Hussain, T.; Palomar, O.; Unsal, O.; Cristal, A.; Ayguade, E.; Valero, M.; Gursal, S.A., "AMMC: Advanced Multi-Core Memory Controller," Field-Programmable Technology (FPT), 2014 International Conference on , vol., no., pp.292,295, 10-12 Dec. 2014. doi:10.1109/FPT.2014.7082802

AMMC: Advanced Multi-core Memory Controller

Tassadaq Hussain[1,2], Oscar Palomar[1,2], Osman Unsal[1], Adrian Cristal[1,2,3], Eduard Ayguadé[1,2], Mateo Valero[1,2]
[1] Computer Sciences, Barcelona Supercomputing Center, Barcelona, Spain
[2] Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Barcelona, Spain
[3] Artificial Intelligence Research Institute (IIIA), Centro Superior de Investigaciones Científicas (CSIC), Barcelona, Spain
Email: {first}.{last}@bsc.es

***Abstract*— In this work, we propose an efficient scheduler and intelligent memory manager known as AMMC (Advanced Multi-Core Memory Controller), which proficiently handles data movement and computational tasks. The proposed AMMC system improves performance by managing complex data transfers at run-time and scheduling multi-cores without the intervention of a control processor nor an operating system. AMMC has been coupled with a heterogeneous system that provides both general-purpose cores and application specific accelerators. The AMMC system is implemented and tested on a Xilinx ML505 evaluation FPGA board. The performance of the system is compared with a microprocessor based system that has been integrated with the Xilkernel operating system. Results show that the AMMC based multi-core system consumes 48% less hardware resources, 27.9% less on-chip power and achieves 6.8x of speed-up compared to the MicroBlaze-based multi-core system.**

I. Introduction

Latest multi-core architectures require both programmability and performance and combine different types of cores, becoming heterogeneous systems. To get programmability, a part of the program is executed on general-purpose cores. To achieve performance and to increase power efficiency, compute intensive tasks are mapped into separate hardware accelerators or application-specific processors. The dedicated application specific accelerator cores have small footprint and low power dissipation and feature high performance [1].

To overcome the *memory wall* and to reduce the system power, a memory system is needed that supports cores with low frequency and low complexity, has efficient local memory and data management, with an intelligent scheduler while supporting a programming model that manages memory accesses in software so that hardware can best utilize them. In this work, we have integrated a memory controller with a heterogeneous multi-core system having Application Specific Hardware Accelerators (*ASHA*) and Scalar Soft Processor (*SSP*), that we term AMMC (Advanced Multi-core Memory Controller). Some salient features of the AMMC are given below:

- The AMMC based system handles heterogeneous (*SSP* and *ASHA*) cores using *Symmetric* and *Asymmetric* scheduling policies, without the support of a master core nor operating system.
- Regular and irregular access patterns of heterogeneous multi-cores are described using a separate *Descriptor Memory*, which reduces the on-chip communication time and run-time address generation overhead.
- The AMMC *Address Manager* and *Scheduler* handles regular and irregular pattern requests of a heterogeneous multi-core system, provides precise timing and allows scheduling mode to be changed at runtime.
- When compared to the baseline multi-core system implemented on the Xilinx FPGA, the AMMC multi-core system achieves 6.8x of speed-up, transfers datasets up to 1.95x faster, consumes 48% less hardware resources and 27.9% less on-chip power.

The research leading to these results has received funding from the European Research Council under the European Unions 7th FP (FP/2007-2013) / ERC GA n. 321253. It has been partially funded by the Spanish Government (TIN2014-34557).

II. Advanced Multi-core Memory Controller

In this section, we describe the AMMC system. The architecture (shown in Figure 1) is divided into five units: the *Bus System* (A), the *Local Memory Unit* (B), the *Memory Manager* (C), the *Scheduler* (D) and the *Pattern Aware SDRAM Controller* (E). The main units of AMMC are shown in Figure 1, as well as the *Multi-Core System*, that executes the applications. The *Multi-Core System* can have general purpose processors, application specific accelerator cores or a combination of both types. The *Bus System* [2] provides a link between AMMC and the *Multi-Core System*. The AMMC *Memory Unit* stores both data and the access pattern descriptors in *Specialized Memory* and *Descriptor Memory* respectively. Each processing core has a separate *Specialized Memory* [3] and a number of *Descriptor Memory* blocks. The descriptors are programmed at compile-time, providing information of the memory access patterns and their priorities. At run-time, the AMMC *Scheduler* receives multiple memory read/write requests from the *Multi-Core System* and selects a processing core, depending upon its priority level and scheduling policy. The *Scheduler* forwards the memory request to the *Memory Manager*. The *Memory Manager* is divided into the *Address Manager* and the *Data Manager*. The *Address Manager* takes a *Task ID* from the AMMC *Scheduler* and fetches its *Descriptor Memory*. Depending on the access pattern the *Address Manager* uses single or multiple descriptors, maps and rearranges addresses in hardware. The *Address Manager* saves mapped addresses into its *Address Buffer* for further reuse. The *Data Manager* improves the *Computational Intensity* [3] by organizing and managing the memory accesses. For a core processing a single *computed point*, the maximum achievable (ideal) *Computational Intensity* is 1. The *Data Manager* accesses the elements in the form of patterns which are required for a single output ($Computed_{element}$). After accessing the first access pattern, the *Data Manager* reuses and updates data where required. The *Pattern Aware SDRAM Controller* [3] is used to transfer data between main memory and the *Specialized Memory*.

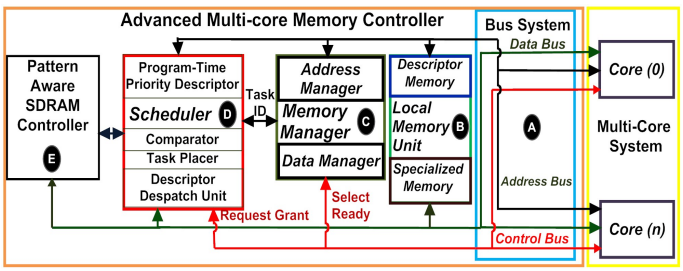
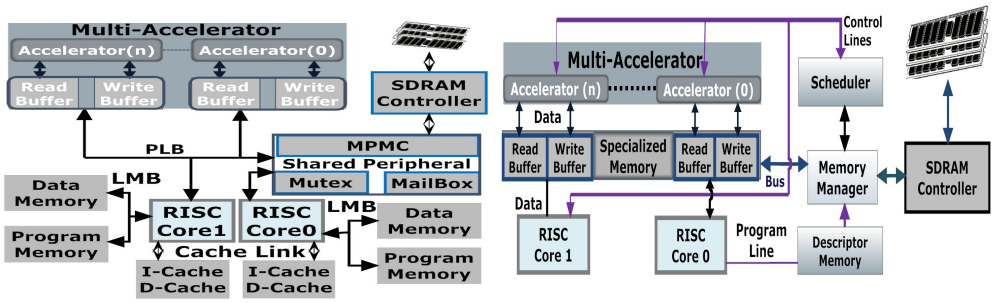


Fig. 1. Architecture of the Advanced Multi-core Memory Controller



(a) (b)

MicroBlaze System	Flip Flops	LUTs	BRAMs
Master Core	2606	2664	14
Timer	358	287	
Mutex Core	145	95	
MailBox Core	350	317	3
Bus System	145	342	
SDRAM Controller	4162	2719	13
Total	7766	6424	30

(c)

AMMC Units	Flip Flops	LUTs	BRAMs
Timer	152	110	
Scheduler	445	390	2
Memory Manager	850	727	3
Bus System	105	90	
SDRAM Controller	2457	1602	13
Total	4009	2919	18

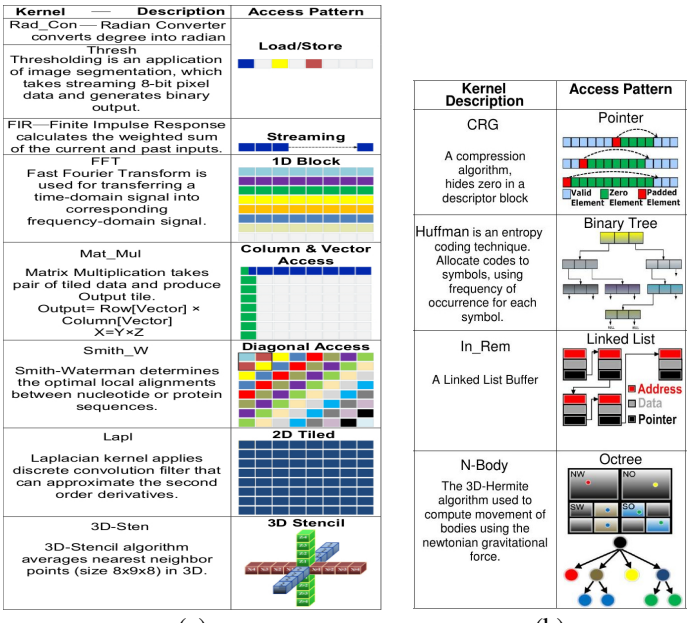
(d)

Fig. 2. Multi-Core Systems: (a) MicroBlaze (b) AMMC (c) MicroBlaze Resource Utilization (d) AMMC Resource Utilization

The AMMC *Descriptor Memory* holds memory access and scheduling information of applications running on the *Multi-Core System*. Each processing core has a separate block of *Descriptor Memory*. The set of parameters for a *Descriptor Memory* block includes *Command*, *Task ID*, *External Address*, *Priority*, *Size*, *Stride* and *Offset*. Command specifies whether to read or write data. The *Task ID* and *External Address* parameters hold the address of processing core (*buffer*) and main memory (*SDRAM*) data set respectively. The *Priority* defines the order in which memory accesses are entitled to be process. The parameters *Size* and *Stride* define the type of memory access. The *Offset* register field is used to point the next linked memory access pattern.

The AMMC *Scheduler* manages and controls the run-time requests and programmed priorities of processing cores. Each processing core's *request* includes a read and write memory operation. At program-time, each processing core is assigned a priority value along with the *Task ID*, which are placed in the *Program-Time Priority Descriptor* (shown in Figure 1). The processing cores are categorized into three states, *busy* (core is processing on local buffer), *requesting* (core is idle), and *request & busy*. In the *request & busy* state the core is assumed to have double or multi buffers. During this state, the core is processing the data of one buffer while making a request to fill another buffer.

AMMC *Scheduler* supports two scheduling strategies of the *requests* of the cores: *symmetric* and *asymmetric*. The scheduling policies are programmed statically at program-time and are executed by hardware at run-time. In **Symmetric** multi-core strategy, the AMMC *Task Placer* (Figure 1) manipulates the incoming requests in FIFO (First in First out) order and places them in the *Dispatch Descriptor*. The **Asymmetric** strategy uses the priority specified for each core and incoming requests. Each core is assigned a fixed priority at program-time, which is placed in *Program-Time Priority Descriptor*. At run-time, the *Scheduler* accumulates requests from the *Multi-Core System*. The *Comparator* and *Task Placer* maintain them in the *Dispatch Descriptor*. The *Comparator* takes requests from multiple processing cores, compares them with programmed priorities and forwards the results to *Task Placer*. The *Task Placer* places the requests in the *Descriptor Dispatch Unit* and executes requests only if it is ready to run, and there are no higher priority cores that are in ready state. The *Dispatch Descriptor* executes processing core requests sequently.



(a) (b)

Fig. 3. Application Kernels: (a) Regular Access Pattern (b) Irregular Access Pattern

III. Experimental Framework

In this section, we describe the MicroBlaze- and AMMC-based *Multi-Core Systems* and the rest of our experimental setup. A Xilinx ML505 evaluation FPGA board is used to test the *Multi-Core Systems*. The Xilinx Integrated Software Environment and Xilinx Platform Studio are used to design the *Multi-Core Systems*. Xilinx Power Estimator does the power analysis. The section is divided into three subsections: the *Computation Units*, the *MicroBlaze based Multi-Core System* and the *AMMC based Multi-Core System*.

A. Computation Units

There are two cores in our heterogeneous system: *ASHA* cores execute application kernels with regular memory accesses while *SSP* cores execute application kernels with irregular memory access patterns. Figure 3 lists all applications used in our experiments. The *ASHA*s are generated by the ROCCC [4] compiler. We have chosen the MicroBlaze *SSP* to implement the general purpose cores of our system. Microblaze is an RISC *SSP* architecture, optimized and implemented with FPGA resources. The multi-core system includes 2 MicroBlaze cores that run 2 applications each and 8 ASHA cores.

B. MicroBlaze-based Multi-Core System

The MicroBlaze-based *Multi-Core System* is used as baseline (Figure 2(a)). The resources utilized by the MicroBlaze based multi-core system is shown in Figure 2(c). Each general purpose core has 32KB of data cache, that is implemented using BRAM. The design uses Xilinx Cache Links (IXCL/DXCL) for I-Cache and D-Cache memory accesses respectively. MicroBlaze instruction prefetcher improves the system performance by using the instruction prefetch buffer and instruction cache streams.

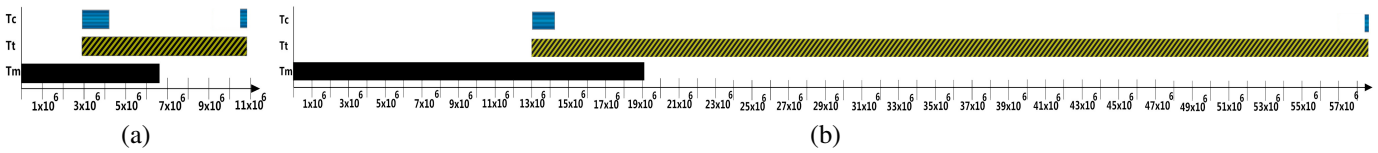


Fig. 4. Symmetric System Performance: (a) AMMC (b) MicroBlaze

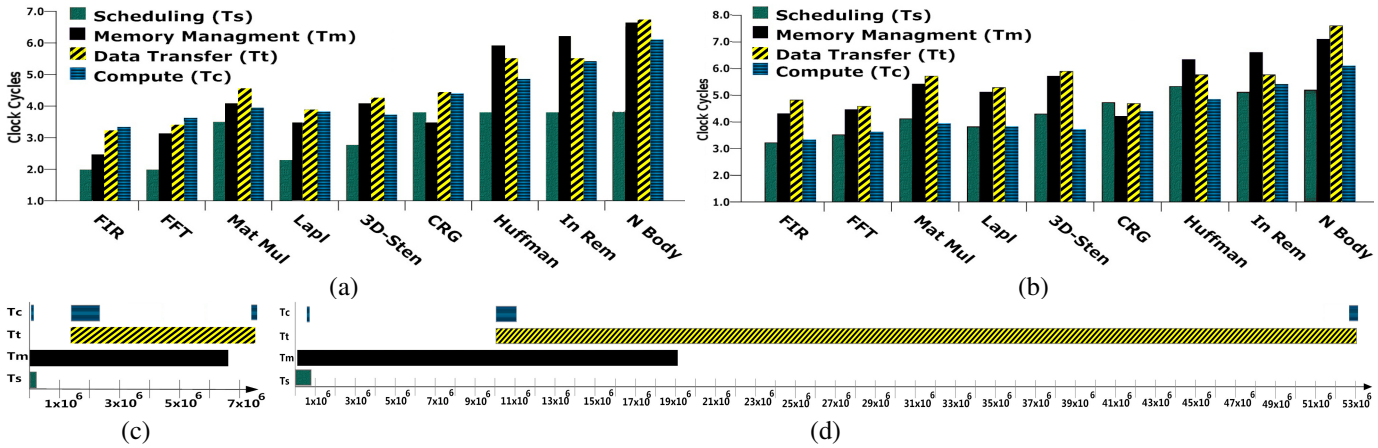


Fig. 5. Asymmetric System Performance: (a) AMMC (b) MicroBlaze (c) & (d) AMMC & MicroBlaze Systems Pipeline and Overlap Time Period

TABLE I. ASYMMETRIC SCHEDULING PRIORITY POLICIES

Kernels	FIR	FFT	Mat_Mul	Lapl	3D-Sten	CRG	Huffman	In_Rem	N-Body	Speed-ups
Symmetric	1	1	1	1	1	1	1	1	1	5.47x
Asymmetric										
Group I	1	4	5	3	2	6	7	8	9	6.84x
Group II	2	3	4	5	1	8	6	9	7	5.83x
Group III	9	6	5	4	8	4	3	2	1	3.45x
Architecture	1	1	1	1	1	2	2	2	2	5.42x

One of the MicroBlaze softcores (Core 0, Figure 2(a)) is the master core and is used to schedule the memory requests and to manage data transfer between multi-cores and main memory (SDRAM). The MicroBlaze cores use Xilkernel a small light-weight easy-to-use Real-Time Operating System (RTOS). Its API performs scheduling, inter-process communication and synchronization with *POSIX* threads (pthreads). From the main function, application spawns into multiple statically declared threads using the pthread library. Each thread controls a single application kernel and manages its memory patterns.

C. AMMC based Multi-Core System

Figure 2(b) shows the implementation of an AMMC-based *Multi-Core System*. *SSP* cores do not integrate any cache but use the local memory provided by AMMC. Similarly, there is no need for an RTOS like Xilkernel. The resources consumed by each AMMC unit is shown in Figure 2(d).

IV. RESULTS AND DISCUSSION

This section analyzes the results of experiments conducted on AMMC and MicroBlaze based system. The experiments are characterized into two subsections: *System Performance* and *Area & Power*.

A. Multi-Core System Performance

The system performance is measured by executing application kernels simultaneously using different scheduling policies, on AMMC and MicroBlaze based systems. Due to the confined FPGA resources, 5 *ASHA* and 2 *SSP* cores are integrated with the *Multi-Core System*. The execution time of both systems is categorized into four factors: scheduling time (*Ts*), memory management time (*Tm*), data transfer time (*Tt*) and computation time (*Tc*). *Ts* holds the arbitration (request, grant and wait) time among the on-chip scheduling. *Tm* comprises the address generation and data management time. *Tt* presents the data access time from external memory. It includes address mapping from physical address space to SDRAM address space, interface timing and synchronization. *Tc* holds the computation time of the application kernels. To measure the overlap and processing time, each application kernel is assigned four timers which count *Ts*, *Tm*, *Tt* and *Tc* clocks.

In the symmetric scheduling policy, the requests are treated with the FIFO method, which removes the scheduling time. Figures 4(a) and (b) present the overlapped/pipelined time of AMMC and MicroBlaze systems respectively. X and Y axis present clock cycles and execution time factors, respectively. While running the *Multi-Core System* using symmetric scheduling, the results show that the AMMC system achieves 5.47x of speed-up. The current *Computation Units* contain application kernels with different access patterns. The symmetric scheduling policy gives higher priority to application kernels with many memory requests. These requests add on-chip bus and memory access delays, therefore the AMMC system does not fully overlap *Tm* & *Tt*. These delays can be decreased by executing *Multi-Core System* with asymmetric scheduling policy.

We categorize the asymmetric scheduling policy into two types; the memory access based asymmetric policy and the architecture based asymmetric policy (shown in Table I). The memory access based asymmetric policy assigns priorities to the application kernels with respect to their access patterns and is further categorized into three groups. In Group I, the highest priorities (1) are allocated to application kernels having less memory requests and dense access patterns. For example, the applications having multiple read/write requests are given low priorities. To check the sensitivity of asymmetric scheduling execution of the assigned priorities, the priorities of Group I are slightly varied in Group II. In Group III, the priorities

are assigned to check the fairness of applications for priorities not assigned properly. For example, the highest priority (1) is allocated to application kernels having the largest amount of memory requests. Like MicroBlaze Xilkernel scheduling model, the AMMC scheduling policies and memory accesses are configured statically at program-time. Unlike Xilkernel, the requests are managed and executed by hardware at run-time. The memory access based asymmetric policy performs load balancing and reduces on-chip communication and memory management delay.

Figures 5(a) & (b) present clock cycles of AMMC and MicroBlaze systems respectively, while executing application kernels simultaneously using memory access based asymmetric scheduling policy. X (logarithmic scale) and Y axis present clock cycles and application kernels, respectively. Each bar represents *Ts*, *Tm*, *Tt* and *Tc*. While running all application kernel together using the asymmetric scheduling, the results show that the scheduling, memory manager and memory transfer of AMMC based system are 21x, 2.9x and 7.1x faster respectively, compared to the MicroBlaze based system. The computation units execution time (*Tc*) remains the same for both systems. Figures 5(c) and (d) present the overlapped/pipelined time of AMMC and MicroBlaze systems respectively. The *Tc* of all application kernels is overlapped (shown in Figure 5(c)). In the AMMC system, *Tt* and *Tm* are dominant for the regular and irregular application kernels respectively. As all AMMC units operate in parallel, AMMC overlaps all other units under the unit that consumes more time. For example for regular application kernels *Ts*, *Tm* and *Tc* are overlapped under *Tt*. The MicroBlaze based system overlaps *Tc* & *Ts* completely and partially overlaps *Tm* and *Tt* (shown in Figure 5(d)). While running all application kernel together using the asymmetric scheduling with priorities of Group I, the results show that the AMMC based system achieves 6.84x of speed-up compared to the MicroBlaze based system. While executing application kernels with priories of Group II and Group III, the AMMC based system achieves 5.83 and 3.45x of speed-up respectively. The AMMC asymmetric scheduling policy manages system resources (Application code, On-Chip Off-Chip Memory) of the *Multi-Core System* without the support of the operating system.

In the architecture based asymmetric policy, the *Computation Units* are assigned priorities depending upon their instruction set architecture, execution and communication (request/grant) speed. The architecture based asymmetric priorities are shown in Table I. All the cores of one type get the same priority. The priority 1 executes *ASHA* core requests with higher priority. Requests having same priorities are executed in FIFO order. While running the *Multi-Core System* using the architecture based asymmetric scheduling policy, the results show that the AMMC based system achieves 5.42x of speed-up compared to MicroBlaze based system. For performance evaluations, we analyzed that the priority based scheduling has the potential for supporting scalability and load balancing.

B. Area & Power

Xilinx V5-Lx110T device dissipates 3.15 watts of on-chip static power, while running the MicroBlaze based system. The AMMC system draws 2.27 watts of on-chip power on a V5-Lx110T device. While comparing the AMMC and MicroBlaze systems without slave units (accelerators and processor), results show that AMMC system consumes 48% fewer slices and 27.9% less on-chip static power than the MicroBlaze system. The AMMC provides low-power and simple control characteristics by rearranging data accesses and utilizing hardware units efficiently.

V. Related Work

Marchand et al. [5] have developed software and hardware implementations of the Priority Ceiling Protocol that control the multiple-unit resources in a uniprocessor environment. Yan et al. [6] has designed a hardware scheduler to assist the synergistic processor cores (SPCs) task scheduling on heterogeneous multi-core architecture. The scheduler supports first come first service (FCFS) and dynamic priority scheduling strategies. It acts as helper engine for separate threads working on the active cores. The scouting hardware thread [7] tends to reduce latency, but also optimizes memory bandwidth usage by predicting memory accesses and by prioritizing valuable memory traffic using a separate core. The information of memory accesses is stored thus helping the scouting core to fetch and update data from the cache. The AMMC holds information of memory patterns in the form of *Descriptor Memory*. Currently, accessed patterns are placed in the address manager of AMMC. The AMMC monitors the access patterns without using a separate core and reuses these patterns for multiple cores if required.

Hussain et al. [8] [9] discussed a programmable pattern based memory controller architecture. The design is appropriate for data intensive applications with regular access patterns only. He also proposes a controller [3] that supports irregular applications running on a single core. Whereas in AMMC, we present a mechanism that supports both application-specific accelerators and RISC cores in a heterogeneous multi-core system having regular and irregular memory access patterns.

VI. Conclusion

In this work, we have proposed AMMC that schedules multi-core operations while taking processing, scheduling, memory management and memory transfer into account. The AMMC architecture supports two types of cores: the general purpose RISC core and application specific hardware accelerator core. The AMMC improves the system performance by reducing the speed gap between accelerators/processors and memory and by scheduling/managing complex memory patterns without master core intervention. The AMMC system is implemented and tested on a Xilinx ML505 evaluation FPGA board. The performance of the system is compared with a microprocessor based system that has been integrated with the Xilkernel operating system. Results show that the AMMC based multi-core system consumes 48% fewer slices and 27.9% less on-chip static power and achieves 6.8x of speed-up compared to the MicroBlaze-based multi-core system having real time operating system.

References

[1] András Vajda et al. *Programming many-core chips*. Springer, 2011.

[2] Tassadaq Hussain et al. MAPC: Memory Access Pattern based Controller. In *24th International Conference on FPL*, 2014.

[3] Tassadaq Hussain et al. Advanced Pattern based Memory Controller for FPGA based Applications. In *24th International Conference on HPCS*.

[4] Jason Villarreal et al. Designing modular hardware accelerators in c with roccc 2.0. In *FCCM 2010*.

[5] P. Marchand and P. Sinha. A hardware accelerator for controlling access to multiple-unit resources in safety/time-critical systems. Inderscience Publishers, April 2007.

[6] L. Yan et al. Hardware Assistant Scheduling for Synergistic Core Tasks on Embedded Heterogeneous Multi-core System. In *Journal of Information & Computational Science (2008)*.

[7] Shailender et al. Chaudhry. Simultaneous speculative threading: a novel pipeline architecture implemented in sun's rock processor. *ACM*, 2009.

[8] T. Hussain, M. Shafiq, M. Pericas, N. Nacho and E. Ayguade. PPMC: A Programmable Pattern based Memory Controller. In *ARC 2012*.

[9] Tassadaq Hussain and Amna Haider. PGC: A Pattern-Based Graphics Controller. *Int. J. Circuits and Architecture Design*, 2014.