Project Number 288008

D 4.3 Dynamic memory controller concepts

Version 1.0
18 September 2012
Final

Public Distribution

Eindhoven University of Technology, Technical University of Denmark

Project Partners: **AbsInt Angewandte Informatik**, **Eindhoven University of Technology**, **GMVIS Skysoft**, **Intecs**, **Technical University of Denmark**, **The Open Group**, **University of York**, **Vienna University of Technology**

Project Partner Contact Information

AbsInt Angewandte Informatik Christian Ferdinand Science Park 1 66123 Saarbrücken, Germany Tel: +49 681 383600 Fax: +49 681 3836020 E-mail: ferdinand@absint.com	**Eindhoven University of Technology** Kees Goossens Potentiaal PT 9.34 Den Dolech 2 5612 AZ Eindhoven, The Netherlands E-mail: k.g.w.goossens@tue.nl
GMVIS Skysoft João Baptista Av. D. Joao II, Torre Fernao Magalhaes, 7 1998-025 Lisbon, Portugal Tel: +351 21 382 9366 E-mail: joao.baptista@gmv.com	**Intecs** Silvia Mazzini Via Forti trav. A5 Ospedaletto 56121 Pisa, Italy Tel: +39 050 965 7513 E-mail: silvia.mazzini@intecs.it
Technical University of Denmark Martin Schoeberl Richard Petersens Plads 2800 Lyngby, Denmark Tel: +45 45 25 37 43 Fax: +45 45 93 00 74 E-mail: masca@imm.dtu.dk	**The Open Group** Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 Fax: +32 2 675 7721 E-mail: s.hansen@opengroup.org
University of York Neil Audsley Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 325 500 E-mail: Neil.Audsley@cs.york.ac.uk	**Vienna University of Technology** Peter Puschner Treitlstrasse 3 1040 Vienna, Austria Tel: +43 1 58801 18227 Fax: +43 1 58801 918227 E-mail: peter@vmars.tuwien.ac.at

Contents

Document Control

Version	Status	Date
0.1	First draft	9 September 2012
1.0	Final version	18 September 2012

Executive Summary

This document describes the deliverable *D 4.3 Dynamic memory controller concepts* of work package 4 of the T-CREST project, due 12 months after project start as stated in the Description of Work. This document explains the problem of increasingly dynamic memory request streams in firm real-time systems. We then survey related work on real-time memory controllers and conclude that existing controllers are either too static, both in terms of implementation and performance analysis, to efficiently cope with dynamic memory streams (e.g. requests with variable sizes and alignments), or too dynamic and fail to provide firm bounds on bandwidth and response times.

To address this problem, the concepts behind a dynamically scheduled memory controller and corresponding performance analysis are introduced along with a step-wise trajectory towards its realization. The goal of this work is to provide guarantees at least equal to those of the existing memory controller for requests with fixed size and alignment and outperform it in presence of more dynamic request streams.

1 Introduction

The complexity of embedded systems is increasing, as a growing number of concurrently executing applications are integrated on a single chip. These applications are mapped on multi-processor systems-on-chips (MPSoC) that strike a good balance between performance, cost, power consumption and flexibility [14, 27, 12, 15]. Some applications have *real-time requirements*, such as a minimum throughput of video frames per second, or a maximum response time for processing those video frames.

To reduce cost, platform resources, such as processors, interconnect, and memories, are shared between applications. However, resource sharing causes *interference* between applications, requiring the resources to be shared in a predictable manner to enable bounds on response times to be computed. This is particularly challenging for shared SDRAM memories for three reasons: 1) Applications have become increasingly dynamic and input dependent, resulting in more diversified memory traffic that is difficult to characterize at design time. 2) The execution times of requests and the bandwidth offered by the memory are *highly variable* and traffic dependent [2]. 3) For cost reasons, SDRAM bandwidth is a *scarce resource* that must be efficiently utilized [28, 15, 8].

Applications may have different types of real-time requirements, depending on the nature of the processing and the degree of pipelining in the processing cores. These application requirements are reflected in the requirements of their corresponding memory clients. For memory clients of throughput-driven applications, such as streaming media decoders, bandwidth requirements are most important. Conversely, clients of latency-driven applications, e.g. in the control domain, primarily have response time requirements. A contemporary complex MPSoC typically have many applications with different types of requirements and a versatile memory controller must hence be able to satisfy both bandwidth and response time requirements at the same time.

The requirements exist in three different classes, depending on the time-criticality of the application. Applications, such as a Software-Defined Radio [18], have *firm real-time requirements* (FRT). Failure to satisfy the requirements of their memory clients is highly undesirable and may result in failure to comply with a given standard, or even violate the functional correctness of the MPSoC [26, 10]. Other applications, such as media decoders, have *soft real-time requirements* (SRT). Missing a soft deadline results in quality degradation of the application output, such as causing visual artifacts in decoded video. Although this is perceived as annoying by the user, it may be acceptable as long as it does not occur too frequently. There are also applications with *no real-time requirements* (NRT), such as a graphical user interface. Their memory clients do not have well-defined requirements, but must still be served fast enough for the application to be perceived as responsive by the user.

The goal of the T-CREST project is to develop a time-predictable system that is easy to analyze and offers good worst-case behavior. This work hence focuses on applications with firm real-time requirements on bandwidth and response times.

Current real-time memory controllers typically rely on a rigid characterization of the memory traffic at design time to satisfy firm real-time requirements. If the provided traffic does not comply with the assumptions, the bounds on bandwidth or response times provided by the controller may either be inefficient or invalid, potentially resulting in failure to satisfy the real-time requirements.

This section provides brief background information to motivate the work outlined in this document. First, we present a problem related to increasing dynamism in the off-chip memory traffic of embedded real-time systems that is not solved by current memory controllers to highlight the requirement of a flexible real-time memory controller with a powerful corresponding performance analysis. We then proceed by elaborating on real-time requirements of applications and focus on the subset of these addressed by the T-CREST project.

The problem addressed by this work is hence to provide a flexible firm real-time memory controller with a powerful corresponding performance analysis to manage the increasing dynamism in the off-chip memory traffic of embedded real-time systems.

The rest of this document is organized as follows: Section 2 discusses related work on memory controllers and explain why existing designs do not satisfy the problem considered in this document. Section 3 then presents our proposed approach to develop a time-predictable dynamically scheduled memory controller with a flexible corresponding performance analysis. Section 4 then proceeds by reviewing the requirements of the T-CREST memory work package and explains how the proposed design relates to them, before we finally conclude the document in Section 5.

2 State of the Art

This section discusses the state of the art in real-time memory controllers and explains why they are unable to provide firm real-time requirements to applications with dynamic behavior in a scalable manner. We first discuss memory controllers designed for firm real-time systems, before elaborating on soft real-time controllers.

2.1 Firm Real-Time Controllers

Firm real-time memory controllers are suitable for memory clients with FRT requirements, whose bandwidth and/or response time requirements must be satisfied even in the worst-case scenario. The goals of these memory controllers are typically to maximize the guaranteed net bandwidth to all memory clients and/or minimize their guaranteed response times. This is challenging, as both bandwidth and response times are highly variable and traffic dependent, and hence difficult to bound tightly.

A problem for FRT memory controllers is that they are typically unable to exploit locality, since locality is difficult to guarantee even for the memory accesses of a single application, and more or less impossible for interleaved memory accesses from a set of concurrently executing applications. This results in bounds on bandwidth and response times that are far away from their average values, making it expensive to provide FRT performance guarantees for general applications. To minimize this cost, some FRT memory controllers employ a *close-page policy* [23], which means that rows are closed immediately after a burst [9, 21, 22] or set of bursts [1] is finished to minimize the overhead of opening another row in the same bank.

Most SDRAM controllers can be classified as either statically or dynamically scheduled, depending on if the sequence of SDRAM commands sent to the memory is determined at design time or at run time. Statically scheduled controllers, such as [4], execute precomputed schedules of SDRAM commands that have been computed at design time. These controllers are suitable for memory clients with FRT requirements, since bandwidth and response times can be bounded at design time by analyzing the schedule. The main drawback of these controllers is that their applicability is limited to single applications whose memory behavior in terms of request sizes, alignments of requests, direction (read/write), can be exactly specified at design time. Since this is typically not the case in our considered complex real-time MPSoCs, these controllers do not solve the problem addressed in this document. However, note that such an application-specific controller can exploit locality, since knowledge about the exact memory access trace enables rows to be opened and closed in an optimal manner, resulting in a highly efficient, although very restrictive solution.

Dynamically scheduled memory controllers, on the other hand, schedule SDRAM commands at run time based on available requests. The challenge with dynamically scheduled FRT controllers, such as [21, 24], is to analyze the behavior of the command scheduler for requests with arbitrary and variable sizes and alignments, which is required to bound bandwidth and response times. This is why these controllers and other FRT controllers [9, 1, 22] assume requests to have fixed size and alignments. If the memory traffic does not fit with these assumptions, the memory controller may break and/or the provided bounds on bandwidth or response times may be invalid. As a result, these dynamically scheduled firm real-time controllers do not cover the problem addressed in this document.

A hybrid memory controller is proposed in [1] that combines aspects of both statically and dynamically scheduled approaches. The hybrid concept is based on *predictable memory patterns*, which are statically computed sequences (sub-schedules) of SDRAM commands. These patterns are dynamically scheduled at run time, based on the incoming requests. The memory patterns exist in five flavors: 1) read pattern, 2) write pattern, 3) read/write switching pattern, 4) write/read switching pattern, and 5) refresh pattern. The patterns are *automatically generated* [3] at design time based on the timing constraints of the particular SDRAM device and the bandwidth and response time requirements of the memory clients [11]. The benefits of this solution is that it lifts the abstraction level from scheduling individual highly inter-dependent SDRAM commands to patterns, which are sets of commands, that are relatively independent. This makes both memory scheduling and performance analysis easier.

Bandwidth in the hybrid memory controller is bounded by determining the worst-case sequence of memory patterns that can be scheduled by the memory controller. This is easily done since the scheduling rules of the patterns are very straight-forward and the length of all patterns and the amount of data they transfer are fixed and known at design time. Response times are bounded in a similar way by knowing the maximum number of interfering requests that can be scheduled by the predictable memory controller arbiter and determining the worst-case sequence of patterns these requests can map to [2].

The main problem with the hybrid controller approach in the context of dynamic applications is scalability. Increasing dynamism requires more types of patterns to be computed at design time and stored in a memory in the memory controller. Examples of this would be read patterns and write patterns that transfer different amounts of data to address variable request sizes. A consequence of this would be that each set of read and write pattern may require different length refresh patterns and each combination of read and write patterns may require different switching patterns to stay efficient. This introduces a lot of dependencies between requests that must be considered by the memory scheduler, defeating the purpose of the pattern-based approach.

2.2 Soft Real-Time Controllers

Memory controllers for SRT requirements are dynamically scheduled controllers that target maximizing bandwidth to optimize cost by accommodating more memory clients with a given memory, and minimize response times to satisfy requirements of latency-driven memory clients. Note that this type of memory controller considers *average* bandwidth and response times rather than the worst-case counterparts. This is because SRT requirements are typically verified by extensive simulation instead of using the formal analysis techniques employed for FRT requirements. It is hence sufficient to assert that application-level deadlines, which may be at the granularity of thousands of memory requests, are satisfied with high probability, rather than providing absolute guarantees at the level of individual requests [26].

Although this class of controllers can have a variety of policies for opening and closing rows [23]. Open-page policies are common, since locality in the memory traffic does not have to be guaranteed. There only has to be enough locality to offset the penalty of row misses with high probability. The average memory efficiency of an SRT/NRT controller can hence be much higher than for their FRT counterparts. However, as the number of concurrently executing applications sharing a single off-chip memory is increasing, there is more and more competition for the open rows in the memory, possibly resulting in thrashing that reduces performance. For this reason, close-page policies are gaining in popularity with multi-programmed workloads.

A benefit of SRT controllers is that they support any type of memory traffic. They are hence very general components, applicable to many systems. The flexibility of these controllers stems from that they are not concerned about formal analysis and providing bounds on bandwidth and response times, and dynamically schedule memory accesses at the level of single SDRAM bursts, rather than as sets of interleaved bursts. Scheduling at this finer level of granularity also reduces the average response times for high-priority clients by reducing blocking effects from clients with lower priorities. However, since no bank-parallelism is guaranteed between consecutive bursts, these memory controllers are unable to guarantee high memory efficiency in the worst case. For example, a typical controller for a 16-bit DDR3-800 memory with a burst length of 8 words only guarantees 16% of the peak bandwidth in case all requests are writes and there is a row miss for every burst. This bound is determined by the minimum time between opening two pages in the same bank. It is hence derived from the memory specification rather than the specifics of the memory controller, and applies by default to many SRT memory controllers.

SRT memory controllers often feature sophisticated mechanisms to improve the average bandwidth or reduce average response times. Examples that improve memory efficiency involve preference for requests that target open rows in the memory banks [19, 25, 17, 29, 16], or that fit with the current direction (read/write) of the data bus [13, 6, 17, 29, 16]. Example mechanisms that reduce average response times of applications prefer reads over writes [25], which is beneficial if reads are blocking while writes are posted, or let high-priority memory clients preempt lower priority clients [16]. Another technique to reduce latency is presented in [19] that schedules memory bursts belonging to the same requests simultaneously thereby unblocking the stalling processor earlier. It is also proposed in [20] to try to schedule refresh operations during idle cycles when there are no requests pending or even executing multiple refresh operations in sequence when idle to amortize overhead [5]. To further improve performance, Intel proposed an adaptive page policy [7] that dynamically switches between open- and close-page policies based on the locality in the memory steams. Although these mechanisms help SRT memory controllers fulfill their goals, the interactions between these mechanisms and dynamic command schedulers are complex, making it difficult to derive useful bounds on response times, and even to show that the 16% bound on bandwidth applies. For this reason, current SRT controllers do not satisfy the problem considered in this work.

We hence conclude that there is currently no memory controller with a corresponding performance analysis that provide efficient bounds on bandwidth and response times to FRT memory clients with variable request sizes and alignments in a scalable manner. This is the motivation for this work.

3 Approach to Dynamic Command Scheduling

Having establish why existing memory controller designs and analyses are insufficient for the problem described in this document, we proceed by outlining our approach to address the issue. First, we provide an overview by explaining our starting point and general idea. We then continue by discussing the design of a dynamically scheduled memory controller back-end and its corresponding performance analysis.

3.1 Overview of Approach

The starting point for this work is the hybrid memory controller discussed in Section 2.1. The main reasons for this choice are that it is a time-predictable architecture with available implementations in both SystemC and VHDL and state-of-the-art analysis implemented in an accompanying tool-flow. This enables the proposed dynamic command scheduling and analysis to build on existing work, speeding up development and reducing risk. The approach used in this work hence implies addressing the scalability problem of the existing hybrid memory controller by introducing a dynamic scheduler and generalize the existing analysis to manage the increased flexibility of the memory controller design, such as efficiently supporting requests with variable size and alignment.

The architecture of the hybrid memory controller is shown in Figure 1. It is divided into a *front-end* and a *back-end*. The front-end is independent of memory technology and contains components to implement time-predictable and composable resource sharing. The back-end interfaces with the actual memory device and makes it into a predictable resource. The back-end is hence different for different types of memories, such as SRAM and SDRAM. In the case of an SDRAM, the back-end is responsible for memory mapping, i.e. translating logical addresses used by software to physical addresses inside the memory device. It also takes care of command generation and scheduling, which in case of the hybrid memory controller implies determining which memory pattern to use and issue its commands. The modularity of this architecture provides an excellent starting point for this work, as it implies developing a new dynamically scheduled SDRAM back-end that replaces the existing one while reusing the complete front-end.

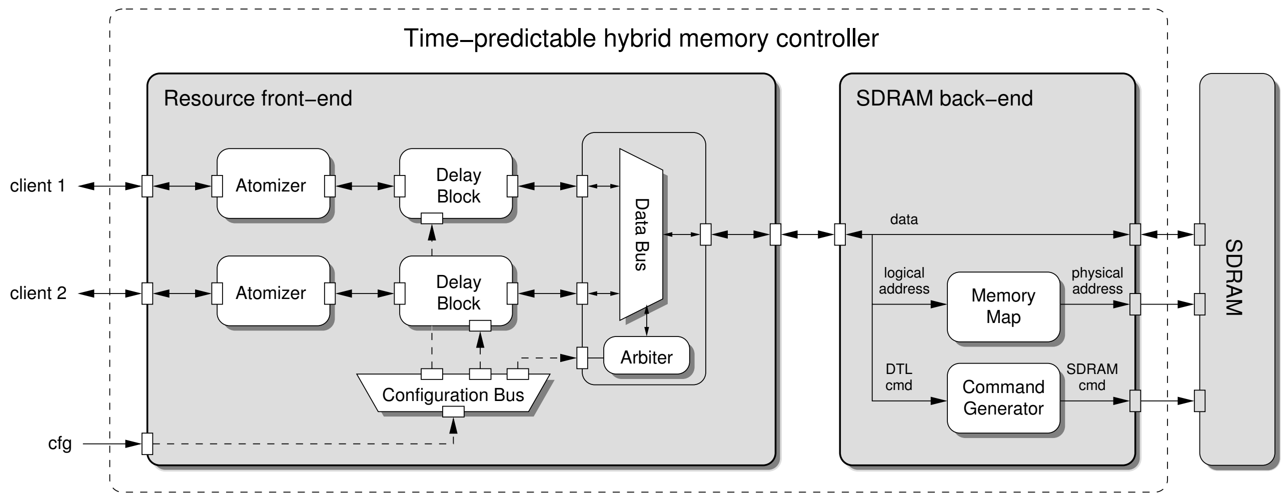

Figure 1: Architecture instance of hybrid memory controller, supporting two clients.

3.2 Dynamically Scheduled Back-End

This section proceeds by discussing the design of the dynamically scheduled back-end that replaces the pattern-based back-end in the hybrid architecture. There are three main goals for its design and implementation: 1) It must be time-predictable and provide bounds on bandwidth and response times that are at least as good as those of the current hybrid controller for fixed request sizes, and better for variable sizes. 2) The area of the implementation should be small to fit comfortably on an FPGA instance of a multi-processor system and synthesize at high frequencies to keep up with fast DRAM memories. 3) To fit with the work on reconfigurable memory controllers done within the T-CREST project, it must furthermore reduce the space required to store configuration data and reduce the (re)configuration time, compared to the hybrid design.

The architecture of the proposed dynamically scheduled back-end is shown in Figure 2. Incoming requests are sent to the command generator that performs memory mapping and generates an appropriate set of SDRAM commands (activates, reads, writes, and precharges) and stores them into queues depending on the targeted memory bank. Refresh commands are generated periodically by the command generator and are stored into a separate queue.

The command scheduler is responsible for looking at the set of available commands and scheduling them in a manner that does not violate the timing constraints of the memory device, which are stored in a register bank. At any given time, there may be many commands whose timing constraints are satisfied, giving the scheduling algorithm space to optimize for different goals. In the T-CREST project, this goal is time-predictability and providing efficient bounds on bandwidth and responses of requests. Three time-predictable off-line scheduling algorithms have been evaluated by the existing hybrid memory controller [3], branch and bound scheduling, as-soon-as-possible (ASAP) scheduling, and bank scheduling. While the branch and bound algorithm is not suitable for on-line implementation in a dynamic scheduler, the other two algorithms might be. Initial work on a dynamic scheduling algorithm will hence evaluate ASAP scheduling and bank scheduling in the dynamic context. If these algorithms are shown to be insufficient for the efficiency and synthesis goals of this work, new time-predictable scheduling algorithms will be developed and evaluated as a part of this work.

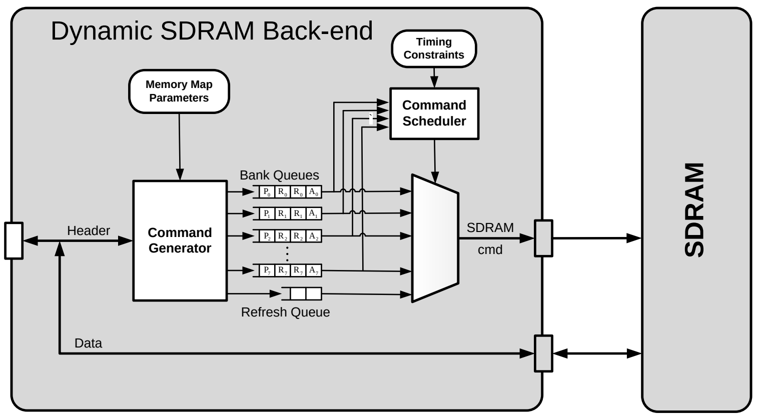

Figure 2: Architecture of proposed dynamically scheduled back-end.

With respect to the reconfiguration goals of the design, the proposed design stores configurations as sets of integers related to memory mapping and timing constraints. This is considerably less data than complete sets of memory patterns, which is expected to reduce the reconfiguration time considerably.

The dynamic memory controller back-end will be designed in two steps with an increasing level of dynamism and complexity. Initially, only a single memory request will be admitted to the back-end, just like in the earlier hybrid design, to reduce the number of options for the command scheduler. Once the implementation and analysis of this step is complete, multiple requests will be admitted into the back-end, although FIFO ordering between them is maintained. This enables increased pipelining of requests that improves efficiency at cost of increased analysis complexity.

3.3 Performance Analysis

The main challenge with a time-predictable dynamically scheduled memory controller is to develop an analysis that captures the flexibility of the architecture, while still providing tight and efficient bounds on bandwidth and response times. The analysis of the proposed back-end extends on the analysis of the hybrid memory controller design. To maximize the bound on bandwidth and minimize bounds on response times, it will assume a close-page policy. This choice of page policy is also beneficial for analysis, as it makes requests relatively independent. The development of the analysis will be carried out in two steps with increasing complexity.

In the first step, we assume fixed request sizes and try to reuse the existing analysis tools for the hybrid memory controller. This enables early indication on whether or not the dynamically scheduled approach delivers on the requirement to outperform the hybrid controller with the same assumptions on traffic. This step may involve making improvements to the existing analysis tool to better capture aspects of the dynamically scheduled back-end.

The second step introduces variable request sizes into the analysis. To improve beyond the existing analysis tools, requests of different sizes in a given interval have to be bounded. Otherwise, a conservative analysis has to assume the worst-case size for all requests, defeating the purpose of the improved analysis. Bounding the sizes of requests in an interval can be done either by characterizing the request streams of the memory clients, or by adding enforcement of a particular traffic profile in the memory arbiter. We will investigate these options as a part of this work.

4 Requirements

This section lists all requirements in aspect CORE and scope NEAR from Deliverable D 1.1 that are relevant for the memory work package. NON-CORE and FAR requirements are not listed here. The requirements are followed by a comment regarding the extent to which it is fulfilled by the research concept in this document.

M-2-010 No Re-Order:

> The processor may have several read or write requests outstanding. The memory controller shall not reorder these read or write requests from the processor.
>
> *The proposed dynamically scheduled back-end preserves FIFO ordering between requests from its memory clients.*

M-2-012 Compare-and-Swap:

> The memory controller (and network interface) shall support a CAS (compare-and-swap) operation.
>
> *An SDRAM controller works efficiently with large blocks of data and small accesses, such as CAS, should be avoided to enable efficient bounds on bandwidth and response times. CAS is hence more suitable for implementation in the controller of a scratchpad memory (that may or more not be close to the SDRAM controller) and is hence not discussed in this document.*

M-4-020 DRAM Configuration:

> DRAM configuration is memory mapped within the memory controller.
>
> *The interesting configuration parameters in the proposed back-end are the memory map configuration (degree of bank interleaving) and timing constraints of the memory device. These are stored in memory mapped registers that will be programmable in the implementation of the memory controller.*

M-0-062 Memory Performance Counters:

> The Memory shall have a cycle counter, which can be read out for performance analysis.
>
> *This will be done in the implementation of the controller.*

M-5-064 Memory Access Instruction Latency:

> The latency of instructions to access main memory shall be latency-bounded.
>
> *The memory controller is time-predictable and is designed to provide efficient bounds on response times of requests.*

M-6-041 Bounded Memory Access Time:

Any access to a processor external resource (i.e. memory, NoC) shall execute in bounded time (depending on resource and access time).

See previous requirement.

5 Conclusions

This deliverable presents the problem of providing guaranteed service to dynamic applications in time-predictable memory controllers. The concepts behind a dynamically scheduled memory controller and corresponding performance analysis are introduced along with a step-wise trajectory towards its realization. The proposed controller builds on an existing memory controller and analysis, reducing the problem to the design and implementation of a time-predictable dynamically scheduled memory controller back-end and adding support for requests with variable size and alignment in both the implementation and its performance analysis. The goal of this work is to provide guarantees at least equal to those of the existing memory controller for requests with fixed size and alignment and outperform it in presence of more dynamic request streams.

References

[1] Benny Akesson and Kees Goossens. Architectures and modeling of predictable memory controllers for improved system integration. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2011.

[2] Benny Akesson, Williston Hayes, and Kees Goossens. Classification and Analysis of Predictable Memory Patterns. In *Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.

[3] Benny Akesson, Williston Hayes Jr., and Kees Goossens. Automatic Generation of Efficient Predictable Memory Patterns. In *Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2011.

[4] S. Bayliss and G.A. Constantinides. Methodology for designing statically scheduled application-specific SDRAM controllers using constrained local search. In *Field-Programmable Technology, 2009. International Conference on*, pages 304 –307, December 2009.

[5] S. Biswas. Refresh-ahead and burst refresh preemption technique for managing dram in computer system, May 1999. US Patent 5,907,857.

[6] Artur Burchard, Ewa Hekstra-Nowacka, and Atul Chauhan. A real-time streaming memory controller. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 20–25, 2005.

[7] J.M. Dodd. Adaptive page management, July 2006. US Patent 7,076,617.

[8] Dave Dunning, Randy Mooney, Pat Stolt, and Bryan Casper. Tera-Scale Memory Challenges and Solutions. *Intel Technology Journal*, 13(4):80–101, December 2009.

[9] S.A. Edwards, S. Kim, E.A. Lee, I. Liu, H.D. Patel, and M. Schoeberl. A disruptive computer design idea: Architectures with repeatable timing. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pages 54–59. IEEE, 2009.

[10] Kees Goossens, Om Prakash Gangwal, Jens Röver, and A. P. Niranjan. Interconnect and memory organization in SOCs for advanced set-top boxes and TV — Evolution, analysis, and trends. In *Interconnect-Centric Design for Advanced SoC and NoC*, chapter 15, pages 399–423. Kluwer, 2004.

[11] Sven Goossens, Benny Akesson, and Kees Goossens. Memory-Map Selection for Firm Real-Time Memory Controllers. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2012.

[12] P. Gumming. The TI OMAP Platform Approach to SoC. *Winning the SoC revolution: experiences in real design*, page 97, 2003.

[13] Sven Heithecker and Rolf Ernst. Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements. In *Proceedings of the Design Automation Conference (DAC)*, pages 575–578, 2005.

[14] International Technology Roadmap for Semiconductors (ITRS), 2009.

[15] P. Kollig, C. Osborne, and T. Henriksson. Heterogeneous Multi-Core Platform for Consumer Multimedia Applications. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1254–1259, 2009.

[16] K.B. Lee, T.C. Lin, and C.W. Jen. An efficient quality-aware memory controller for multimedia platform SoC. 15(5):620–633, 2005.

[17] C. Macian, S. Dharmapurikar, and J. Lockwood. Beyond performance: Secure and fair memory management for multiple systems on a chip. In *IEEE International Conference on Field-Programmable Technology (FPT)*, pages 348–351, 2003.

[18] Orlando Moreira, Frederico Valente, and Marco Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 57–66, 2007.

[19] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Shared Memory Controllers. *IEEE Micro*, 29(1):22–32, 2009.

[20] S.T. Novak, J.C. Peck Jr, and S. Waldron. Method and apparatus for optimizing memory performance with opportunistic refreshing, November 2000. US Patent 6,147,921.

[21] M. Paolieri, E. Quinones, F.J. Cazorla, and M. Valero. An Analyzable Memory Controller for Hard Real-Time CMPs. *Embedded Systems Letters, IEEE*, 1(4):86 –90, December 2009.

[22] Jan Reineke, Isaac Liu, Hiren Patel, Sungjun Kim, and Edward A. Lee. PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation. In *CODES+ISSS '11: Proceedings of the IEEE/ACM international conference on Hardware/software codesign and system synthesis*, October 2011.

[23] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. Memory access scheduling. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 128–138, 2000.

[24] H. Shah, A. Raabe, and A. Knoll. Bounding wcet of applications using sdram with priority based budget scheduling in mpsocs. Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE), 2012.

[25] J. Shao and B.T. Davis. A burst scheduling access reordering mechanism. In *Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, pages 285–294, 2007.

[26] Liesbeth Steffens, Manvi Agarwal, and Pieter van der Wolf. Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach. *ECRTS '08: Proceedings of the Euromicro Conference on Real-Time Systems*, pages 255–265, 2008.

[27] C.H. van Berkel. Multi-core for Mobile Phones. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1260–1265, 2009.

[28] Pieter van der Wolf and Jeroen Geuzebroek. SoC Infrastructures for Predictable System Integration. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1 –6, March 2011.

[29] Wolf-Dietrich Weber. *Efficient Shared DRAM Subsystems for SOCs*. Sonics, Inc, 2001. White paper.