International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 6, No. 2, July 2017, pp. 69~81
ISSN: 2089-4864, DOI: 10.11591/ijres.v6.i2.pp69-81

Software and Hardware for Managing Scratch Pad Memory

Chabane Hemdani[1], Rachida Aoudjit[2], Mustapha Lalam[3], Khaled Slimani[4]
Laboratoire de Recherche en Informatique (LARI) BP n 17 RP, University of Tizi Ouzou, Algeria

Article Info

Article history:

Received Apr 2, 2017
Revised Apr 20, 2017
Accepted May 7, 2017

Keywords:

Core
Embedded Systems
FPGA
Scratch Pad Memory

ABSTRACT

This paper proposes a low-cost architecture to improve the management SPM (Scratch Pad Memory) in dynamic and multitasking modes. In this context, our management strategy SPM based on Programmable Automaton implemented in Xilinx Vertex-5 FPGA is entirely different from prior research works. SPM is generally managed by software (by a strong programming logic or by compilation). But our Programmable Automaton facilitates access to SPM in order to move code or data and liberates space in SPM. After this step, software takes over content management of SPM (what part of code or data should be placed in SPM, locates spaces of Heap and Stack). So the performance of the programs is actually improved thanks to minimization of the access latency at the DRAM (Dynamic Random Access Memory or Main Memory).

Corresponding Author:

Mustapha Lalam,
Laboratoire de Recherche en Informatique (LARI)
BP n 17 RP University of Tizi Ouzou, Algeria.
Email: lalamustapha@yahoo.fr

1. INTRODUCTION

A well finished conception of the memory hierarchy subunits contributes to the improvement of the performances of an Embedded System in particular on the timing, the scheduling, the energy consumption, the multi-tasking and the predictability. It is in this context that locates our idea to manage effectively a local static memory SRAM to be used as Scratch Pad Memory (SPM). Our approach on use of this SPM is inspired from previous works of the literature while proposing an original hybrid solution that is to manage SPM by hardware (Programmable Automaton) and by System calls. SPM endowed with its coprocessor is connected to a core of a multi-core system. Every core has its own SPM and this one can be accessible by other cores for a better efficiency of the Multi-Tasking and of Multi-processing.

SRAM on chip, called SPM, is a small memory of fast access which is located in a space of separate addressing of the DRAM but connected to the same addresses bus and to the same data bus [1]. SPM is managed by software. In other words, the stored data in SPM are managed by Programmers or Compilers [2]. It is possible to have Cache memory and SPM in the same Embedded System. Contrary to the SPM, the cache is managed by hardware. But both have a latency of access of a cycle; on the other hand the access memory off chip (generally a DRAM) asks for several cycles (from 10 to 20 cycles). One of the main differences between the SPM and the cache is that SPM guarantees an access time of a cycle while the cache is subject to the compulsory, capacity and conflict misses [1]. Our solution founded itself on the works proposed in [3] in the partitioning of SPM but we propose an original management of SPM by combining hardware and software (OS level). For that purpose, we endow SPM of an implemented function in FPGA circuit (Xilinx Vertex-5 FPGA).

This paper is composed of 6 sections. The first Section gives an overview onto the state of the art of SPM. The section 2 shows the organization of SPM in frames. We then chain on the structure SHFS (Stack Heap Frame Structure) stored in DRAM memory. We introduce functions of Automaton in the section 3. The section 4 recommends various Algorithmic State Machines that Automaton executes as well as DMA mode

(Direct Memory Access) executed under aegis of the Operating System (OS). The following section completes the hardware of the automaton under the form of a circuit FPGA realized under EDK. The section 6 gives an insight of simulation of the coprocessor which we recommend and the resources of the circuit FPGA used during the synthesis for realizing the coprocessor.

2. STATE OF THE ART

The use of SPM showed an improvement of the performance and a reduction of the energy [1], [3]-[5], [16]-19]. In 1997, [1] proposed a static strategy for partitioning scalar and array variables in an application code into SPM and off-chip DRAM accessed through data cache. They described a method of partition that shows the usefulness on-chip SPM in addition to a data cache. [9] Proposed SPM scheme which can be embedded onto FPGA's circuit or multi-core processors chips as a coprocessor to decrease the memory access time of the key/value pairs used directly from SPM to accelerate MapReduce applications. The proposed MapReduce scratchpad memory is used to replace the Reduce stage with a single special memory unit that is used to store and automatically accumulate the values of the keys in MapReduce application. This architecture proposed in [9] does not correspond to our initiative of design of the coprocessor which manages SPM. According to [5], [8] presented a hardware/software approach for dynamic management scheme of SPM. Their approach introduces a hardware component designed specially to manage the copying of instructions from the main memory into SPM. They utilize instructions execution frequency information to model applications as graph and perform graph partitioning to determine locations within the program for initiating the copying process. The Authors of [7] showed a different approach to static usage scheme for SPM by mapping applications on the existing hardware [5]. The paper [10] proposed a runtime memory management approach for SPM at the OS-level that can be combined with other compile-time approaches. The OS memory manager takes annotations inserted into the code by the programmer as hints to choose the most appropriate memory (i.e. Main Memory or SPM) for each allocation. Experimental results confirm the approach's efficiency when compared to a similar compile-time technique [10].

The technique which assigns heap dated in the SPM was proposed in [14]. Despite of being the first technique that allocates heap data to SPM, it follows the dynamic compile-time-approach [10].
According to [5], [6] proposed another static management scheme for recording data and instructions memory. Their scheme uses a polynomial time algorithm for partitioning and instructions into DRAM and SPM. Their results show energy improvement ranging from 39 % to 84 % over a no partitioned SPM. Another work that deals with the management of Heap data was presented in [15]. Paper [11] specified that multi-core systems have been a popular design for high performance Embedded Systems. The Authors propose one polynomial-time algorithm to solve the data allocation problem on multi-core system with exclusive copy of data. The proposed solution reduces time cost of memory accesses by 16.45 % on average. The solution also can reduce the energy cost significantly. There are also several works proposed for SPMs on the multi-core Systems [1], [11]-[13].

Our contribution in this domain consists in managing the SPM by hardware (a circuit of type FPGA called coprocessor) and by software by means of system calls (OS level). SPM endowed with its coprocessor is connected with Core of a multi-core system. Each Core has its own SPM and this one can be accessible by other Cores for a better efficiency of the Multi-Tasking. The coprocessor absorbs certain requests of reading or writing and so the performance of the programs will be actually improved thanks to the minimization of access frequencies to DRAM memory.

3. PARTITION OF SPM INTO FRAMES

The SPM is divided into frames of equal capacities [3]. Each of (n-1) first frames contains two parts:

a. Space for heap,
b. Space for stack.

Each of these two spaces is glanced through an index pointer (heap pointer and Stack point respectively). The last frame of the SPM (SHF_{n-1}) contains the profilers of (n-1) frames, as shown in Figure 1. Every task has its profiler. What do we find into the profiler of a frame SHF_k that belongs to the task k? The profiler SHF_k contains:

a. outset address of the stack in BS (Backup Storage) situated in Main Memory,
b. outset address of the heap in BS,
c. outset address of the stack in the SPM,
d. Outset address of the heap in the SPM.

The stack can be on one or on several frames SHFs. The heap can also be on one or several frames SHFs. SHF is independently assigned of the other SHFs and there is no internal relation between them. Every

SHF is completely used until there is not a free place any more. No new SHF is assigned to a task on the condition that all the SHFs assigned to this task are completely filled. This way of operating implies three interests:

a. No empty fragment,
b. Reduction of the complexity in the track and in the preservation of the structures which are used to manage various stacks and heaps,
c. Opportunity to use multitasking.

Partition of SPM into frames is as follows Figure 1:

A mechanism is necessary to manage these SHFs [3]. For that purpose, we plan a structure SHFS (Stack and Heap Frame Structures) taken up residence in Main Memory which will specify the state of every SHF and its membership. When a request of a task is thrown to have a free frame SHF, the structure SHFS will be referenced by system primitive to find a free possible one SHF in the SPM and attribute it to the task. Across SHFS, the tasks are informed if they can make a request for a SHF and where it is exactly situated. If there is not a free space any more in SPM and we cannot allocate a new SHF, we request the Main Memory to assign a space for the stack and a space for a heap for every task at the level of the BS. We associate an additional structure BS taken up residence in Main Memory for the stack and the heap of every task. It is the role of OS to insure the management of the structures SHFS and BS. All BS are created and maintained in the Main Memory [3]. They will be unused if there are enough spaces in SPM. The contents of a BS will not be again transferred towards the SHFs [3]. SPM accesses are less expensive in energy consumption than the accesses for DRAM (off chip DRAM). The SHFS structure in Main Memory is as follows Figure 2:

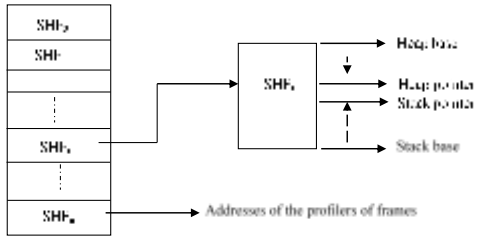


Figure 1. Organization of the SPM into frames [3]

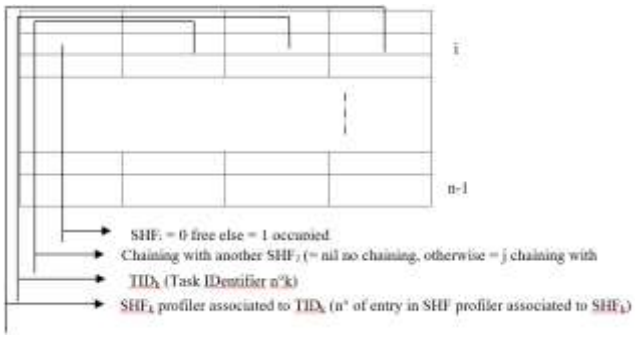


Figure 2. Structure of SHFS (Stack Heap Frame Structure) in DRAM Memory.

4. ROLE OF AUTOMATON

The Automaton fills four features:

a. Solicits the DMAC (Direct Memory Access Controller), through the OS for the Data Transfer by DMA from the DRAM towards the SPM or vice versa,
b. Reading or writing in the Heap of the SPM,
c. Push or Pull in the stack of the SPM,
d. If the space of addressing of the request of access of the data is out of the limits of the SPM, it sends back the request towards the upper level of the hierarchy memory.

4.1. DMA Transfer

In DMA access, DMAC (Direct Memory Access Controller) formulates a request of access to Core so that it frees the Buses of addresses and data. As soon as possible, Core informs the DMAC by means of a signal of permission (DGRANT) to take up buses and use the SPM. Core puts its Buses in state of high impedance. The DMAC can then reaches SPM by regaining control freed buses by Core.

4.2. Reading or writing in the Heap of the SPM

The automaton receives read request (Load) or write request (store) of the $task_i$ and it accesses to profiler of the $task_i$ situated in SH_{n-1} of SPM to reach Tas_i.

Profile of the TAS_i=(Register SH_{n-1})+TID_i./(TID_i=Task Identifier i)/

4.3. Push or Pull in the stack of SPM

The automaton receives the request of the $task_i$ (Push or Pull) and accesses to profiler of the $task_i$ situated in SH_{n-1} of the SPM to reach $stack_i$.

4.4. Space of Addressing of the Request Outside the Limits of SPM

If the addressing space of the request of access to the data is out of the limits of SPM, the Automaton sends back the request towards the superior hierarchy memory (DRAM). Programmer of code specifies, in its code, if he wants to allocate a dynamic memory in SPM. This can be made, for example, by type MALLOC's instruction or its equivalent. The liberation of the dynamic memory is made, for example, by type FREE's instruction or its equivalent. On the meeting FREE's instruction in code, a procedure (it doesn't belong to the Automaton) reaches the structure SHFS in Main Memory by putting all fields SHF_k of the Task K to 0 (nil), field TDI_K to 0 and releases BS_k. This procedure can be integrated into the code by Programmer of code. Synoptic scheme of Automaton is showed in following Figure 3:

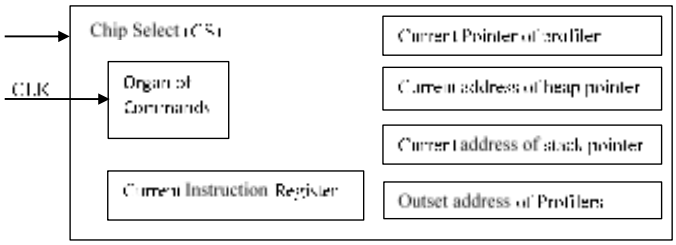
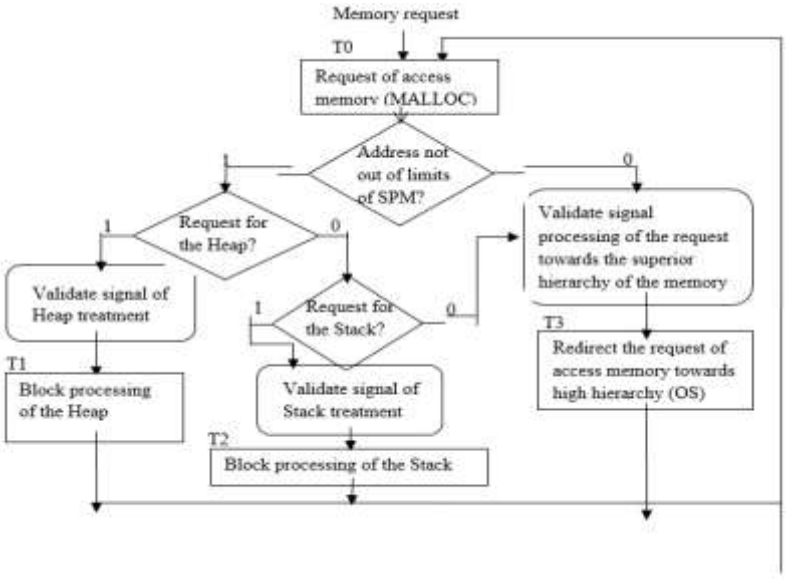


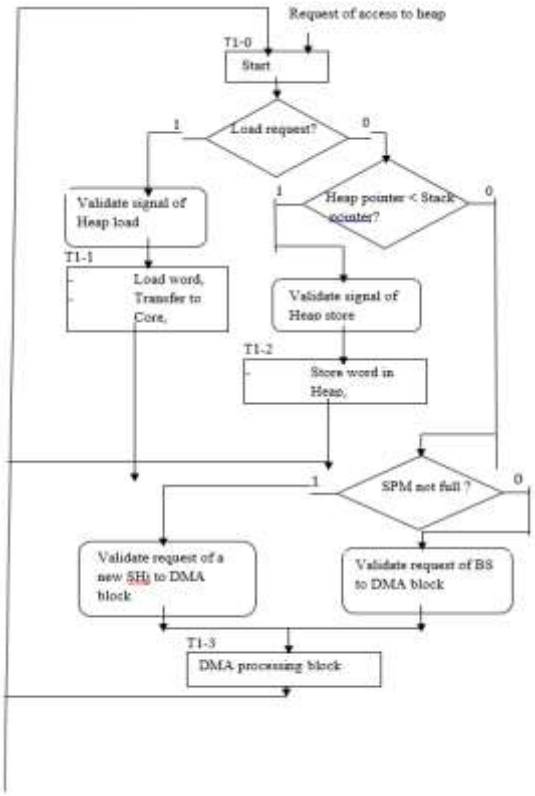
Figure 3. Internal Synoptic scheme of the automaton

5. ALGORITHMIC STATE MACHINE (ASM) DIAGRAM OF AUTOMATON AND DMA MODE

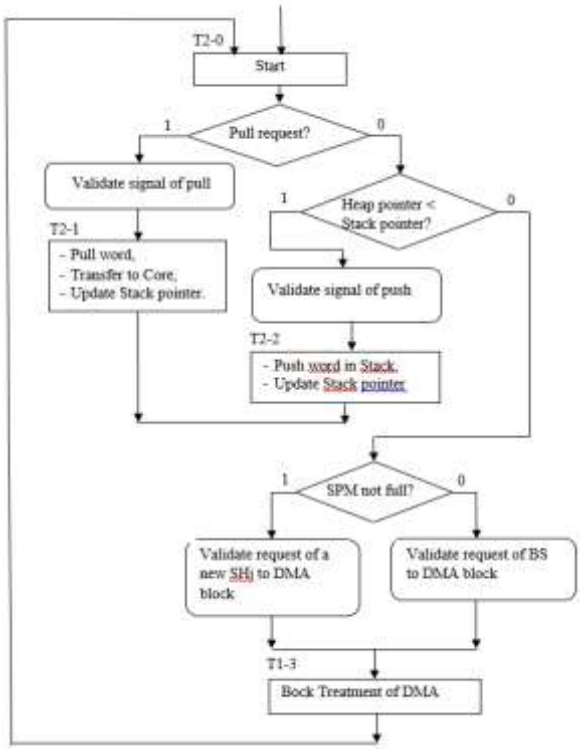
5.1. We present Algorithmic State Machine (ASM) chart of Automaton, diagram easier to understand and less formal that State diagram:



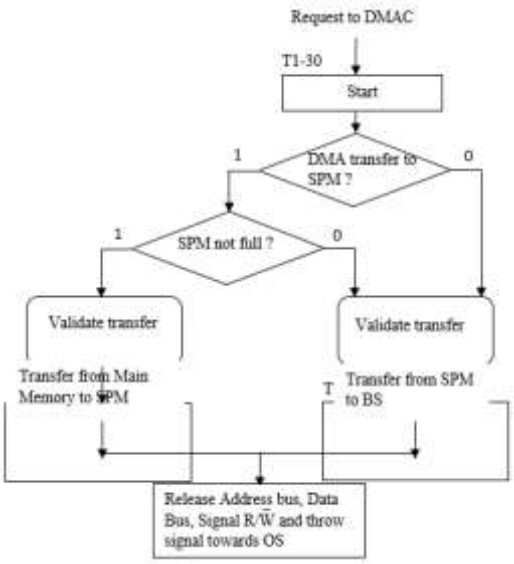
5.2. ASM diagram of Heap block T1



5.3. ASM diagram of Stack block T2



5.4. ASM diagram of DMA T1-3



6. HARDWARE REALIZATION AND SIMULATION OF AUTOMATON EMBEDDED ON A FPGA CIRCUIT

To design a complete system as shown in Figure 4 we used EDK (Embedded Development Kit) of Xilinx who allows us to conceive our own embedded system with a processor Micro blaze on the EDK. Our coprocessor (SPM_IP) is implemented as an IP (Intellectual Property) directly to connect to the Micro blaze through a connection DFSL (Direct Fast Simplex Link) and to the memory SPM as shown in Figure 4).

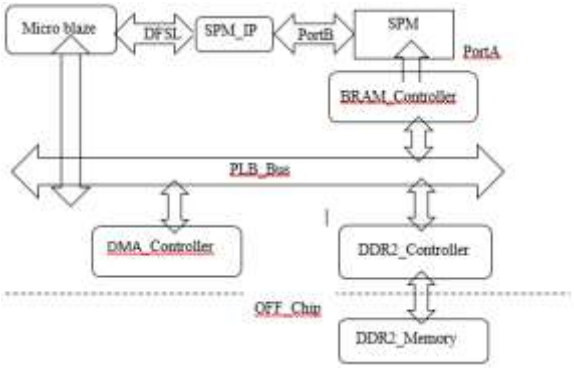


Figure 4. System architecture

Components of the system, The system which we built includes the following components:

6.1. Micro Blaze

Micro Blaze is an embedded soft-core processor designed and optimized for the implementation in the FPGA Xilinx. Micro Blaze is a RISC processor in 3 floors with architecture Harvard and 32 internal registers of 32 bits. The processor contains approximately 70 options of configuration allowing the user to select or to parameterize the internal components according to its needs. Among the configurable options, we may cite: size of the cache, depth of pipeline, integrated peripherals, MMU, bus interfaces etc.

Xi kernel is the operating system which works on our Micro Blaze. Xi kernel includes an API of POSIX. This API allows us to define the various tasks (threads) and the various functions to reach the peripherals of the system.

6.2. SPM

SPM is implemented by using BRAM of 128 KB. The block BRAM is a configurable memory module (implemented in VHDL) which possesses two ports A and B. In our case the port A is connected to the BRAM CONTROLLER which is an interface between the bus PLB and SPM. The port B is connected to the coprocessor.

6.3. DMA Controller

DMA Controller supplies simple services of direct access to the memory (DMA) to peripherals and in devices memory connected to the bus PLB. DMA Controller transfers a quantity of data resulting from an address source towards an address destination without intervention of the processor. In our case, it serves to transfer the data between SPM and the DRAM according to the algorithm previously presented.

6.4. DFSL

FSL (Fast Simplex Link) is a bus of unidirectional communication channel based FIFO used to make a fast communication between two elements of FPGA. The depth of FIFO can achieve 8K. The interface FSL is available on the processor Micro Blaze. In our system we used the DFSL version (Direct FSL) which is implemented without FIFO (to increase the speed of communication), to interconnect our coprocessor to the Micro Blaze.

6.5. PLB bus

PLB (Processor Local Bus) supplies an infrastructure of bus to connect an optional number of peripherals (masters and slaves) in the system. In our system we also find a DRAM of capacity of 256 MB, an interruption controller and a timer.

6.6. Coprocessor SPM_IP

In this part we present the signals of SPM_IP as shown in Figure 5:

Figure 5. Signals of SPM_IP

SPM_IP possesses two interfaces:

6.6.1. DFSL: connects the coprocessor to the Micro Blaze to exchange the data with it.

It possesses two types of signals: Master's degree (Micro blaze-> SPM_IP) and Slave (SPM_IP-> Micro Blaze).

6.6.2. BRAM_PORT_B: connect SPM_IP to BRAM (SPM) to be able to read and write in the latter.

The internal structure of SPM_IP is described in Figure 6. It consists of four units: INSTRUCTION_DECODER, SPM_HANDLER, SPM_INTERFACE and CPU_INTERFACE.
This separation allows realizing every function in a separate component. Therefore, it facilitates the translation in code VHDL the various components Hardware in the synthesis tool.

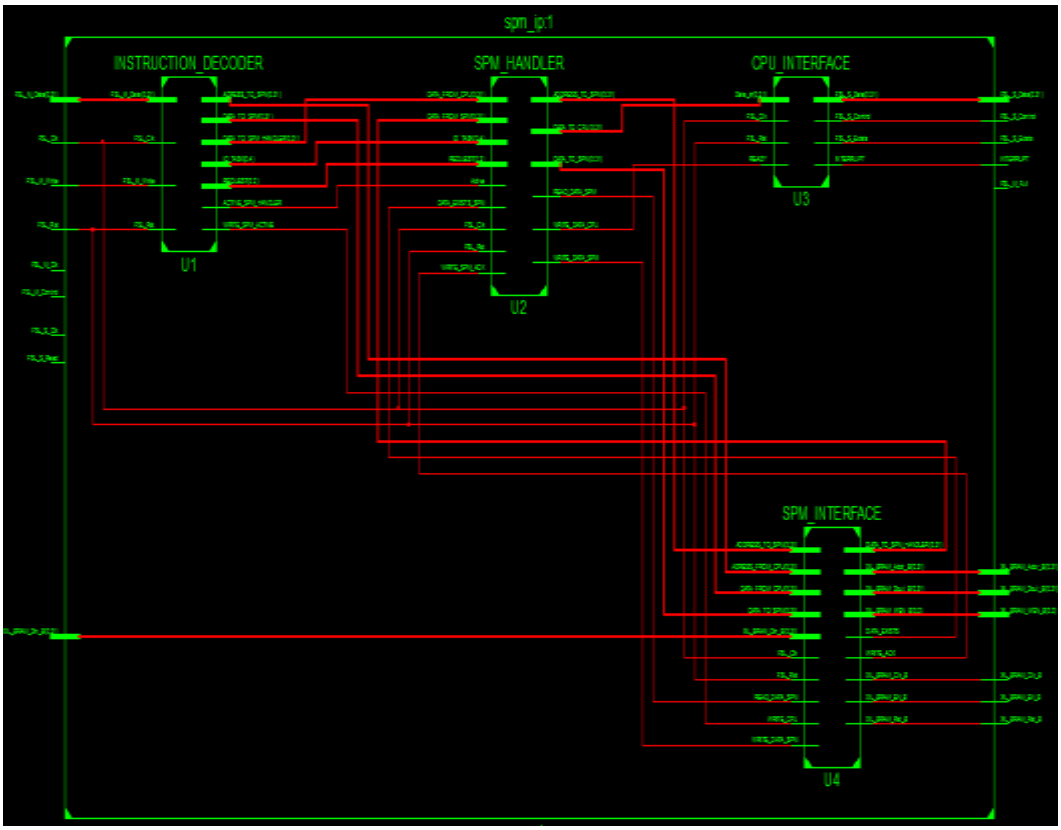


Figure 6. Internal structure of SPM_IP

7. SIMULATION

Direct writing in SPM to initialize it shown in Figure 7.

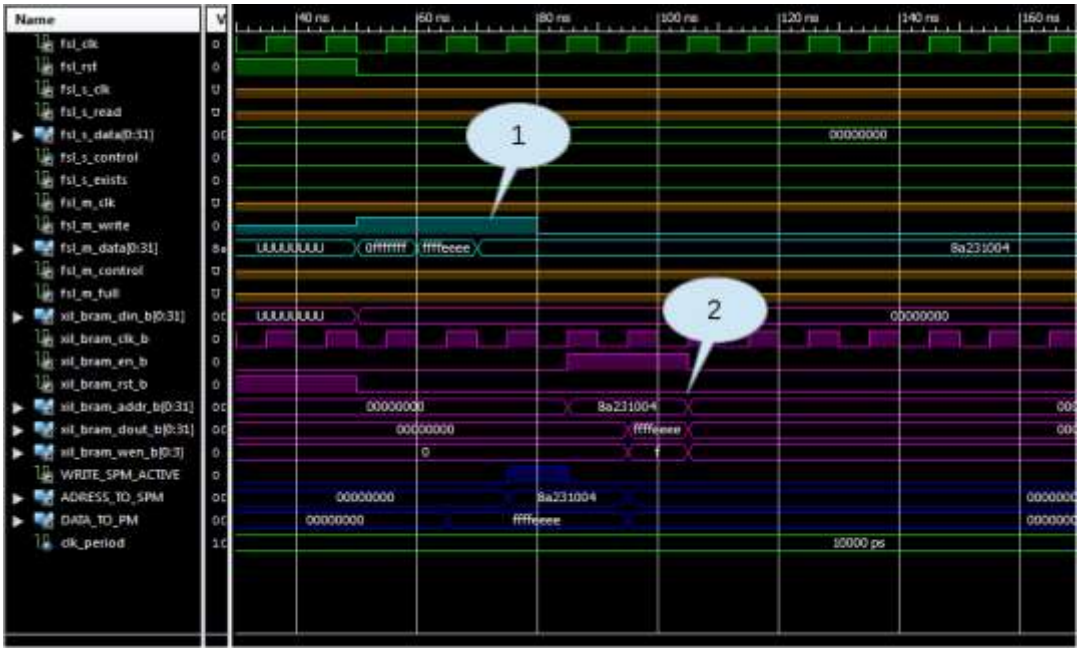


Figure 7. Direct writing in the SPM to initialize it

In “1”, coprocessor receives data from Micro blaze (FSL_M_Write='1'). First, it receives the instruction via FSL_M_Data 0x0FFFFFF (CMD=00), then the data to be written and finally the address 0x8A231004. In “2”, data is written in the SPM.

7.1. Read in TAS

In "1", coprocessor receives data from Micro Blaze (FSL_M_Write='1'). In "2", it decodes the instruction (CMD=11, TASK_ID=0001, REQUEST=001). In "3", SPM_HANDLER hurries up and begins to load registers 4, 5, 6, 7, 8.
PROFILER_POINTER: 0x8A300004-PROFILER_ADDRESS: 0x8A310004-HEAP_CURR_PTR_SPM: 0x8A2000F-STAK_CURR_PTR_SPM: 0x8A2005F-HEAP_CURR_PTR_DRAM**:** 0x900000FF
STAK_CURR_PTR_DRAM: 0x9000 10EF.

In "8", SPM_HANDLER receives a memory word of the SPM 0xCCCCDDDD that it sends it towards the CPU in "10" via FSL_S_Data and it positions FSL_S_Control, FSL_S_Exists and INTERRRUPT to the high state impedance. Read in TAS shown in Figure 8.

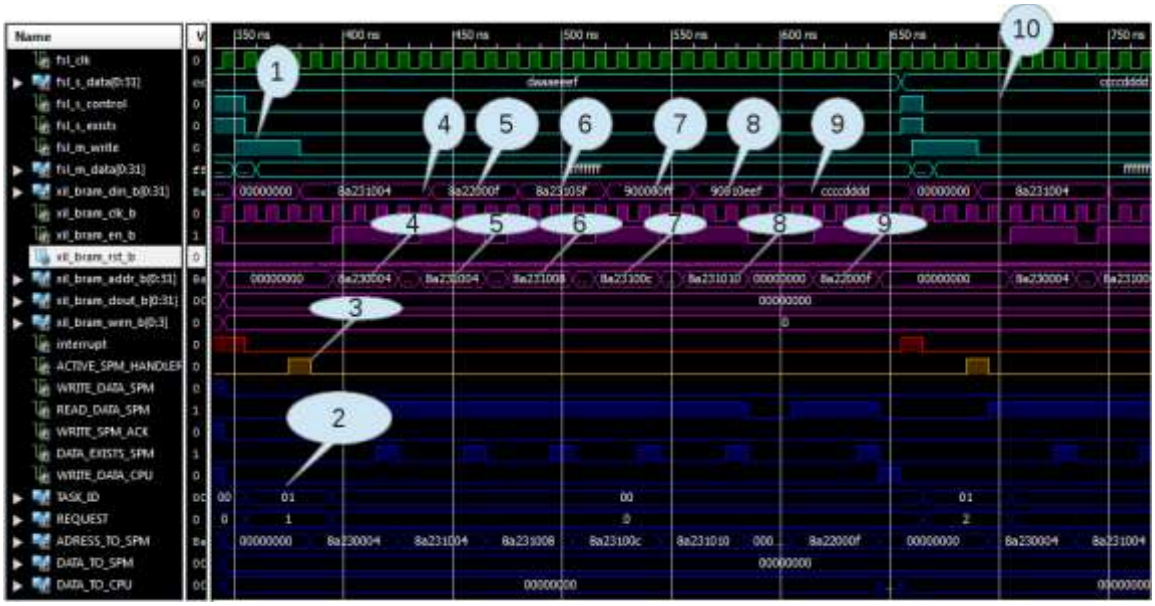


Figure 8. Read in TAS

7.2. Write in TAS

Shown in Figure 9 write in TAS. Always after the load of registers
PROFILER_POINTER: 0x8A300004-PROFILER_ADDRESS: 0x8A310004-HEAP_CURR_PTR_SPM: 0x8A20000F-STAK_CURR_PTR_SPM: 0x8A20005F-HEAP_CURR_PTR_DRAM: 0x900000FF
STAK_CURR_PTR_DRAM: 0x9000 10EF

In "1", Writing of the data 0xF2222222 at the address HEAP_CURR_PTR_SPM=0x8A2000F+4=0x8A200013. In "2" And "3", Updated HEAP_CURR_PTR_SPM 0x8A200013) and HEAP_CURR_PTR_DRAM (0x900000FF+4=0x90000103) in the SPM. In "4", sending a suite of A, 0xAAAAAAAA as request to inform the Micro Blaze that a DMA transfer (SPM towards DRAM) is necessary.

Figure 9. Write in TAS

7.3. Pull the stack

Shown in Figure 10 pull the stack. Always after the load of the registers PROFILER_POINTER: 0x8A300004-PROFILER_ADDRESS**:** 0x8A310004-HEAP_CURR_PTR_SPM**:** 0x8A20000F-STAK_CURR_PTR_SPM**:** 0x8A20005F-HEAP_CURR_PTR_DRAM**:** 0x900000FF STAK_CURR_PTR_DRAM**:** 0x9000 10E.

Figure 10. Pull the stack

In "1", reading data 0xDAAAEEEF from address STAK_CURR_PTR_SPM: 0x8A20005F. In "2", updating of STAK_CURR_PTR_SPM (0x8A20005F+4=0x8A200063) in the SPM. In "3", sending data to the Micro Blaze.

7.4. Push in the stack

Shown in Figure 11 Push in the stack. Always after the load of registers PROFILER_POINTER: 0x8A300004-PROFILER_ADDRESS**:** 0x8A310004-HEAP_CURR_PTR_SPM**:** 0x8A20000F

STAK_CURR_PTR_SPM**:** 0x8A20005F-HEAP_CURR_PTR_DRAM**:** 0x900000FF-STAK_CURR_PTR_DRAM: 0x9000 10EF

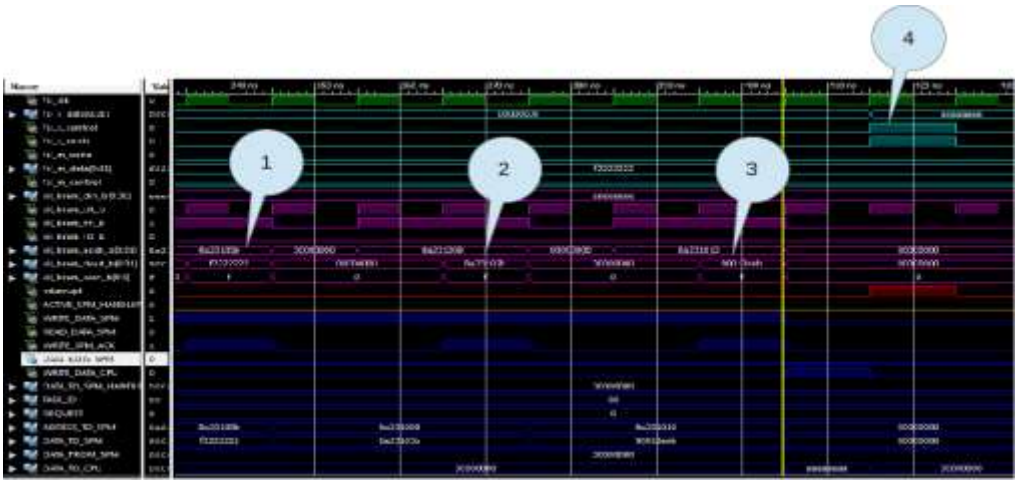


Figure 11. Push in the stack

0x8A20005B) and HEAP_CURR_PTR_DRAM (0x9000 10EF-4=0x900010EB) in the SPM.

In "4", sending a suite of A, 0xAAAAAAAA as request to inform the Micro Blaze that a DMA Transfer (SPM to DRAM) is necessary.

7.5. Case HEAP_CURR_PTR_SPM is not lower than STAK_CURR_PTR_SPM:

In this case we send a suite of 0xEEEEEEEE as request to inform the Micro Blaze that the profiler of the concerned task is full and thus a DMA transfer (SPM to DRAM) is necessary.

7.6. DMA Transfer

Shown in Figure 12 DMA Transfer.

Figure 12. DMA Transfer

7.7 Resources of FPGA used during the Synthesis:

We implemented our coprocessor on the Xilinx vertex 5 LX50T-1156 of DIGILENT; the following table shows resources of the FPGA used by our coprocessor. As Shown in Table 1:

Table 1. Resources of FPGA used by the coprocessor

Device Utilization Summary				[-]
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	591	28,800	2%	
Number used as Flip Flop	591			
Number of Slice LUTs	972	28,800	3%	
Number used as logic	968	28,800	3%	
Number using O6 output only	736			
Number using O5 output only	111			
Number using O5 and O6	121			
Number used as exclusive route-thru	4			
Number of route-thrus	115			
Number using O6 output only	115			
Number of occupied Slices	282	7,200	3%	
Number of LUT Flip Flop pairs used	1,012			
Number with an unused Flip Flop	421	1,012	41%	
Number with an unused LUT	40	1,012	3%	
Number of fully used LUT-FF pairs	551	1,012	54%	
Number of unique control sets	9			
Number of slice register sites lost to control set restrictions	5	28,800	1%	
Number of bonded IOBs	173	480	36%	
Number of BUFG/BUFGCTRLs	1	32	3%	
Number used as BUFGs	1			
Average Fanout of Non-Clock Nets	4.52			

8. CONCLUSION

The hardware circuit of Automaton totally finalized, in complement with software management of SPM, constitutes a platform of tests which promises considerable advantages in the choice of an effective management of data and of code to put into SPM for their exploitation in the Embedded Systems. This original realization hardware offers the opportunity to look easily and dynamically the storage space of the data and the code for an optimized solution in the communication of the data in core or multi-core for a better efficiency of the Multi-tasking. Afterward, we intend to set up a software method of exploitation of the locality of the data and of the code to feed SPM and proceed to the study of the performance (latency of access, rate of success, run time, energy consumption) of this new SPM endowed with the coprocessor which we designed.

REFERENCES

[1] Preeti Ranjan Panda, Nikil D. Dutt, Alexandru Nicolau, *"Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications".* Proceedings of the 1997 European Design and Test Conference (ED&TC'97), pp. 7-11, Paris, Mar. 1997 IEEE.

[2] Sheng-Wei Huang, Yung-Chang Chiu, Zhong-Ho Chen, Ce-Kuen Shieh, Alvin Wen-Yu Su, Tyng-Yeu Liang, *"A Region-based Allocation Approach for Page-based Scratch-Pad Memory in Embedded Systems".* International Conference on Computational Science and Engineering. DOI 10.1109/CSE.2009.350. 2009 IEEE.

[3] Wei Hu, Tianzhou Chen, Qingsong Shi, Feng Sha, "Efficient Utilization of Scratch-Pad Memory for Embedded Systems", 978-1-4244-3304-9/09, 2009 *IEEE*.

[4] R. Banakar et al., *"Scratchpad memory: A design alternative for cache on-chip memory in embedded systems"*, In Proceedings of 10th International Symposium on Hardware/Software Codesign (CODES), 2002, pp 73-78, ACM Press.

[5] Andhi Janapsatya, Aleksandar Ignjatović and Sri Parameswaran, "Exploiting Statistical Information for Implementation of Instruction Scratchpad Memory in Embedded System", *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, Volume 14, issue 8, August 2006, pp 816-829. DOI: 10.1109/TVLSI.2006.878470. ISSN: 1063-8210.

[6] F. Angiolini, L. Benini and A. Caprara, *"Polynomial-Time Algorithm for On-chip Scratchpad Memory Partitioning"*, International Conference on Compilers, Architecture and Synthesis for Embedded Systems. Proceedings of the 2003.

[7] F. Angiolini et al., *"A Post-Compiler Approach to Scratchpad Mapping of code"*, International Conference on Compilers, Architecture and Synthesis for Embedded Systems. Proceedings of the 2004.

[8] A. Janapsatya, A. Ignjatovic, and S. Parameswaran, *"Hardware/Software Managed Scratchpad Memory for Embedded System"*, International Conference on Computer Aided Design, Proceedings of 2004.

[9] Christoforos Kachris, Georgios CH. Sirakoulis; Dimitrios Soudris, "A MapReduce Scratchpad Memory for Multi-core Cloud Computing Applications", Microprocessors and Systems, Volume 39, issue 8, November 2015, pp599-608, Reprint submitted to Elsevier pp1-27, *Elsevier* 12, 2016.

[10] Tiago Rogério Mück and Antônio Augusto Fröhlich,"Run-Time Scratch-pad Memory for Embedded Systems", 978-1-61284-971-3/11, 2011 *IEEE*.
[11] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Meikang Qiu and Edwin H. M. Sha, *"Optimal Data Allocation for Scratch-Pad Memory on Embedded Multi-core Systems"*, 2011 International Conference on Parallel Processing. DOI 10.1109/ICPP.2011.79. IEEE Computer Society.
[12] Z. Hu, G. Gerfin, B. Dobry, and G. R. Gao, "Programming experience on Cyclops-64 multi-core chip architecture", in *STMC '06*, 2006.
[13] W. Che, A. Panda, and K. S. Chatha, "Compilation of stream programs for multi-core processors that incorporate scratchpad memories", in DATE '10, 2010, pp1118-1123.
[14] U. Angel Dominguez and R. Barua, "Heap Data Allocation to Scratch-Pad Memory in Embedded Systems", *J. Emdedded Comput*. 1(4):521-540, 2005.
[15] R. Mcllroy, P. Dickman, and J. Sventek, *"Efficient dynamic heap allocation of scratch-pad memory"*, In *ISMM'08*: Proceedings of the 7th international symposium on Memory management, pages 31-40, New York, NY, USA, 2008, ACM.
[16] Jun Zhang et al., *"Optimizing Data Allocation for loops on Embedded Systems with Scratch-Pad Memory"*, 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. DOI 10.1109/RTCSA.2012.20.
[17] B. Anuradha, Sandhya Nair LJ, Dr C Vivekanandan, "An Optimization Technique to Improve Power Consumption of Embedded System", *International Journal of Engineering and Innovative Technology (IJEIT)*, Vol. 2, issue 10, April 2013.
[18] Vishwesh Jatala, Jayvant Anatpur, Amey Karkare, "Scratchpad sharing GPUs", arXiv:1607.3238v5 [CS.AR], 12 feb 2017, pp1-34.
[19] Böhnert M., Scholl C. (2016) Task Variants with Different Scratchpad Memory Consumption in Multi-Task Environments. In: Hannig F., Cardoso J.M.P., Pionteck T., Fey D., Schröder-Preikschat W., Teich J. (eds) Architecture of Computing Systems-ARCS 2016. ARCS 2016. Lecture Notes in Computer Science, vol 9637. *Springer, Cham*.

BIOGRAPHIES OF AUTHORS

Chabane HEMDANI received his Engineering degree in Computer Science from University of Tizi Ouzou (Algeria) and his Master II degree in Computer Science from the same University in 1998. He teaches Computer Architecture at the University of Tizi Ouzou (Algeria) since 1999. He is currently preparing his PhD Thesis in Computer Science. His research interests include Embedded Systems, Compiler Design and Operating Systems.

RACHIDA AOUDJIT is an Assistant Professor at Computer Science Institute, form University of Tizi Ouzou. She is also a research member at the LARI Laboratory of the Computer Science Department. Her areas of interest include Mobile and Sensor networks, Embedded systems, Vehicular networks, Internet of Things.

Mustapha LALAM received his Engineering degree in Computer Architecture from the School of Computer Science (Center of Studies and of Research in Computer Science, Algiers, Algeria) in 1980 and PhD in 1990 in Computer Science from University of Toulouse III (France). He is one of Inventors of four Patents on Serial Multiport Memory. He is Professor in the Computer Science at the University of Tizi Ouzou (Algeria) since 2004. His research interests include Hardware Design, Embedded Systems, Smart Cities, Distributed Systems and Mobility management for Wireless Mobile Computing and Communications.

Khaled SLIMANI received his Master II degree in Computer Science from the University of Tizi Ouzou (Algeria) in 2013. He is currently preparing his PhD thesis in Computer Science at the University of Tizi Ouzou since 2014. His research interests include Computer Architecture, Hardware Design and Embedded Systems.