Adaptive Shared Memory Control for Multimedia Systems-on-Chip

Alexsandro C. Bonatto
IFRS – Federal Institute of Rio Grande do Sul
Porto Alegre, Brazil 91791-508
alexsandro.bonatto@restinga.ifrs.edu.br

Marcelo Negreiros
UFRGS – Federal University of Rio Grande do Sul
Porto Alegre, Brazil 90035-190
marcelo.negreiros@ufrgs.br

Fábio I. Pereira
UFRGS – Federal University of Rio Grande do Sul
Porto Alegre, Brazil 90035-190
fabio.irigon@gmail.com

André B. Soares
UFRGS – Federal University of Rio Grande do Sul
Porto Alegre, Brazil 90035-190
andre.borin@ufrgs.br

Altamiro A. Susin
UFRGS – Federal University of Rio Grande do Sul
Porto Alegre, Brazil 90035-190
altamiro.susin@ufrgs.br

ABSTRACT

Memory subsystem is a major concern in the design of multimedia systems and the memory performance may become the bottleneck for the overall processing performance. This work presents a memory subsystem implementation with adaptive control for access to shared memory in multimedia Systems-on-Chip (SoC). A memory-centric design approach is used to implement the memory subsystem for the purpose of meet bandwidth and deadline requirements of the heterogeneous processing elements. A latency-based model is obtained based on the Worst-Case Response Time (WCRT) criterion. In the presented approach, an adaptive arbiter is designed and implemented to control memory access requests targeting to comply with clients' deadline requirements managing data latency of accesses. Also, an algorithmic implementation of the adaptive arbitration control is presented and evaluated. This paper presents hardware implementation results for the proposed memory subsystem with an adaptive arbiter control. Our proposal of simple estimation of WCRT leads to a hardware implementation with low latency calculation of the system limits imposed to clients. This way the proposed adaptation approach can be implemented at run-time.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Algorithms implemented in hardware*

General Terms

Performance

SBCCI'14 September 01 – 05 2014, Aracaju, Brazil

http://dx.doi.org/10.1145/2660540.2661014

Keywords

Multimedia; System-on-Chip; Hardware; Memory Subsystem; DRAM.

1. INTRODUCTION

Memory management complexity increases with the integration of processing elements in the same system. The optimal memory hierarchy design for a system depends on the specific latency and bandwidth requirements to execute workloads which can vary in time. In the case of SoC platforms for multimedia applications, memory accesses are data-dependent and information is mainly processed in blocks, performing iterations on a given memory region before moving to the next one.

A computing element of the platform is named *client*, implemented as a thread of an application running on a CPU or as a hardware module. Due to the heterogeneity of memory accesses, clients are classified according to latency and bandwidth requirements to match Quality of Service [7, 8, 15]. Because processing elements can have a timing variant data access behavior, adaptive priorities control and client access preemption for memory access has benefits to the memory subsystem, as presented before in [8, 6]. Memory system model and design are widely explored in recent works [1, 2, 7, 8, 6, 12, 3, 4, 15, 16, 17] and show the state-of-the-art in the area of memory systems design.

Adaptive arbitration schemes for DRAM memory access are commonly implemented by issuing bank and address scheduling or command scheduling. Schedule policies are used to give more priority to memory accesses that target an open memory bank or to avoid interleaving of read and write commands in a memory access. Despite the fact that command reordering arbitration algorithms brings benefits to power and bandwidth, these approaches lead to an unpredictable behavior of the memory controller. A predictable behavior is necessary to manage memory accesses issued from real-time applications, where all deadlines have to be met. Thus, predictable memory controllers [1, 15] are proposed to generate well known memory access behaviors that can be easily measured in terms of clock cycles.

The timing predictability of a system is related to differ-

ences between the Worst-Case Execution Time (WCET) [14] and the required timing deadlines. Therefore it is important to provide worst-case timing analysis, i.e., to compute an upper bound of a data transaction between memory and application, called Worst-Case Response Time (WCRT). This measure is used to verify whether the system meets its timing requirements. WCRT was evaluated in previous works as an efficient analysis method to model access delays in data communication through a memory subsystem [12, 4, 16].

In this paper we propose an adaptive access control with priority classification based on the WCRT of a set of clients connected to a memory controller. We propose that clients generate a deadline requirement to complete their transaction, so that the memory controller can classify them when scheduling memory accesses, managing priority of accesses. The memory controller arbiter states an analysis of command schedule by knowing the WCRT for the set of clients. To this, a cycle-based analytical estimation of delays in the memory controller is proposed to be used by the adaptive control. In our proposed adaptive algorithm, the granularity of transactions is the parameter used to improve the sustained bandwidth of data transactions with external memory, regarding tradeoff between arbitration time delays and bandwidth.

A *memory-centric design* method is proposed in this paper as the set of guidelines that uses an evaluation of memory characteristics to design parts of the SoC, aiming to improve the performance in a shared memory subsystem. A digital television set-top box SoC with external DDR3-800 memory is evaluated and results are discussed for the processing of Full-HD (1920 x 1080) at 30 fps video sequences.

This paper is organized as follows. Section 2 presents the memory subsystem background and briefly describes the latency problem in DRAMs. Section 3 presents the delay model used to estimate WCRT of the memory subsystem. Section 4 presents the proposed adaptive control and scheduling of memory transactions. Section 5 presents the results analysis and discussion. Finally, section 6 presents the conclusions.

2. MEMORY SUBSYSTEM BACKGROUND

2.1 DRAM Channel Timing Behavior

The evolution of DRAM was directed to the definition of synchronous interfaces for accessing the memory core, with double data rate capacity. Data are sampled at both edges of the clock (i.e., both the rising edge and the falling edge), doubling the data bus bandwidth. The DRAM core is organized in banks, typically 4 or 8, indexed by row and column addresses. A single address input generates information for bank, row and column selection and decoding.

Memory bus frequencies have risen in the successive generations of SDRAM, from 533 MHz in DDR2 [9] to 1066 MHz in DDR3 [10]. In modern memories like DDR3, the internal DRAM has 8-bit prefetch queue for each bit read in the bus. For example, in a DDR3 SDRAM [10] with bus clock 533 MHz, the internal memory core clock is 66.6 MHz and the bus interface is able to perform 1066×10^6 data transfers per second. Data can be transferred in bursts with 8 data words in each memory access. This feature is known as burst access, which is used to exploit spatial locality in an open bank/row in the memory core.

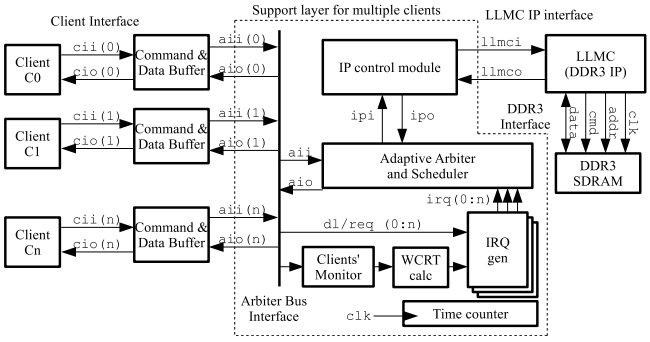

Figure 1: A simplified block diagram with main modules and interconnections to implement a multi-client support layer connected to a single lower-level external memory controller.

The DRAM evolution has been supported by the voltage reduction, power efficiency and data bandwidth increases. However, despite the increase in the memory bus bandwidth, the DRAM core still suffers from several timing limitations. The memory efficiency is limited by row access latencies and internal refresh. Time needed to open a row in a specific memory bank is determined by the row cycle time (tRC). The row cycle time has not evolved at the same rate as the memory bus frequency in the DRAM generations, about 15% comparing DDR3-2133 in relation to DDR2-1066.

2.2 Multi-Client Memory Controller

The memory controller is the lowest level system interface with the main memory. In multimedia systems, external DRAMs are preferred because of their large storage capacity at lower price if compared to internal SRAMs. The memory controller generates commands to the memory port and performs address translation from a linear memory space to an address space formed by bank, row and column. Furthermore, multi-channel memory can be provided from a single memory controller, supporting recent embedded memory generations as Wide-I/O DRAMs [5].

Multi-client support can be added to the memory controller, improving memory accesses by qualifying the clients' requests in a proper way to the DRAM memory. The memory controller is designed with support to more than one client interface, regarding the scalability of the interfaces for the SoC modules. An arbiter is implemented in order to control when and how each client accesses data in the memory subsystem. The arbiter receives command requests from clients, processes the requests and generates commands to the Lower-Level Memory Controller (LLMC). Data is buffered at each memory transfer, in which the data transfer size is regulated by the arbiter. This way it is possible to set an optimal transfer size based on the client's behavior. Fig. 1 shows an architectural implementation of the memory subsystem. The proposed architecture is flexible and can be extended to more clients and channels.

A data transaction issued in a client to the external memory is defined by the action of transferring a sequence of command requests and addresses from the client to the memory. In the case of a write, data is sent with commands and addresses. In the case of a read, a data transaction is ended when read data is delivered to the client. The size of a

Table 1: Memory Subsystem Parameters

Parameter	Description
g	Transaction granularity
l_n	Client transaction length
dl_n	Client deadline
tCK	DRAM Clock cycle time
tRD	Read Data time
tRC	Row Cycle time
tR	Response time
tRL	Read Latency time
tCBR	Command Buffer Retention time
tCCD	Column-to-Column Delay time
tAR	Auto-Refresh time
tWD	Write Data time

transaction can be measured in bytes as well as clock cycles, needed to place commands to the memory controller. The transaction granularity is the minimum time slot that a client can transfer data continuously, without interruption.

3. LATENCY-BASED MODEL

This section presents the implementation of a latency-based analytical model for the memory subsystem. Timing delays are used to model the behavior of memory accesses generated by clients connected to the memory subsystem.

3.1 Analytical Estimation of Delays

To measure delays associated with data transactions in the memory subsystem, a latency-based model is developed from the behavior of the external memory when reading data from or writing data to the DRAM. Timing limitations concerning the internal memory core, because of the DRAM cell behavior, are minimized using transfers in data bursts. The row cycle time is about 10× greater than the data transfer time for a DDR3 SDRAM [10]. In the case of DDR3-800, a data burst of length 8 can be issued in 10 ns, while the internal row-cycle delay is 52.5 ns. The latency-based model is a delay modeling of the transactions in the DRAM, assuming the worst-case to preview the maximum system latency and bandwidth limitations that are imposed to the set of clients. The description of parameters used in the model development is shown in Table 1.

Command requests generated by clients are stored in a temporary memory implemented as buffer, until receive the permission of the arbiter to perform a data transaction. The *Command Buffer Retention* time ($tCBR_n$) is the time interval that a client remains waiting for the arbiter grant. Memory buffers are modeled based on their size, which is related to clock cycles transferring data. Data buffer latency is related to the processing time for queued data when reading or writing a sequence of data bursts. This delay is measured in clock cycles, as presented in (1), and is proportional to the number of active clients accessing the same memory channel.

$$tCBR_n(l_n) = \sum_i (l_i \cdot tCCD + K) \quad (1)$$

In this model, a client n can wait until the other i clients to complete their transactions of size equal to l_i times tCCD that is the column-to-column delay needed to repeat a burst cycle. The penalties due to close and open a memory page is described by the constant $K = tWR+tRP+tRCD$. Following the WCRT analysis, it is estimated that for every group of data transferred there is a memory row change, as the arbiter switches to a new client accessing a different memory region. This effect reduces the spatial locality of DRAM and increase the $tCBR_n$ proportionally to the K constant. For this analysis we obtain the worst case, by generating a row change in every transaction request issued by another client to the memory channel. This model considers that the client n waits for all other clients to complete their transactions configuring the WCRT for this client.

Write and read data delay times (tWD and tRD respectively) are delays associated to data transfer commands in the external memory. These delays are measured in clock cycles and are proportional to the transaction length requested by a client, as presented in (2) and (3).

$$tWD_n(l_n) = l_n \cdot tCCD + K \quad (2)$$

$$tRD_n(l_n) = l_n \cdot tCCD + tRL + K \quad (3)$$

This time interval depends on the number of issued reads or writes in the data transaction. Also, it depends on the read latency (tRL). In the case of a data read, the tRL parameter for reading data, which represents the worst case in time of a transaction from memory.

3.2 Worst-Case Response Time

The memory subsystem model is used to estimate the worst-case response time of a data transaction. Combining equations (1), (2) and (3) leads to a measure in clock cycles of the time spent to transfer a set of data in the memory subsystem. Different combinations of elements can be formed to implement a memory subsystem, but in the present case we use an architectural implementation as proposed in Fig. 1. The memory subsystem response time for the worst-case scenario is calculated by (4).

$$wcrt_n = tR_n + tAR = tCBR_n + tWD_n + tAR \quad (4)$$

The response time for a client n (tR_n) is calculated by the sum of delays inserted by each memory subsystem element in the command path and data path between client and external memory. In the memory subsystem model, command and data transfer delays in a transaction request generated by a client is calculated by the combination of buffer delay and memory controller delay models. The worst-case of delays occurs when all clients are accessing the memory subsystem and also an auto-refresh operation is issued in the external memory. Time needed to perform the auto-refresh operation is denoted by $tRCF$ but is increased by K due to page activation penalties, resulting in $tAR = tRFC + K$.

The choice of transfer granularity (g) influences on the overall system performance as presented in [2]. The preemption of an ongoing transaction regarding a minimum granularity size leads to the minimization of data latency across the memory subsystem to complete requests that are processed by the memory controller and increase data bandwidth for the client that is preempted. As proposed in [2], an optimal value of granularity can be selected by the arbiter to preempt clients. In this implementation, we will make $g_n = l_n/2$, that leads to a simplification of the architectural implementation.

4. ADAPTIVE CONTROL

In our approach, transactions are scheduled by the deadline requirements that are compared with the worst case response time to complete a data transaction. The arbiter works to guarantee that the deadline requirement can be met by generating an interruption request (IRQ) to the scheduler based on the information of an absolute time counter and the deadline requirement. This way it is possible to meet deadlines limited by the rate $dl_n/wcrt_n > 1$. If the deadline requirement is less than the WCRT the arbiter works in a "best-effort" mode and is able to guarantee that the transaction will be finished before the worst-case previewed by the model.

4.1 Adaptation Approach

The proposed method can be used to find the optimal values that can reduce the response time in the memory subsystem. In order to calculate the memory subsystem behavior for each client, it is possible to implement the proposed procedure in an arbiter. The arbiter can control the transaction granularity for each client, with respect to its deadline requirements. Each client generates information about its deadline requirement to a transaction, in clock cycles. The following steps are proposed to implement adaptation:

- WCRT is computed from data transaction sizes and number of clients information;
- The future time instant to generate an IRQ is calculated from the deadline requirement, the WCRT and the absolute time counter value, for each client;
- Generated IRQs for all clients are ordered in a queue that is used by the scheduler to preempt transactions regarding to minimum granularity sizes.

The proposed arbitration algorithm performs a continuous evaluation of the accessing behavior for all clients attached to the memory subsystem. The evaluation is based on the data transaction size in each memory access ($l(n)$) and the client deadline requirement ($dl(n)$). Algorithm 1 is used to calculate the WCRT function based on equation (4) and to generate IRQs for clients. DRAM parameters are stored in a structure named *dparm*. The number of active clients is detected in line 3, by a function implemented by the clients monitor that detect if all clients connected to the memory subsystem are generating valid command requests.

In the proposed algorithm, the WCRT is calculated in the loop presented between lines 5 and 8, in a simple manner multiplying the transaction size by the $tCCD$ parameter and adding a constant. At the end (line 9), the auto-refresh time (tAR) is added. This calculus is of simple hardware implementation, using multiply-and-accumulate structures. The IRQs generation is done between lines 10 and 16, subtracting the previously calculated WCRT from the client deadline requirement, added to the absolute time counter.

The current time instant are compared against the calculated WCRT and deadline requirement (Fig. 1) that generates a future time instant, stored in the *irq* variable, used by the scheduler to generate preemptions. In the case that a deadline requirement is less than the calculated WCRT, so the client generates an IRQ immediately and the *irq* variable is updated with the current time instant. IRQs are generated conditionally to a command request generated by a client.

Algorithm 1 WCRT calculus and IRQs generation.

Ensure: $wcrt = 0, timeCounter = 0$
1: $timeCounter \leftarrow timeCounter + 1$
2: $n \leftarrow numberOfClients(clients)$
3: $[l(n), dl(n)] \leftarrow detectClientsParms(clients)$
4: $K \leftarrow dparm.tWR + dparm.tRCD + dparm.tRP$
5: **for** $i = 0 \rightarrow n - 1$ **do**
6: $\quad tR(i) \leftarrow l(i) \cdot dpartm.tCCD + K$
7: $\quad wcrt \leftarrow wcrt + tR(i)$
8: **end for**
9: $wcrt \leftarrow wcrt + tAR$
10: **for** $i = 0 \rightarrow n - 1$ **do**
11: $\quad$ **if** $wcrt > dl(i)$ **then**
12: $\quad\quad irq(i) \leftarrow timeCounter$
13: $\quad$ **else**
14: $\quad\quad irq(i) \leftarrow timeCounter + (dl(n) - wcrt)$
15: $\quad$ **end if**
16: **end for**
17: **return** irq;

Table 2: Experimental Setup Configuration

Clients	*BW (MB/s)*	*ln*	*dln*
CPU	220	1	290
H264	93.3	1 - 6	680 - 4100
MC	769.8	1 - 18	500
Video	93.3	1 - 6	680 - 4100
Grap	186.6	1 - 6	340 - 2050
TS	2.5	1	25000
DDR3-800 parameters			
Parameter	**ns**	**Parameter**	**ns**
tCK	2.5	tCCD	10
tRC	52.5	tRL	12.5
tWR	15	tRP	15
tRCD	12.5	tRFC	110

5. RESULTS ANALYSIS

In this section, we apply the memory model analysis to a digital television set-top box.

5.1 Experimental Setup

The simulated STB is composed from several processing units and this case study combines six memory clients accessing one DRAM channel, as described before in [13]. A Leon-3 CPU (SPARC-v8 ISA) [11] is connected to the memory subsystem through one level of data/instruction cache (L1), with block size of 32 bytes configured as 1, 2 or 4-way. In this evaluated scenario, the CPU generates accesses to the memory to build graphics with a throughput of 220 MB/s.

In this implementation only single burst access is provided to transfer data between the CPU and main memory. Table 2 presents the parameters used in this analysis, obtained from the hardware simulation behavior presented in [4, 13]. The testbench used is composed by procedures described in VHDL to generate reads and writes simulating the memory access behavior for the digital television STB.

5.2 WCRT Evaluation

An analysis results presented in Fig. 2 shows the evaluated results of WCRT observed in simulation normalized in

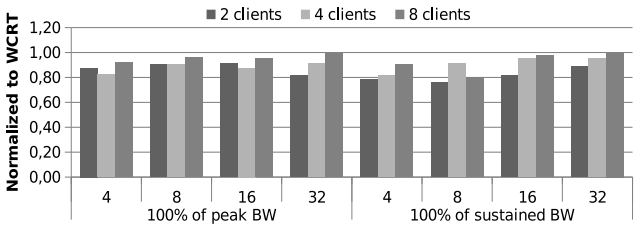

Figure 2: Observed worst-case compared to the WCRT obtained with the model. Maximum normalized values should be less or equal to 1. Horizontal axis represents the transaction sizes (l_n) that was varied in the simulation in a range of 4 to 32 sequential bursts.

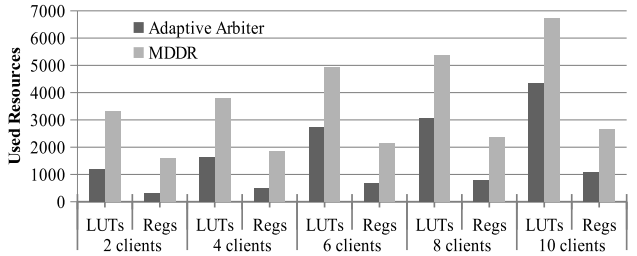

Figure 3: Used resources for synthesis results in a Virtex-6 XC6VLX240T FPGA platform.

relation to the calculated values using the model proposed in (4). The proposed model was simulated for 2, 4 and 8 clients resulting in confidence values that are less than the values calculated by the model. Two cases were simulated: with clients generating accesses to demand 100% of the memory peak bandwidth and generating accesses to demand 100% of the sustained bandwidth. Different transaction lengths were evaluated, from 4 to 32 consecutive repetitions of bursts. The WCRT calculus can be used in the adaptive arbiter to generate an IRQ time instant with enough time to complete a transaction.

5.3 Hardware Implementation

The proposed arbitration algorithm was implemented and validated using VHDL language. Initial implementation results showed the ability of this architecture to manage clients' access requests dynamically. The adaptation process initiates each time a client changes its deadline requirement or the requested transaction size. The hardware cost required supporting the adaptive arbiter depends on the number of clients connected to the memory subsystem. The adaptive arbiter is implemented in a Virtex-6 XC6VLX240T FPGA and logic usage is presented in Fig. 3. The implemented memory controller is able to run at 200 MHz clock frequency, needed to interface with a DDR3-800 memory physical interface controller.

5.4 Case Study – Digital Television STB

Table 3 shows the configuration of corner cases used to simulate the memory accessing behavior of hardware modules that compose the digital television STB. Deadline requirements are calculated proportionally to the desired bandwidth (presented in Table 2) and to the data transaction

Table 3: Configuration for corner cases simulation of the Set-Top Box for Full-HD 30fps video processing.

Clients	**Case 1**		**Case 2**	
	l(n)	***dl(n)***	***l(n)***	***dl(n)***
CPU	1	290	1	290
MC	1	500	1	500
H264	1	680	6	4100
Video	1	680	6	4100
Grap	1	340	6	2050
TS	1	25000	1	25000
WCRT	695 ns		775 ns	
Clients	**Case 3**		**Case 4**	
	l(n)	***dl(n)***	***l(n)***	***dl(n)***
CPU	1	290	1	290
MC	18	500	18	500
H264	1	680	6	4100
Video	1	680	6	4100
Grap	1	340	6	2050
TS	1	25000	1	25000
WCRT	755 ns		925 ns	

size requested by each memory client. The average period of CPU requests and its deadline requirement is 290 ns, to maintain the required bandwidth. In the case of MC, deadline requirement is stricter than CPU and has to be complied in 80 ns for transactions in single burst mode (cases 1 and 2), due to the elevated bandwidth. This requirement is impracticable by the memory subsystem due to time to preempt another client and the tAR parameter. The accessing period of MC client in cases 2 and 3 are approximately 1400 ns, but the deadline requirement is fixed to 500 ns due to processing time needed by this module.

The proposed Adaptive Arbiter (AA) algorithm implementation was compared with Fixed Priority with Preemption (FPP), First-Come First-Served (FCFS) and Round-Robin (RR) arbiters with infinite time slot (RR) and single access (RR1) quantum sizes. In FCFS arbitration scheme, incoming data transaction requests are attended to in the order that they arrived, without priority. In RR1 arbitration scheme, time slices are assigned to each client in equal sizes and in circular order, handling all clients without priority.

Simulated WCRT and bandwidth results for the proposed adaptive arbiter compared with FCFS, RR and FPP are shown in Figures 4 and 5. The proposed adaptive arbitration approach is able to adjust priority of client transactions by calculating the WCRT for the set of clients and generating IRQs as needed to complete a data transaction. For all cases presented in Table 3, the WCRT is different because transaction lengths change in every case. The measured average response times for MC and CPU are respectively: 90 and 85 ns (case 1), 55 and 55 ns (case 2), 245 and 60 ns (case 3) and 195 and 55 ns (case 4).

In cases 1 and 2, all deadlines can be met with exception for the CPU in FCFS, RR and RR1 arbitration schemes. FPP and RR1 presents better results for latency in the CPU client, because the fastest clients switching in RR1 and because fixed prioritization with preemption in the FPP scheme. WCRT normalized for RR1 is higher than 1.2. In cases 1 and 2, the minimum bandwidth required for the MC client cannot be supplied by the memory controller, but the

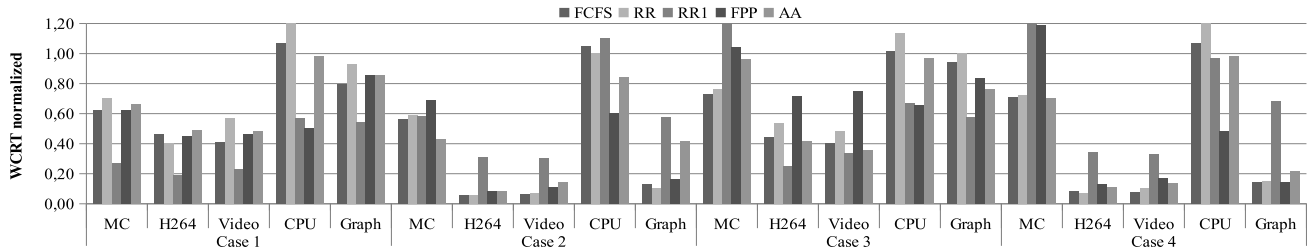

Figure 4: Observed worst-case response time for clients in four simulated cases in the STB. Results presented are normalized to the calculated values.

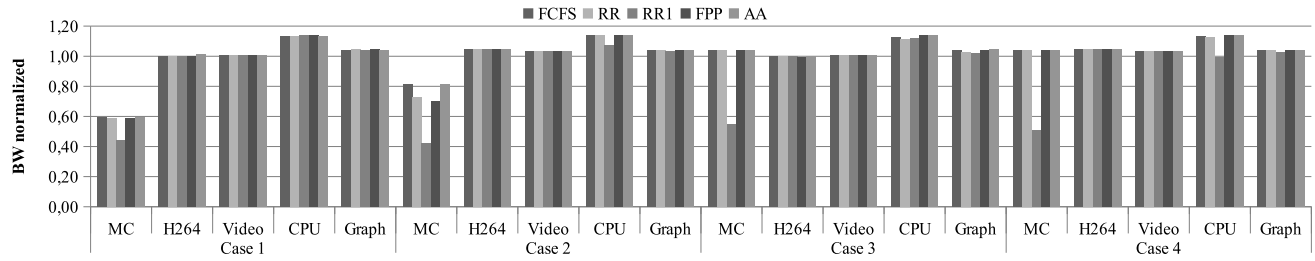

Figure 5: Observed bandwidth for clients in the four availed cases in the STB. Results presented are normalized to the calculated values.

AA scheme achieves 80% of the required bandwidth, as the FCFS scheme. Bandwidth cannot be guaranteed because it exceeds the maximum sustained bandwidth by the memory when accessing data at single bursts, limited in 634 MB/s.

Long data transactions were evaluated in cases 3 and 4, with the MC client performing sequences of 18 bursts. In cases 3 and 4, MC deadline requirement of 500 ns cannot be met using FPP or RR1 arbitration schemes. Also, the CPU deadline requirement of 290 ns cannot be met using RR scheme. The AA scheme can sustain the 500 ns deadline requirement for MC and 290 ns for CPU. Bandwidth demanded for clients can be supplied in all arbitration schemes, with an exception for RR1 arbitration scheme, which inserts higher level of time delays due to page changes and single bursts per transaction. In case 4, FCFS and RR cannot guarantee the 290 ns deadline requirement of CPU client. The AA scheme can guarantee deadlines for MC and CPU in both cases 3 and 4. Deadline requirements in for CPU and MC in all cases and for Grap, H264 and Video in cases 1 and 3 are less than the WCRT for the ensemble of clients.

5.5 Related Work

We present in this paper an analytical model for WCRT estimation with simple hardware implementation based on delays estimation for memory transactions. As studied before by Shah et al. [12], an analytical analysis method achieves better results in latency estimation if compared to a method based on abstraction of data communication, as the latency-rate method [3]. Even based on the worst case, the analytical analysis of delays is preferred than the use of temporal estimation [12]. If compared to [12] and [4], our proposed model achieves better results in a overloaded memory subsystem. As we present, the normalized WCRT from the observed values are between 0.8 and 1.0 from the calculated ones, resulting in a better approximation than the presented in other works.

Our proposal of simple estimation of WCRT leads to a hardware implementation with low latency calculation of the system limits imposed to clients. This way the proposed adaptation approach can be implemented at run-time, different from the arbiters proposed in [6, 1, 4], that use static priority arbiters.

6. CONCLUSIONS

Predictable memory controllers' implementation has many benefits for system performance because they lead to the correct estimation of the influence of concurrence in a shared memory channel. The accuracy in estimation of the worst-case response time for a data transaction requested by a client is fundamental to guarantee time deadline requirements and client performance. In this paper we present an approach to estimate worst-case response times for a set of clients in a memory subsystem with simple hardware implementation and enough accuracy to improve system performance.

An adaptive arbitration scheme was proposed here, based on the estimation of worst-cases to complete data transactions and time deadlines imposed by clients. For each client request it is calculated the future time instant to initiate the transaction and latency of other clients are fitted into their deadline limits. With the proposed approach, time deadlines can be met in a variety of different accessing behaviors that are present in systems composed of heterogeneous modules. The proposed adaptive arbitration algorithm presents fast performance results as FCFS arbitration schemes with a behavior of the fixed priority with preemption arbitration but preventing from starvation of clients.

7. ACKNOWLEDGMENTS

This work is partially supported by CAPES and CNPq, Brazilian founding agencies for human resources and scientific research.

8. REFERENCES

[1] B. Akesson and K. Goossens. Architectures and modeling of predictable memory controllers for improved system integration. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[2] A. C. Bonatto and A. A. Susin. Run-time SoC memory subsystem mapping of heterogeneous clients. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 578–581, June 2014.

[3] M. Gomony, C. Weis, B. Akesson, N. Wehn, and K. Goossens. DRAM selection and configuration for real-time mobile systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 51 –56, march 2012.

[4] M. D. Gomony, B. Akesson, and K. Goossens. Architecture and optimal configuration of a real-time multi-channel memory controller. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1307–1312, San Jose, CA, USA, 2013. EDA Consortium.

[5] JEDEC. *JEDEC-JESD229: Wide I/O Single Data Rate*. JEDEC Solid State Technology Association, Virginia, USA, 2011.

[6] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver. A QoS-aware memory controller for dynamically balancing GPU and CPU bandwidth use in an MPSoC. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 850–855, June 2012.

[7] K.-B. Lee and T.-S. Chang. *Essential Issues in SoC Design*, chapter SoC Memory System Design, pages 73–118. Springer Netherlands, 2006.

[8] K.-B. Lee, T.-C. Lin, and C.-W. Jen. An efficient quality-aware memory controller for multimedia platform SoC. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(5):620–633, May 2005.

[9] Micron. *DDR2 SDRAM: 2Gb: x4, x8, x16 DDR2 SDRAM Features*. Micron Technology, Inc, 2007.

[10] Micron. *DDR3 SDRAM: 4Gb: x4, x8, x16 DDR3 SDRAM Features*. Micron Technology, Inc, 2009.

[11] G. Research. Grlib IP core user's manual, jan 2013.

[12] H. Shah, A. Knoll, and B. Akesson. Bounding SDRAM interference: Detailed analysis vs. latency-rate analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 308–313, March 2013.

[13] A. B. Soares, A. C. Bonatto, and A. A. Susin. Development of a SoC for digital television set-top box: Architecture and system integration issues. *International Journal of Reconfigurable Computing*, 2013(1), Jan. 2013.

[14] L. Thiele and R. Wilhelm. Design for timing predictability. *Real-Time Syst.*, 28(2-3):157–177, Nov. 2004.

[15] P. van der Wolf and J. Geuzebroek. SoC infrastructures for predictable system integration. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[16] G. Zhang, Y. Jiang, W. Wang, and M. Su. A laxity-aware memory access scheduler for high performance multimedia SoC. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 603–608, Aug 2011.

[17] G. Zhang, H. Wang, X. Chen, and P. Li. Fair memory access scheduling for quality of service guarantees via service curves. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 174–181, July 2012.