Run-Time SoC Memory Subsystem Mapping of Heterogeneous Clients

Alexsandro C. Bonatto[1,2] and Altamiro A. Susin[2]
[1]Federal Institute of Rio Grande do Sul (IFRS), Porto Alegre, Brazil; email: alexsandro.bonatto@restinga.ifrs.edu.br
[2]Microelectronis Program (PGMicro) – UFRGS, Porto Alegre, Brazil; email: altamiro.susin@ufrgs.br

***Abstract*—Systems-on-chip for multimedia applications present strict requirements for the memory subsystem regarding bandwidth and latency. Typically the CPU and several IPs are connected to a multi-client memory subsystem to access data in a shared memory channel. An arbiter manages accesses conflicts due to concurrent memory requests. In this paper it is proposed an intelligent arbitration algorithm able to classify clients at run-time, according to their access requirements. The arbiter state variables are adjusted at run-time, increasing the memory channel performance if compared to a non automated client scheduling. Simulated results showed an average latency improvement of 29.4% for a multimedia SoC.**

Keywords—Multimedia, System-on-Chip, Adaptive System, DDR3 SDRAM, Memory Controller.

I. INTRODUCTION

Memory subsystem efficiency is a critical factor for embedded system performance, since it is the bottleneck in current multimedia processing systems. Architectural decisions made in early system design stages have great influence on memory bandwidth, which may be reduced because of memory access latency. Design space variables are defined and evaluated before implementation of the system architecture. These design considerations refer to multi-variable equations that are used to model the memory system characteristics.

In the case of System-on-Chip (SoC) architectures for multimedia applications, information is mainly processed in blocks, performing iterations on a given data region before moving to the next one. Also, multimedia SoCs are composed by several heterogeneous processing modules, and most data transfers pass through the main memory. Because multimedia processing is data-dependant, the memory access rate and behavior can vary along the execution time. This may be a problem for strategies that rely on worst case scenarios or that target a fixed state for the system.

In this work it is proposed an algorithm that dynamically classifies clients (processing modules) of a memory subsystem at run-time. A latency-based model for memory subsystem in a multimedia SoC is used while memory subsystem parameters are monitored at execution time. This approach can cope with the time variability of the memory accesses. It is proposed a memory-centric design approach that targets the integration of heterogeneous modules by the memory subsystem. In the proposed approach, the memory subsystem evaluates the target memory timing and system modules access characteristics to give an optimal state for the system.

Previous works address the classification of clients according to latency, bandwidth and deadline requirements in a shared memory subsystem [1-7]. Also, analytical models for memory subsystem were studied by [1-2], [5], [8]. However these approaches may result in over-dimensioned systems, because they are too pessimistic, and are not capable to optimally configure memory subsystems at run-time. An analytical analysis of the memory subsystem behavior, even a worst case scenario analysis, is preferred to the use of temporal estimation, as is presented in [8].

In this work, two main contributions are presented: a) a latency-based analytical model used for estimate the worst-case execution time in clients' accesses to the memory subsystem and b) an adaptive arbitration algorithm for run-time classification of heterogeneous clients. The proposed algorithm calculates design space variables based on an analytical model [7] and classifies clients at system run-time. This classification is used in the memory controller arbiter to classify memory accesses in different levels of priority. The goal is to implement an optimized algorithm to perform the memory subsystem adjustment based on adapting the transaction granularity which maximizes data burst capacity.

The rest of the paper is organized as follows. Section II presents the memory subsystem model and the adaptive algorithm description. Section III presents the experimental results comparisons and hardware implementation results. Finally, section IV presents the conclusions.

II. SOC MEMORY SUBSYSTEM MODEL

The development of the proposed memory subsystem model is based on the system DRAM parameters and memory subsystem elements characteristics. Table I contains a description of the parameters that are used along this paper.

TABLE I. MEMORY SUBSYSTEM MODEL PARAMETERS.

Parameter	Description	Unit
$L_C(n)$	Transaction Length of a client *n*	*cycles*
L_T	Transaction Granularity	*cycles*
$D_C(n)$	Deadline for a client *n*	*ns*
tCK	Clock period	*ns*
tRC	Row-cycle time	*ns*
tRL	Read Latency	*ns*
tCCD	Column to Column Delay	*ns*
$tR(n)$	Response time for a client *n*	*ns*

This work was partially supported by CAPES and CNPq.

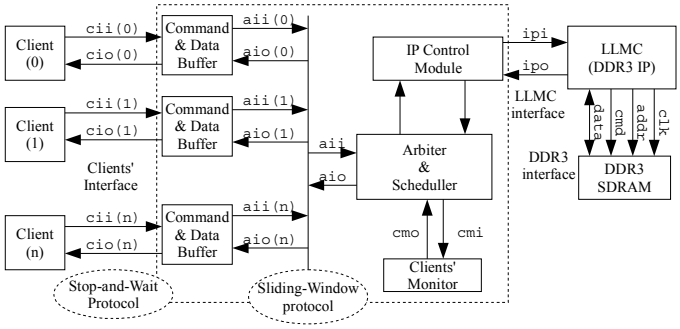

Fig. 1. Memory controller architecture with multi-client support.

A. The DRAM Core Performance Characteristics

The DRAM core is organized into independent banks of memory cells, indexed each one by row and column addresses. For example, a DDR3 SDRAM [9] contains eight memory banks accessed through an 8-bit pre-fetch queue for each bit read in the data bus. If the bus clock frequency is 1066 MHz, the internal memory core clock is 133,5 MHz and the bus interface is able to perform 2133 Mega transfers per second (MT/s). Due to this architecture, data can be transferred in bursts of length equal to 8 bits per data memory pin. This is the minimum transfer granularity.

In the last years, the evolution of DRAM was directed to increase the synchronous interfaces speed with double data rate capability. However the DRAM core latency was not improved at the same rate as the bandwidth improvement. Memory bus bandwidth has doubled from the fastest DDR2 to the fastest DDR3 SDRAM. Nevertheless, for the same memories, latency improvement is only 15% in *tRC* (row-cycle time) and 17% in *tRAS* (row address strobe time). Due these factors, DRAM core access latencies are the performance bottleneck in most of computing systems [10-11].

B. Memory Subsystem Architecure for Multimedia SoCs

In embedded multimedia systems it is preferred to use DRAM memories as main data storage. DRAMs have large storage capacity at lower price if compared to internal SRAMs. A memory controller is used to interface with the DRAM. Furthermore, multi-client and multi-channel support can be provided from a single memory controller. Multi-client support is used to improve memory access by classifying the clients' requests in a proper way to schedule the DRAM memory.

In this work, the memory controller is designed with multi-client support, regarding the scalability of the interfaces to the SoC modules. Also, client interfaces are classified according to their access characteristics, named *Quality of Service* (QoS) requirements. Fig. 1 shows the description of the architectural implementation of the elements in the proposed memory subsystem. Interfaces with clients are composed by command and data buffers used to hold temporary requests before be granted by the scheduler. Atomic data transfers are supported with the use of buffers implemented to hold command, address and data before to complete a memory transaction. In the proposed approach, clients should specify the transfer duration and a deadline requirement of each data transaction to memory.

An arbiter and scheduler module (Fig.1) process commands concurrently generated by the clients' buffers and interfaces with a memory IP controller module. The arbiter classifies clients' priority dynamically, according to their deadline requirements and requested transfer duration ($L_C(n)$). The arbiter sets the optimal transaction granularity (L_T) based on the clients' accesses behavior. A clients' monitor unit (Fig.1) generates statistics from accesses generated by clients, as mean time interval between requests and transfer request durations. The scheduler controls accesses to the memory channel from priorities classification and the transaction length granularity information, generated by the arbiter. The transaction length granularity is the minimum access time slot for a client.

C. Memory Subsystem Latency-Based Model

A memory subsystem latency-based model is used to estimate the worst-case execution time for each client accessing the shared memory channel. This model combines arbitration delay, memory buffers delay and external memory delay. A previous analysis of the memory subsystem model was presented before in [7]. The proposed model is implemented as an algorithm to calculate memory subsystem parameters that maximize data transfer bandwidth. The arbiter is modeled based on transfer granularity delays. Memory buffers are modeled based on their size, which is related to clock cycles. The objective function to be minimized is the delay through the elements of the memory subsystem, mainly dominated by the memory latency.

The analytical estimation of optimal points is obtained from the derivation of previously presented equations [7]. Equation (1) shows the worst-case response time for a client *n*. It contains a global minimum that can be calculated in order to found the optimal value for L_T that reduces *tR(n)*. As discussed before, data and command buffer delays can be reduced by reducing the client's transaction duration size ($L_C(n)$) or by augmenting the transaction granularity (L_T) [7]. The global minimum is found taking the derivative of (1) with respect to L_T and making *tR(n)'=0*, resulting (2).

$$tR(n) = \sum_{i=0}^{n-1} L_C(i) \cdot tCK + L_C(n) \cdot \left(tCK + \frac{tRC}{L_T} \right) + (tCK + tCCD) \cdot L_T + tRL \quad (1)$$

$$L_T = \sqrt{\frac{L_C(n) \cdot tRC}{tCK + tCCD}} \quad (2)$$

D. Client Quality of Service Requirements

In a shared memory subsystem Quality of Service (QoS) requirements are established to characterize different access characteristics for clients sharing the same memory channel. Usually, clients are classified according to bandwidth and latency requirements as proposed in [3] and [4], resulting in a fixed configuration for the system. In our proposed design approach, both deadline requirement information and data transaction duration must be set by the client. The deadline requirement $D_C(n)$ represents the maximum time interval that a client can wait until complete a data transaction $L_C(n)$. The arbiter calculates the response time for a data transaction to

guarantee for all clients the ratio $D_C(n) / tR(n) > 1$, otherwise the deadline requirement cannot be guaranteed and the arbiter would adjust again the L_T value.

E. Adaptive Arbitration Algorithm

The proposed arbitration algorithm performs a continuous evaluation of the accessing behavior for all clients connected to the memory subsystem. The evaluation is based on the data transfer duration in each memory access ($L_C(n)$) and the client deadline ($D_C(n)$) requirement. The arbitration algorithm is segmented into five steps described below:

a) Access priorities are sorted based on deadline requiments $D_C(n)$ using the function sort_clients_deadline() resulting an ascending order of index n stored in the variable clients_dl;

b) The optimal value of transaction granularity (L_T) that reduces the response time (tR(n)) for each client is calculated using the function calc_transaction_lenght() and values are stored in the variable LTc;

c) The arbiter choses the value of L_T that benefits the client with the most strict deadline requirement (n=0);

d) For each client it is calculated the worst-case response time (tR(n)) using the function calc_response_time(), following the sorted index k, to verify that this configuration meets the deadlines requirements for all clients;

e) If a client deadline is not satisfied, the arbiter set L_T with the value of this client and restart verifing the response time values for all clients.

Function *arbitration_algorithm* described in the Alg.1 is implemented by the arbiter to determine the transfer granularity based on the worst-case response time. This function is called every time a client changes its access behavior. Therefore, it is expected that the adaptation algorithm runs in a long term evaluation of clients. Two sets of information entries this function: a) transaction duration and deadlines requirements (*clients*) and b) target DRAM parameters (*dparrm*). Function *calc_response_time* implements equation (1) and function *calc_transaction_length* implements (2).

Memory subsystem predictability is maintained by using the same minimum transfer granularity for all clients. All clients can be interrupted after L_T cycles of continuous accessing to the memory channel. By this way, any client can access memory contents in a predictable time interval.

III. Results Analysis

A multi-client memory subsystem model and arbitration algorithm was evaluated using numerical simulation, in a scenario that simulates the arbiter run-time adaptation. A synthetic platform with 25 clients was modeled using Matlab to evaluate the adaptation of the proposed algorithm.

A. Synthetic Application Analisys and Comparison

Latency and bandwidth improvement results are presented in Table II, if compared to a transaction size of one single access ($L_T=1$). Table III shows the results analysis for an adaptation in the memory subsystem with 25 clients configured with $L_C(n)=4$ and $D_C(n)=1000$. The first adaptation occurs when client $n=15$ reduces its deadline requirement. Second and third adaptations occurs when client $n=20$ increases its transaction length and after decreases its deadline requirement. In the third adaptation, L_T parameter is increased to support the strict deadline requirement. In this last adaptation, client $n=15$ deadline requirement is satisfied because $tR(15) = 238,7$ *ns.*

TABLE II. Memory Subsystem Adaptation Results For 25 Clients with Starting from $L_T=1$ for $L_C(N)=1, 4, 8$ and 16.

Transaction Duration	L_T (cycles)	tR max. (ns)	ALI[a]	ABI[a]
$L_C(n)= 4$	4	309.4	39 %	65 %
$L_C(n)= 8$	6	532.5	47 %	95 %
$L_C(n)= 16$	9	958.1	54 %	128 %

a. Average Latency Improvement (ALI) and Average Bandwidth Improvement (ABI).

TABLE III. Memory Subsystem Adaptation Example For a Synthetic Application with 25 Clients ($L_C(N)$ = 4 and $DC(N)$ = 1000).

Adaptation Stages	Altered Parameters	L_T (cycles)	tR(n) (ns)	LI(n)[a]	BI(n)[a]
$D_C(n) \downarrow$	$D_C(15) \rightarrow 500$	4	129.3	45 %	81 %
$L_C(n) \uparrow$	$L_C(20) \rightarrow 32$	4	678.8	− 150 %	220 %
$D_C(n) \downarrow$	$D_C(20) \rightarrow 400$	13	340.2	50 %	99 %

a. Latency Improvement (LI) and Bandwidth Improvement (BI) for the modified client.

TABLE IV. Set-Top Box Results Evaluation Using the Proposed Algorithm for Heterogeneous Clients (L_T = 9).

Clients	BW (MB/s)	$L_C(n)$ (cycles)	$D_C(n)$ (ns)	n	tR(n) (ns)	LI(n)[a] (%)
CPU_Sw	150	1	426.7	1	159.4	- 51.8
MC_in	769.8	18	406.5	0	253.1	74.7
H264_out	93.3	6	4115.8	4	221.3	45.1
Video_in	93.3	6	4115.8	5	232.5	43.9
CPU_Gr	186.6	6	2057.9	2	198.8	53.3
Grap_in	186.6	6	2057.9	3	210.0	51.9
TS_in	2.5	1	25600.0	6	206.3	- 11.1

a. Latency Improvement (LI) for the modified client.

Algorithm 1: Adaptive arbitration algorithm

```
01: function arbitration_algorithm(clients, dparm) {
02:   variable: n, clients_dl, tR, LT, LTc, j← 1;
03:   clients_dl ← sort_clients_deadline(clients);
04:   for i = 0 → n do
05:     LTc(i) ← calc_transaction_length(clients(i), dparm);
06:   end for
07:   LT ← LTc(0);
08:   while j=1 do
09:     for i = 0 → n do
10:       k ← clients_dl(i);
11:       tR(k) ← calc_response_time(clients, k, LT, dparm);
12:       j ← 0;
13:       if tR(k)>clients(k).Dc then
14:         LT ← LTc(k);
15:         j ← 1;
16:       end if
17:     end for
18:   end while
19:   return LT;
20: }
```

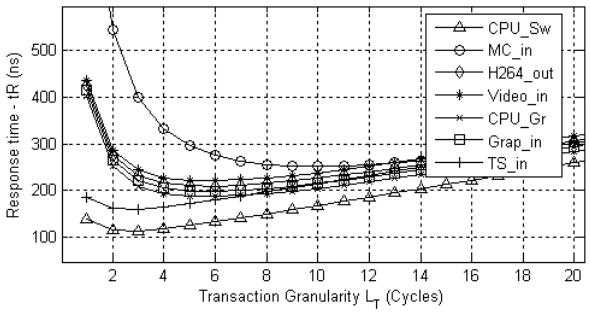

Fig. 2. Latency-based response time analysis for the STB as a function of the transaction granularity for heterogeneous clients sharing the DRAM channel.

B. Case Study – Digital Television Set-Top Box

In this section, we apply the memory model analysis to a case study on digital video processing multimedia SoC to implement a digital television set-top box. The set-top box is composed by 7 IPs accessing a single DRAM channel, with heterogeneous access patterns. A CPU has two client interfaces differentiated by the system address map: video graphics (*CPU_Gr*) and software instruction/data (*CPU_Sw*). A graphics display module and composition uses two client interfaces: *Grap_in and Video_in.* A video decoding and storing module uses two client interfaces: *H264_out* and *MC_in*. The bitstream received from the antenna is also stored in memory through the *TS_in* client interface.

Table IV shows bandwidth, data transfer access duration and deadline requirement for the 7 clients. Clients *MC_in* and *CPU_in* have the most critical deadline requirement. The studied case was modeled to a DDR3-1066 memory with 64 bit wide data bus (tCK = 1.875ns, *tRC* = 50.625ns, *tRL* = 15ns and *tCCD* = 7.5ns). Fig. 2 shows the worst-case response time obtained with the latency-based analysis for the study case using equation (1) applied to the clients presented in Table IV. The transaction granularity influences on the response time for each client connected to the memory subsystem is presented in Fig.2. There is a compromise between the transactions granularity and the response time, that can be reduced significantly adjusting the L_T parameter. Simulated results presented in Table IV shown an average latency improvement of 29.4%.

C. Architectural Implementation Results

The proposed arbitration algorithm was implemented and validated using VHDL language. This implementation executes the arbitration algorithm calculus in a single clock cycle. Initial implementation results showed the ability of this architecture to manage clients' access requests dynamically in single clock cycle. The hardware cost required to support the runtime adaptive arbitration process and the adaptation time depends on the number of clients connected to the memory subsystem. The adaptive arbiter is implemented in a Virtex-6 FPGA technology supporting two clients using only 235 slice registers, 1028 slice LUTs and 2 DSP48E modules. Additional clients connected to the arbiter increases architecture cost in average of 90 slice registers, 355 slice LUTs and 1 DSP48E module per client.

The clock frequency needed to interface with a DDR3-1066 memory is 267 MHz was not attained in this first implementation of the adaptive arbiter. A 90 MHz clock frequency was obtained in an implementation for 7 clients. The design is being optimized to a new implementation with internal pipeline for the arithmetic units, sorting algorithm hardware and multi-stage calculus of the equations. With the pipelined architecture, the adaptive arbiter will be able to run at the desired clock frequency, taking one decision per clock cycle and adapting in estimated $n^2 + n + 7$ clock cycles, where n is the number of clients.

IV. CONCLUSIONS

This paper presents an adaptive algorithm to manage at run-time the access concurrency for heterogeneous clients to a shared memory channel. Simulation results shown that the performance can be largely increased if requests are monitored and classified according to the access pattern of clients, considering deadline requirements and sequential access duration. The arbiter is than able to schedule the clients' requests in an optimal way. Simulated results showed an average latency improvement of 29.4% for a set-top box SoC implementation with 7 heterogeneous clients and a single DDR3 memory channel. Effective bandwidth seen by the clients is consequently increased proportionally, due to the use of high throughput of DRAMs in burst access.

REFERENCES

[1] M. D. Gomony, B. Akesson, and K. Goossens, "Architecture and optimal configuration of a real-time multi-channel memory controller," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, vol., no., pp.1307-1312, March 2013.

[2] M. D. 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 , pp.51-56.

[3] K.B. Lee and T.S. Chang: "Essential Issues in SOC Design", chap. SoC Memory System Design. Springer Netherlands, pp. 73–118, 2006.

[4] P. V. Wolf and J. Geuzebroek, "Soc infrastructures for predictable system integration," in Design, Automation Test in Europe Conference Exhibition (DATE), 2011, pp. 1–6.

[5] B. Akesson, K. Goossens, "Architectures and modeling of predictable memory controllers for improved system integration," in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011.

[6] M. K. Jeong; E. M. Sudanthi, N. C. 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, 2012, pp.850–855.

[7] A.C. Bonatto; M. Negreiros; A.B. Soares; and A.A. Susin, "Towards an Efficient Memory Architecture for Video Decoding Systems," *Computing System Engineering (SBESC), 2012 Brazilian Symposium on* , vol., no., pp.198-203, 2012.

[8] H. Shah, A. Knoll, and B. Akesson. "Bounding SDRAM interference: detailed analysis vs. latency-rate analysis," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, vol., no., 2013.

[9] Micron Technology inc., 4 Gb DDR3 SDRAM Features. 2009.

[10] S.-J. Cho; A. Jaewoo; C. Hyojin; W. Sung, "Performance analysis of multi-bank DRAM with increased clock frequency," *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on* , vol., no., pp.2477-2480, 2012.

[11] Cuppu, V.; Jacob, B., "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?," *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on* , vol., no., pp.62,71, 2001.