Memory-Map Selection for Firm Real-Time SDRAM Controllers

Sven Goossens, Tim Kouters, Benny Akesson, and Kees Goossens
Eindhoven University of Technology, Eindhoven, The Netherlands

***Abstract*—A modern real-time embedded system must support multiple concurrently running applications. To reduce costs, critical SoC components like SDRAM memories are often shared between applications with a variety of firm real-time requirements. To guarantee that the system works as intended, the memory controller must be configured such that all the real-time requirements of all sharing applications are satisfied. The attainable worst-case bandwidth, latency, and power of the memory depend largely on *memory map* configuration. Sharing SDRAM amongst multiple applications is challenging, since their requirements might call for different memory maps.**

This paper presents an exploration of the memory-map design space. Two contributions improve the memory-map selection procedure. The first contribution reduces the minimum access granularity by *interleaving requests over a configurable number of banks* instead of all banks. This technique is beneficial for worst-case performance in terms of bandwidth, latency and power. As a second contribution, we present a *methodology to derive a memory-map configuration*, i.e. the access granularity and number of interleaved banks, from a specification of the real-time application requirements and an overall memory power budget.

I. Introduction

Embedded applications with real-time requirements are mapped to heterogeneous multiprocessor systems. The computational demands placed upon these systems are continuously increasing, while power and area budgets limit the amount of resources that can be expended [1]–[3]. To reduce costs, applications are often forced to share hardware resources. Functional correctness for *Firm Real-Time* application is only guaranteed if their timing requirements are considered throughout the entire system. When the requirements are not met, it may cause an unacceptable loss of functionality or severe quality degradation [4].

We focus on the real-time properties of the (off-chip) memory. SDRAM is a commonly used memory type because it provides a large amount of storage space at low cost per bit. The response time of an SDRAM is inherently difficult to bound, since it is highly variable because of its internal architecture. It comprises a hierarchical structure of banks and rows that have to be opened and closed explicitly by the memory controller, where only one row in each bank can be open at a time. Requests to the open row are served at a low latency, while request to a different row results in a high latency, since it requires closing the open row and subsequent opening of the requested row. Locality thus strongly influences the performance of the memory subsystem.

The worst-case (minimum) bandwidth and worst-case (maximum) latency are determined by the way requests are mapped to the memory. The worst-case latency can be optimized by accessing the memory at a small granularity (i.e. few words), such that the individual requests take a small amount of time to complete. This allows fine-grained sharing of the memory resource, at the expense of efficiency, since the overhead of opening and closing rows is amortized over only a small number of bits. Latency sensitive requests like cache misses favor this configuration. Conversely, to optimize for bandwidth, the memory has to be used as efficiently as possible, which requires memory maps that use a large access granularity. Such a configuration targets streaming applications (e.g. video) that are latency tolerant but require high bandwidth. Existing memory controllers offer only limited configurability of the memory mapping and are unable to balance this trade-off based on the application requirements.

A memory controller must take the latency and bandwidth requirements of all of its applications into account, while staying within the given power budget. This requires an understanding of the effect that different memory maps have on the attainable worst-case bandwidth, latency and power.

This paper contains the following two contributions: 1) We *explore the full memory map design space* by allowing requests to be interleaved over a variable number of banks. This reduces the minimum access granularity and can thus be beneficial for applications with small requests or tight latency constraints. 2) We propose a *configuration methodology* that is aware of the real-time and power constraints, such that an optimal memory map can be selected.

The rest of this paper is organized as follows. In Section II, related work is discussed. Section III gives background on the SDRAM architecture. Section IV shows the effect of configuring the number of banks over which requests are interleaved. Section V shows how to compute the memory configuration based on the requirements of a real-time workload. This method is applied in a case study in Section VI, followed by conclusions in Section VII.

II. Related work

Several SDRAM controllers focusing on real-time applications have been proposed, all trying to maximize the worst-case performance. [5] uses a static command schedule computed at design time. Full knowledge of the application behavior is thus required, making it unable to deal with dynamism in the request streams.

The controller proposed in [6] dynamically schedules precomputed sequences of SDRAM commands according to a fixed set of scheduling rules. The controller proposed in [7] follows a similar approach. [8] dynamically schedules commands at run-time according to a set of rules from which an upper bound on the latency of a request is determined. [6] and [8] use a memory map that always interleaves requests over all banks in the SDRAM, which sets a high lower bound on the smallest request size that can be supported efficiently.

978-3-9810801-8-6/DATE12/©2012 EDAA

[6] supports multiple bursts to each bank in an access to increase guaranteed bandwidth for large requests. [7] allows only single burst accesses to all banks in a fixed sequential manner, although multiple banks can be clustered to create a single logical resource. None of the mentioned controllers take power into account, despite it being an increasingly important design constraint [2].

Our work considers design-time selection of both the number of bursts per bank and the number of banks that is interleaved over. This allows the access granularity to have any size equal to or larger than one burst. Unlike [6]–[8], the choice of memory map can thus completely be optimized for the mix of application requirements.

III. Background

A. SDRAM

An SDRAM consists of a number of banks, each containing a matrix-like memory, structured in rows and columns. Each bank has a row buffer that can store one row. Before data can be read or written, a row has to be opened by moving it to the row buffer. This is done by issuing an *activate* (ACT) command. Columns in the activated row can then be accessed, each read (RD) or write (WR) command resulting in a burst data transfer. The number of transmitted words per read or write command depends on the *burst length* (BL), which for a DDR3 SDRAM is 8. Closing a row is done by *precharging*. To retain data, all rows in the SDRAM have to be refreshed regularly, which is done by precharging all banks and issuing a refresh (REF) command. If no command is required during a clock cycle, a no-operation (NOP) command is issued.

The *peak bandwidth* of the memory is the product of the clock frequency, the interface width (IW), and the data rate. Throughout this paper, we use a MT41J64M16 DDR3-800 module from Micron as the example memory [9]. It runs at 400 MHz, has a 2 byte interface width and a data rate of 2 words per cycle, resulting in a peak bandwidth of 1600 MB/s. *Memory efficiency* is defined as the fraction of the peak bandwidth that can be guaranteed to the applications. There are several factors that reduce the memory efficiency, connected to command timing constraints. Some of these constraints work on a per-bank basis. For example, t_{RC} specifies the minimum time between two ACT commands to the same bank, while t_{RCD} denotes the minimum time between an ACT and a RD or WR command. Overhead from this type of constraints can be partially hidden by pipelining commands to different banks, for example by reading from bank 0 while waiting for bank 1 to activate. Other constraints have to be considered for the memory as a whole. For example, t_{FAW} specifies a time window in which maximally 4 ACT commands may be issued. Both read and write commands use the same data bus, and the command and data bus are shared between different banks. The data bus needs a few cycles to switch direction between reads and writes, which again reduces worst-case efficiency.

B. Real-time memory controller

The *access granularity* (AG) is the minimum number of bytes that can be fetched from the SDRAM in a single request. It can range from a single burst up to a few kilobytes depending on the controller. If an application issues a request smaller than AG, then the excess bytes are fetched and subsequently discarded. The fraction of the fetched data that is useful to the application is called *data efficiency*. *Gross bandwidth* is defined as the worst-case guaranteeable bandwidth, without taking data efficiency into account. *Net bandwidth* is the product of the gross bandwidth and the data efficiency for a given request size and hence corresponds to the bandwidth useful to the application. This is the bandwidth that we guarantee.

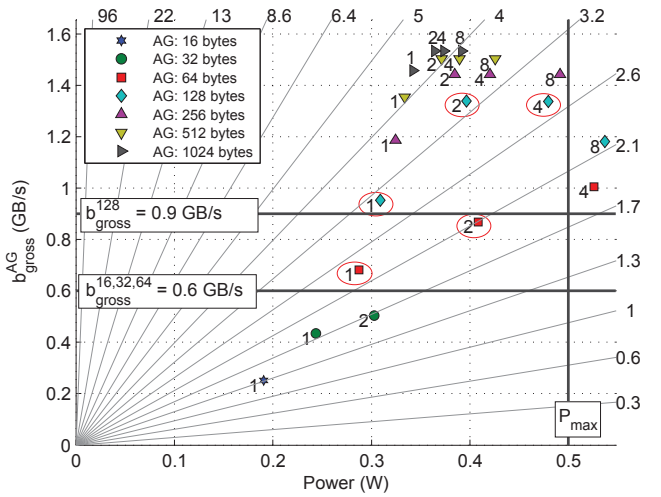

Fig. 1. Maximum guaranteed gross bandwidth at different access granularities versus worst-case power for our example DDR3-800 memory. The labels denote BI, shapes denote the access granularity in bytes. The isolines denote energy efficiency in GB/J (higher is better). The encircled data points are feasible in the case study (Section VI).

This paper uses the concepts of the SDRAM controller proposed in [6]. The controller dynamically schedules precomputed non-preemptive sequences of SDRAM commands, called *memory patterns*, according to a fixed set of scheduling rules. There are five types of patterns: 1) read, 2) write, 3) read/write switching, 4) write/read switching, and 5) refresh. Large requests are divided into smaller requests that are as large as the access granularity by a preprocessing block. The controller guarantees a service latency [10] and rate (bandwidth), based on analysis of the arbiter and the patterns [11]. A small state machine schedules the appropriate pattern given the request and the memory state.

The SDRAM controller uses a close-page policy that precharges a row as soon as possible after a request. The advantage of this policy is that the time penalty caused by the precharge-to-activate constraint can be (partially) hidden by bank parallelism within the access, which improves the worst-case bounds.

IV. Memory-map parameters

This section explores different ways in which a memory request can be mapped to the bank, row, and column structure of an SDRAM, and we evaluate the impact on the worst-case bandwidth, latency, and power. This is illustrated in Fig. 1 and 2 for our example DDR3-800 memory. The exact derivation of the figures is discussed later in Section V.

[11] has shown that the memory efficiency monotonically increases with the access granularity (AG), because the constant overhead of a transfer is amortized over an increasing amount of data. To exploit this effect, they introduced the *Burst Count* (BC) parameter, which controls the number of bursts to each bank per memory access. While for large requests efficiency

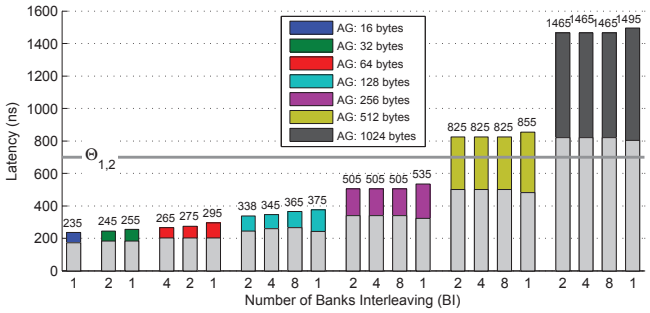

Fig. 2. Worst-case latency with two interfering requests for our example DDR3-800 memory. Bars are grouped and sorted ascending by access granularity. The bottom part of the bars shows the latency as a result of one interfering request and refreshes, which is independent of the arbitration policy. The top part shows the latency caused by one additional request.

improves when increasing the access granularity, a penalty is paid in terms of minimum latency. For small requests, increasing AG may even reduce the net bandwidth. This is because if AG is larger than the request size, time is spent fetching data which is later discarded. Since the workload for a memory controller can consist of both small and large requests, efficient support for small requests is of key importance.

We propose that the number of banks involved in an access can also be used as a parameter to scale the size of a memory access. Instead of interleaving an access over all available banks (e.g. [6], [8]), we propose to use only a subset of the banks. For example, if the SDRAM has 8 banks, *bank clusters* of 8, 4, 2 and 1 bank can be used. We refer to this parameter as the number of *Banks* the memory map *Interleaves* over (BI). With the introduction of this parameter, the access granularity in bytes is given by $\text{AG} = BI \cdot BC \cdot BL \cdot IW$. The minimum access granularity for a DDR3 with 8 banks, a burst length of 8 words and a 2 byte IW can be reduced from 128 to 16 bytes by using this extra degree of freedom.

Interleaving over a subset of the banks has two effects on the net memory efficiency: 1) Small requests can be served without a data efficiency penalty by choosing BI such that the access granularity is equal or smaller than the request size. 2) The negative effect of the t_{FAW} constraint within an access can be avoided by maximally interleaving over 4 banks, while using the burst count to maintain a constant access granularity. Across multiple accesses, the ACT-to-ACT (t_{RC}) constraint already enforces the activates to be spaced sufficiently, since $t_{\text{RC}} \geq t_{\text{FAW}}$ for all DDR2 and DDR3 memories [12], [13].

Based on the effect of BI and BC on the properties of a memory access, we draw the following four conclusions: 1) *Worst-case gross bandwidth* is maximized when the interference of timing constraints within and between accesses is minimized. Interference from read/write switches of the data bus can be minimized by amortizing over large requests. Bank specific timing constraints, such as t_{RCD} and t_{RP}, can be hidden by exploiting bank parallelism and/or increasing the burst count. Increasing either BI or BC thus increases worst-case gross bandwidth. 2) *Worst-case net bandwidth* is the product of worst-case gross bandwidth and data efficiency. It is maximized at the best possible configuration in terms of gross bandwidth that has an access granularity that is smaller than or equal to the request size. At larger granularity, the data efficiency drops below 1, which means part of the fetched data is not used. This is very expensive in terms of performance and energy efficiency. 3) *Worst-case latency* consists of two components: time required for refreshes and read/write switches, and requests from other applications. The worst case contains at least one other request, since requests are non-preemptive. Once the gross memory efficiency is maximized, increasing AG no longer results in more bandwidth, but only increases latency as it takes more time to serve large requests. 4) Activate and precharge commands are relatively expensive in terms of *power* [14]. An activate and precharge command is required for each bank involved in a memory access, so for a given access granularity, decreasing BI (and thus increasing BC) decreases power.

By introducing BI as a degree of freedom, as proposed in this paper, all three aspects of memory performance are thus improved for small requests. Consider for example requests of 64 bytes in Fig. 1 and 2. Optimizing for bandwidth, the (BI4, BC1) combination delivers 70% more net bandwidth and reduces the worst-case latency by 28% compared to the best configuration for 64 byte requests interleaving over all banks (BI8, BC1). The improvement can mostly be attributed to the increase in data efficiency when switching from an access granularity of 128 bytes to 64 bytes, which avoids fetching unnecessary data. If power is more important, choosing (BI1, BC4) reduces worst-case power by 46%. For large requests, we perform at least equally well in terms of bandwidth and latency, while reducing power, e.g. by 7% at an AG of 1024 bytes when comparing (BI8, BC8) with (BI2, BC32).

V. Computing a Memory Controller Configuration

In this section, we show how to configure a memory controller with the new degree of freedom in BI. Given a set of application requirements and a maximum power budget for the memory subsystem, a (BI, BC) tuple must be determined. The proposed methodology is generic and can be applied to any SDRAM type. For the power estimations, any model can be used. We choose the model from [14], which allows us to analyze DDR2, DDR3, LPDDR and LPDDR2 memories.

We specify a real-time application i using three parameters: 1) its request size (RS_i) in bytes, 2) its minimum net bandwidth requirement (b_i) in MB/s, and 3) its maximum service latency requirement (Θ_i) in nanoseconds. For the applications that use the memory, we create a set A_{RT} of these 3-tuples. This allows the derivation of the data efficiency per application for each possible access granularity. We further assume there is a predefined power budget for the entire memory subsystem, P_{max}, given in Watts.

The proposed four-step method is to compute the total required bandwidth and then find the (BI, BC) controller configurations that deliver at least that much bandwidth within the maximum power budget and within the maximum latency requirements of each application. The first step is to convert the individual minimum bandwidth requirements to an *aggregate gross bandwidth requirement* that takes the data efficiency (e_i^{data}) of each application into account, using the access granularity (AG) as a parameter. Equation (1) shows the minimum gross bandwidth that must be supplied by a configuration for it to be a valid candidate for the set of applications:

$$b_{gross}^{AG} = \sum_{b_i \in A_{\text{RT}}} \frac{b_i}{e_i^{data}} = \sum_{b_i \in A_{\text{RT}}} \frac{b_i}{min\left(\left\lceil \frac{RS_i}{AG} \right\rceil, 1\right)} \qquad (1)$$

The second step is to determine the worst-case gross bandwidth that is provided by a (BI, BC) configuration. To that end, the memory pattern generation algorithm from [15] is applied to generate pattern sets for all feasible configurations. The set of feasible values for BI is limited by the number of banks in the memory. The product of BI and BC is part of AG and thus bounded by Equation (1), which shows that as the access granularity increases, so does the aggregate gross bandwidth requirement. Since the guaranteed gross bandwidth can never exceed the peak bandwidth, this bounds the set of valid combinations. By applying the efficiency analysis from [11], the guaranteed gross bandwidth can be derived for each generated pattern set. In Fig. 1, this is shown for our example memory. All configurations that cannot satisfy the aggregate gross bandwidth requirement can be removed from the solution space. This is illustrated in the figure by a horizontal line at the aggregate gross bandwidth for each AG. All data points of an AG that lie below the corresponding line are not feasible.

In the third step, we take the power budget into account. For all feasible (BI, BC) configurations, the energy consumed in each of the memory patterns is derived using the power model in [14]. We define the worst-case power as the power that would be drawn if the most energy consuming combination of patterns possible within the pattern scheduling rules would be repeated indefinitely. All (BI, BC) configurations that consume higher worst-case power than P_{max} can be removed from the solution space. In Fig. 1, the valid configurations then lie left of a vertical P_{max} line.

In the final step, the maximum service latency for the applications is taken into account. For this we use the analysis from [11]. It provides an upper bound on the service latency that is parameterized with the number of interfering requests. When combined with a predictable arbiter like Round Robin or any other arbiter in the class of latency-rate servers [10], the maximum number of interfering requests for each application can be derived, which can in turn be used to derive the worst-case latency in nanoseconds. The choice of arbiter depends on the specific mix of bandwidth and latency requirements and is left to the system designer. A line with the latency requirement for each application can be drawn in a latency plot, where valid configurations lie below these lines, as shown in Fig. 2.

VI. Case study

The procedure of Section V is now applied to a small set of example applications, specified in Table I. We use our example DDR3-800 memory and set the maximum power requirement P_{max} to 0.5 W. We start by determining the aggregate gross bandwidth requirement at different access granularities using Equation (1), and find that only four access granularities are feasible, i.e. yield a bandwidth requirement smaller than the peak bandwidth of the memory: $b^{16}_{\text{gross}} = b^{32}_{\text{gross}} = b^{64}_{\text{gross}} = 600$ MB/s and $b^{128}_{\text{gross}} = 900$ MB/s.

We then determine the worst-case gross bandwidth that is provided by the patterns generated for the (BI, BC) configurations within the solution space, shown in Fig. 1. Since the configurations with an access granularity of 16 and 32 bytes are only capable of providing 251 and 512 MB/s, respectively, we discard them from the solution space. Discarding all the tuples that consume more power than P_{max} leaves five configurations: (1, 4), (2, 2), (1, 8), (2, 4) and (4, 2).

TABLE I
The applications used in the case study

Application	RS_i (bytes)	b_i (MB/s)	Θ_i (ns)
App. 1	128	300	700
App. 2	64	300	700

The final requirement to consider is service latency. We assume that a round-robin arbiter is used to arbitrate between the applications, making a scheduling decision at fixed intervals. The worst case consists of the application just missing its own scheduling slot, followed by the slot reserved for the second application. In Fig. 2, this latency is plotted. All remaining candidate configurations provide a maximum latency within the required 700 ns. This allows us to select the most power efficient memory map, which in this case is (1, 4).

VII. Conclusion

This paper addresses the problem of finding a memory map for firm real-time workloads in the context of SDRAM memory controllers. Existing controllers use either a static memory map or provide only limited configurability. We improve existing work with two contributions. 1) We use the number of banks requests are interleaved over as flexible configuration parameter, while previous work considers it a fixed part of the controller architecture. We use this degree of freedom to optimize the memory configuration to the mix of applications and their requirements. This is beneficial for the worst-case performance in terms of bandwidth, latency and power. 2) We propose a configuration methodology that takes the real-time and power constraints of all applications into account. We have shown how these requirements in terms of bandwidth, latency and a total power budget can be used to find a memory configuration that satisfies the requirements.

VIII. Acknowledgements

This work was supported in part by Agentschap NL, as part of the EUREKA/CATRENE/COBRA project CA104.

References

[1] C. van Berkel, "Multi-core for Mobile Phones," in *Proc. DATE*, 2009.
[2] "International Technology Roadmap for Semiconductors (ITRS)," 2009.
[3] P. Kollig *et al.*, "Heterogeneous Multi-Core Platform for Consumer Multimedia Applications," in *Proc. DATE*, 2009.
[4] L. Steffens *et al.*, "Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach," *Proc. ECRTS*, 2008.
[5] S. Bayliss *et al.*, "Methodology for designing statically scheduled application-specific SDRAM controllers using constrained local search," in *Proc. FPT*, 2009.
[6] B. Akesson *et al.*, "Architectures and modeling of predictable memory controllers for improved system integration," in *Proc. DATE*, 2011.
[7] J. Reineke *et al.*, "PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation," in *Proc. CODES+ISSS*, 2011.
[8] M. Paolieri *et al.*, "An Analyzable Memory Controller for Hard Real-Time CMPs," *Embedded Systems Letters, IEEE*, vol. 1, no. 4, 2009.
[9] Micron Technology Inc., "DDR3-800-1Gb SDRAM Datasheet, 02/10 EN edition," 2006.
[10] D. Stiliadis *et al.*, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, 1998.
[11] B. Akesson *et al.*, "Classification and Analysis of Predictable Memory Patterns," in *Proc. RTCSA*, 2010.
[12] *DDR2 SDRAM Specification*, JESD79-2E ed., JEDEC Solid State Technology Association, 2008.
[13] *DDR3 SDRAM Specification*, JESD79-3D ed., JEDEC Solid State Technology Association, 2009.
[14] K. Chandrasekar *et al.*, "Improved Power Modeling of DDR SDRAMs," in *Proc. DSD*, 2011.
[15] B. Akesson *et al.*, "Automatic Generation of Efficient Predictable Memory Patterns," in *Proc. RTCSA*, 2011.