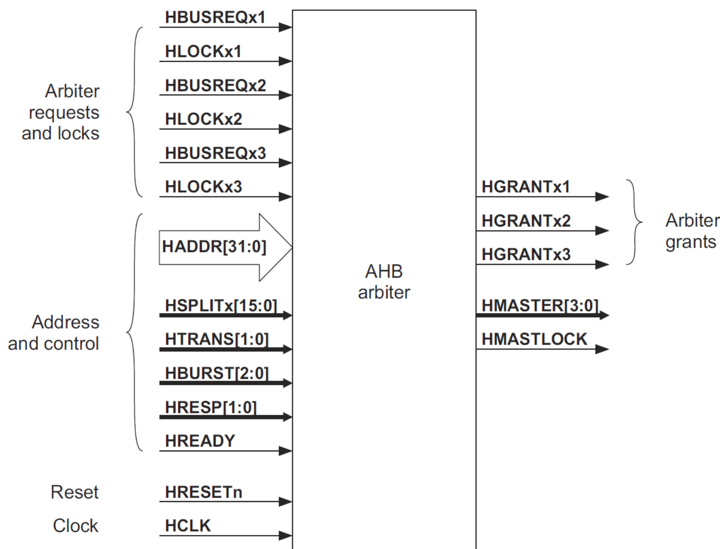

Fig. 2: AMBA arbiter [7]

III. Dual-layer bus arbiter for mixed-criticality systems

The new bus arbiter uses a two-layer arbitration scheme (cf. Figure 3). The first layer is based on *time division multiple access* (TDMA) and is responsible for guaranteeing the temporal properties of critical partitions, while the other layer employs a *round-robin* (RR) arbitration policy that controls the bus access only for non-critical partitions. TDMA is an arbitration policy that guarantees a fixed bus bandwidth by a-priory assigning a time slot of fixed length to each master. In contrast, the RR arbitration scheme grants bus access to every master on the bus in a circular manner. A master relinquishes control over the bus when it no longer has data to transfer. When a master has completed its transfer, it passes control to the next master in line [9].

In the dual-layer bus arbiter the access time to the shared bus is partitioned in a periodic scheduling frame that consists of TDMA slots. Each critical partition has one or more exclusive TDMA slots. Non-critical partitions share a set of sequential TDMA slots for dynamic arbitration. Figure 3 shows that the critical partitions on core0 and core1 have predetermined time slots in the scheduling frame. Whenever the critical partition generates a request for a bus access, the dual-layer bus arbiter delays the grant until the time slot of the partition has arrived. If a critical partition issues a bus-access request after a TDMA slot of this partition has already started, the arbiter delays the grant until next TDMA slot of that partition arrives (see Figure 3: In case core0 requests a bus access after the first slot has already started, core0 still has to wait until the start of the second slot). If the bus arbiter would not implement this strict access scheme, late data transfers could overlap with the next TDMA slot reserved for another partition, thus violating the temporal isolation between the slots. This is the reason why the requests of critical partitions are only served if they are present at the beginning of a slot. Individual partitions that need higher bandwidth can be scheduled more often by placing multiple slots within one scheduling frame. We know the duration of one slot and the length of the scheduling frame, thus we can derive the bandwidth and the maximum latency for each critical partition, which makes the system predictable.

Part of the TDMA slots are used for non-critical partitions where we allow dynamic arbitration as follows. We dedicate a few sequential TDMA slots to the RR arbiter (cf. Figure 3, Dynamic arbitration). During these TDMA slots, the RR arbiter is active and all non-critical partitions can compete for the bus. The activation and deactivation of RR arbitration is controlled by the TDMA arbitration scheme. To ensure that non-critical partitions cannot effect time properties of critical ones, an additional TDMA slot is appended after the sequence of TDMA slots dedicated for dynamic arbitration. In this additional TDMA slot only ongoing transfers are allowed to finish, but no new bus requests from any partition is served. Otherwise, bus transfers from non-critical partitions started in this slot could interfere with possible bus requests allocated to the subsequent critical-partition slot.

A. Arbiter architecture

The arbiter architecture consists of the TDMA and the RR arbiter (Figure 4). TDMA has a wrap incremental counter, a shift register and a controller. The wrap counter is incremented at each clock cycle with a maximal value equal to number of cycles for one slot. The function of this unit is to shift the token in a shift register at the end of the time slot. The shift register keeps records on slots. Depending on the active slot, the controller gives a grant to an associated critical request (HBUSREQ0 or HBUSREQ1) or activates/deactivates the RR arbiter.

All request signals from non-critical partitions are connected to the RR arbiter queue. When the RR arbiter is activated, it grants partitions bus access in the same order as the requests have arrived (HBUSREQ2 or HBUSREQ3).

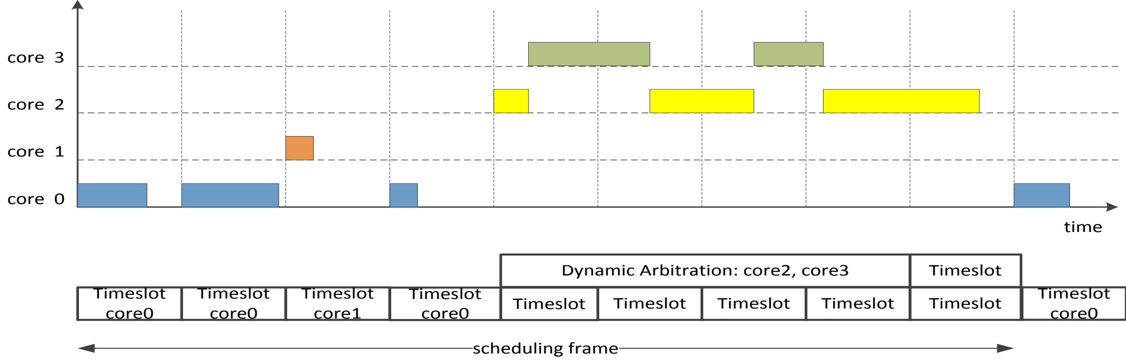

Fig. 3: Dual-policy arbitration scheme

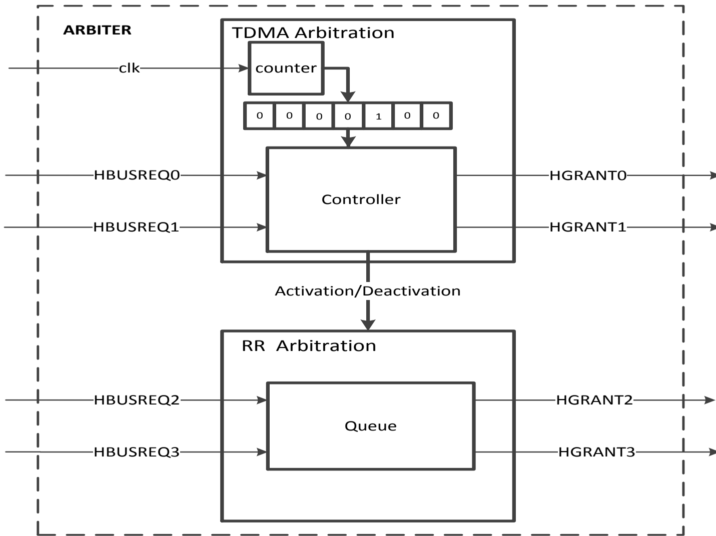

Fig. 4: Arbiter architecture for mixed-criticality systems

IV. Evaluation

We implemented and deployed the dual-layer bus arbiter on an FPGA development kit (terasic DE2-115) to demonstrate the feasibility of the solution and to evaluate its performance. The arbiter is written in VHDL language and deployed as part of the Grlib IP library [10]. We used grlib-gpl-1.3.4-b4140 version [10] for the SoC with four LEON3 cores. The cores were all clocked at 50MHz. Each core had a dedicated 16kB instruction cache and 16kB data cache. Each CPU is controlled and debugged individually through the Debug Support Unit (DSU). The interconnection with the memory controller and the I/O resources is done through the AMBA bus. The bus is configured not to support split transactions.

For this evaluation a bubble sort algorithm was chosen. It sorts vectors of size from 100 to 1000 elements. The execution is done on a bare-metal processor, without any operating system in order to avoid the overhead generated from the OS. All four cores were executing the same code. The goal was to compare execution performances of noncritical partitions by using three different bus arbitration policies: TDMA, RR and Dual-Layer (DL). The TDMA scheduling frame consisted of four slots and each slot was assigned to a core. The DL arbiter used five slots where the critical CPUs (CPU0 and CPU1) were assigned to the first two slots (slot 0 and 1) while the non-critical CPUs were allowed on the following two slots. No arbitration was allowed on the fifth slot (see above). The slot size was 20 clock cycles (the longest transfer was taking 18 cycles).

Figure 5 shows the execution time of the bubble sort algorithm for different vector sizes. The execution time of non-critical partitions is illustrated for TDMA, RR and DL. From the results we see that the execution time of the algorithm in non-critical cores with DL arbitration is 3.1 time shorter than in a system with pure TDMA. Compared to pure RR, the execution of bubble sort with DL lasts 1.2 times longer. For critical partitions the execution time with TDMA and DL is the same. This experiment suggests that DL arbitration improves the performance of non-critical partitions without affecting the time properties of the critical ones.

V. Related Work

Most of the research in multi-core mixed-criticality systems is focused on software-based scheduling [11]. There are a few studies that examine temporal and spatial properties at the hardware level.

Akesson et al. [12] presented a memory controller for SDRAM called *Predator* that provides a guaranteed minimum bandwidth and maximum latency bound. The controller combines elements of statically and dynamically scheduled memory controllers. Command-operation time for read or write are fixed and can be precomputed at design time. These operations obey all SDRAM timing constraints and have fixed latencies. Arbitration uses credit controller static priority

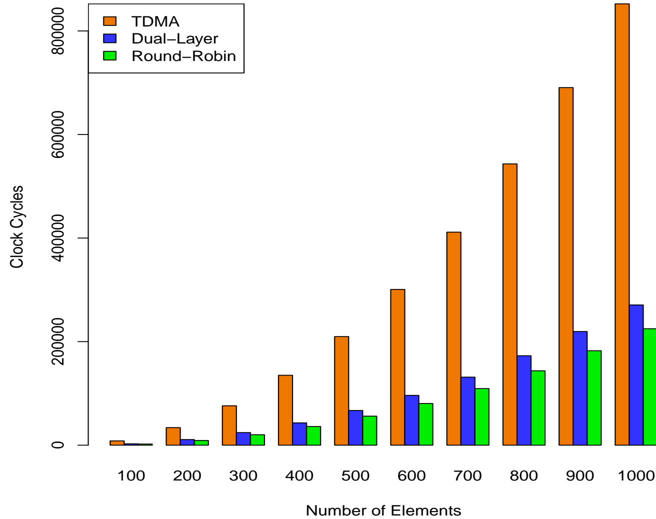

Fig. 5: Execution time of bubble sort algorithm

(CCSP) scheduling to regulate the rate of requests and to guarantee maximum bandwidth. This arbitration policy allows a decoupling of latency and rate for each client. All hard real-time responses are delayed to their worst-case latency to provide the temporal isolation between requesters.

Paolieri et al. [13] built a memory controller that enforces the upper-bound delay for critical tasks to not be violated. This is a main feature that makes this controller analysable. The arbitration is split into two levels, one that schedules requests among different cores (inter-core bus arbiter) and the other one that schedules the thread requests from the same core (intra-core bus arbitration). Critical tasks are always with higher priority than non-critical tasks. The WCRT is calculated by using the maximum possible time needed for a single transfer multiplied by the number of possible other critical tasks that can generate requests at a given moment.

The approach of Reineke et al. [14] for a DRAM controller for the PRET machine is based on a multi-thread concept. The processor implements a four thread-interleaved pipeline and a separate bank is assigned to each thread. Accessing DRAM in this way removes the conflicts between threads considering banks as a non-shared memory and also reduces the WCET by interleaving bank accesses. The arbitration is simple round-robin.

Audsley [15] proposes a separated shared memory-tree architecture developed for a NoC architecture. The tree consists of 2-to-1 multiplexers with CPUs as a leaves and main memory as a root. A priority-aware protocol is used to pass the requests and data through the tree.

VI. Conclusion

Applications with different time criticality use different design and verification methods. Using a hypervisor in mixed-criticality systems can reduce the price of the certification process by providing appropriate isolation between virtual partitions. However, the hypervisor is not able to ensure temporal isolation of the critical partitions when they access a shared bus. Implementing the dual-layer arbiter helps the hypervisor to guarantee time bounds for critical applications and also improves performance of non-critical applications when they access a shared bus.

Acknowledgment

This work has been supported in part by the European Community's Seventh Framework Programme [FP7] under grant agreement 287702 (MultiPARTES) and the EU COST Action IC1202: Timing Analysis on Code Level (TACLe).

References

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

[2] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "Flexpret: A processor platform for mixed-criticality systems," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS). IEEE*, 2014.

[3] M. Masmano, I. Ripoll, A. Crespo, and J.-J. Metge, "Xtratum: a hypervisor for safety critical embedded systems," in *11th Real-Time Linux Workshop*, 2009.

[4] M. Masmano, I. Ripoll, S. Peiró, and A. Crespo, "Xtratum for leon3: an open source hypervisor for high integrity systems," in *European Conference on Embedded Real Time Software and Systems. ERTS2*, vol. 2010, 2010.

[5] E. Carrascosa, M. Masmano, P. Balbastre, and A. Crespo, "Xtratum hypervisor redesign for leon4 multicore processor."

[6] A. Gaisler and S. Göteborg, "Leon3 multiprocessing cpu core," *Aeroflex Gaisler, February*, 2010.

[7] A. A. Specification, "Multi layer ahb specification,(rev2. 0)," 2001.

[8] Y. Godhal, K. Chatterjee, and T. A. Henzinger, "Synthesis of amba ahb from formal specification: a case study," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 5-6, pp. 585–601, 2013.

[9] S. Pasricha and N. Dutt, *On-chip communication architectures: system on chip interconnect*. Morgan Kaufmann, 2010.

[10] Leon3. [Online]. Available: http://www.gaisler.com/

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

[12] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: a predictable sdram memory controller," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '07. New York, NY, USA: ACM, 2007, pp. 251–256. [Online]. Available: http://doi.acm.org/10.1145/1289816.1289877

[13] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for wcet analysis of hard real-time multicore systems," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 57–68.

[14] J. Reineke, I. Liu, H. Patel, S. Kim, and E. Lee, "Pret dram controller: Bank privatization for predictability and temporal isolation," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, 2011, pp. 99–108.

[15] N. C. Audsley, "Memory architectures for noc-based real-time mixed criticality systems," *Proc. WMC, RTSS*, pp. 37–42, 2013.