Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC

Ferid Gharsalli Samy Meftali Frédéric Rousseau Ahmed A. Jerraya

TIMA laboratory

46 av. Felix Viallet

38031Grenoble cedex (France)

Tel: (33) 4 76 57 46 41

{ferid.gharsalli, samy.meftali, frederic.rousseau, ahmed.jerraya}@imag.fr

ABSTRACT

Embedded memory plays a critical role to improve performances of systems-on-chip (SoC). In this paper, we present a new methodology for embedded memory design in the case of application specific multiprocessor system-on-chip. This approach facilitates the integration of standard memory components. The concept of memory wrapper allows automatic adaptation of physical memory interfaces to a communication network that may have a different number of access ports. We give also a generic architecture to produce this memory wrapper. This approach has successfully been applied on a low-level image processing application.

General Terms

Design, Experimentation.

Categories and Subject Descriptors

M1.9 and T4.5: High level and architectural synthesis

Keywords

Embedded Memory, System-on-Chip, Memory access, Memory Wrapper Generation.

1. INTRODUCTION

Embedded memory is becoming the new paradigm allowing entire systems to be built on a single chip. In the near future, new developments in process technology will allow the integration of more than 100 Mbits of DRAM and 200 millions gates of logic onto the same chip. Hence, several applications types (networking, computer graphics and multimedia applications) that require a large number of embedded memories (DRAM, flash, etc) can exploit this evolution. Unfortunately, nowadays, there is not yet a complete and automatic method allowing the designers to integrate all these memory types (particularly the global memory) into a multiprocessor heterogeneous systems-on-chip (SoC) from a high abstraction level. The memory integration process in such systems is still hand-made. This is time-consuming and error prone.

This evolution is creating several breaking points in the memory design process.

- The SoC specification consists of heterogeneous components in terms of communication protocols and abstraction levels. To integrate existing memory components in such systems, the physical memory ports should be adapted to the number of accesses required by the application.
- During the design space exploration, designers need to try different kind of memories (in terms of type, size, and consumption). This exploration should not modify the rest of the architecture. In this case, memory interface flexibility is needed.
- SoC architectures allow the integration of several master CPUs sharing a global memory. One problem is the parallel access to the memory. Multiple memory access may require sophisticated synchronization scheme.

This paper presents a systematic approach for memory integration into multiprocessor SoC. Section 2 explains the main problems of integrating memory into multiprocessor SoC. In section 3, we describe the basics of our methodology and our architectural models at different abstraction levels. In section 4, we present a generic wrapper architecture as well as its automatic generation. Section 5 highlights this methodology applied to an image processing application for a digital camera. Finally, section 6 concludes this paper.

2. MEMORY FOR MULTIPROCESSOR SYSTEM-ON-CHIP

To reduce the complexity of memory design, most of current system design methods adopt design reuse [1, 2, 3, 4, 5, 6]. In such design methods, system architecture specification includes heterogeneous components in terms of communication protocols and abstraction levels.

Figure 1 shows an example of a conventional multiprocessor SoC composed of CPU, DSP and/or memory components.

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

Compared with the design of conventional embedded systems, the implementation of system communication becomes much more complicated in multiprocessor SoC design, since (1) heterogeneous processors are involved in communication, (2) complex communication protocols and networks are used, and (3) standard memory components need to be connected to processors and/or network [7,8].

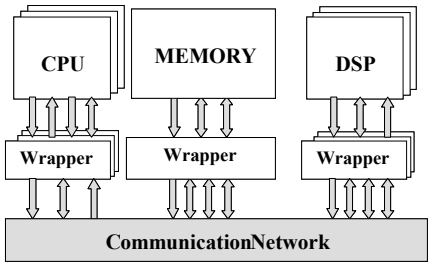


Figure 1: A typical Multiprocessor SoC

Existing memory components may have a fixed access structure (e.g. number of ports). To integrate this memory component into a heterogeneous system, we often need a different number of access ports. The key issue is then to adapt the physical memory interface (with fixed number of access ports and protocols) to the logical interface. Wrappers have been widely used to resolve this problem for simulation and synthesis [4, 9, 10, 11, 12]. For simulation, BFM (bus functional model) encapsulates a memory functional model with a cycle accurate interface [4, 9]. In [10], a wrapper for mixed-level cosimulation between FIFO channel and cycle-accurate models is presented. In [1, 3, 4], a protocol wraaper is used to adapt a communication protocol of IP to the communication protocol of on-chip bus. Most previous approaches using wrappers have the following limitations: (1) there is no generic architecture model for memory wrappers and (2) there is no automatic method for memory wrapper generation. Our contribution is (1) the introduction of a generic memory wrapper architecture that can be applied to several types of memory and (2) the definition of a method to generate automatically these memory wrappers.

3. MEMORY DESIGN FLOW

This work is an extension of the work presented in [1] and [5] to cover memories. The overall flow allows component based design for multicore SoC. The system is initially described as a set of virtual components interconnected via channels. Virtual component consists of a wrapper and internal module. In this environment an internal model may correspond to a software task or hardware module. The wrapper adapts the interface of the module to the communication network. This section introduces the concept of virtual memory module in the component based design flow.

3.1 Memory design based on separation between behavior and communication

Memory flexibility is ensured by the separation between the high-level memory access code and the physical memory access protocols. From the processor view, memory access must be completely independent of the memory type (synchronous, asynchronous, static, dynamic, etc). Accordingly, physical memory accesses are abstracted to logical accesses at the system level. A logical access may hide the memory protocol, several operations (address decoding, row and column selection, refresh management, etc) that depend on the type of memory and control signals. The operations on these ports are fixed data type assignments (e.g. real, integer). On the other side, physical memory access needs explicit signals and explicit protocols.

To adapt the logical interface to the physical interface, we use a virtual memory module (Figure 2). The latter consists of behavior and interface. The interface, which encapsulates the memory, is composed of two kinds of ports: internal ports and external ports. External ports correspond to logical ports that allow a logical access to the memory, whereas the former correspond to physical ports and allow a physical access to the memory. In the remainder of the paper, we will use the expressions “physical ports” and “logical ports”, and the correspondence between them is made by a wrapper.

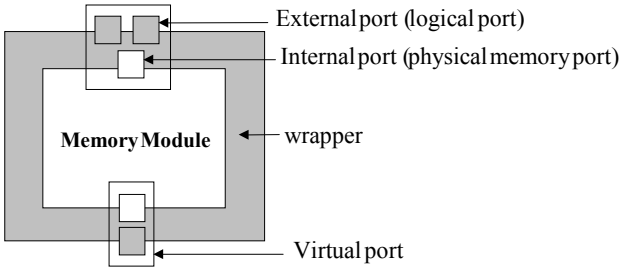


Figure 2: Logical and physical port concept

The wrapper introduces the required flexibility in the design. It allows the designer to decide quite late during the design process which component will be used for the implementation of the memory. Flexibility is achieved thanks to the separation between behavior and interface.

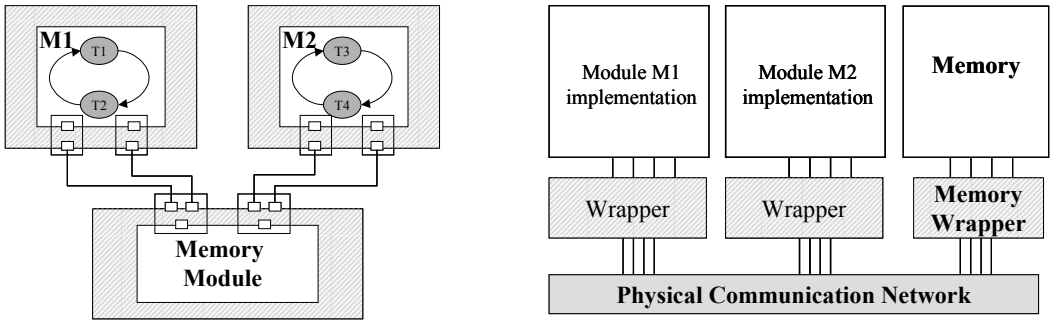
The memory module and its wrapper compose a virtual memory module. We can access to the virtual memory module through its virtual ports. The virtual port is hierarchical and has internal as well as external ports.

3.2 Architectural model at different abstraction levels for SoC design

In order to master the complexity, the architecture model may be described at different abstraction levels. We distinguish two main levels: virtual architecture level and micro-architecture level.

3.2.1 Virtual architectural model

The virtual architectural model (also called macro-architecture model) represents an abstract architecture. This abstract architecture is composed of a set of virtual modules interconnected through logical wires (Figure 3.a). Each module may represent a software processor (e.g. DSP or a micro-controller executing software), a hardware processor (specific hardware) or an existing memory module in the final architecture. The logical wires are abstract channels that transfer fixed data types (e.g. integer, real) and may hide low-level protocols (e.g. handshake or memory mapped I/O).



(a) Virtual architecture model (b) Micro-architecture model

Figure 3: Architectural models

Each virtual module consists of a module and its wrapper, and they communicate with each other through abstract channels connected to their virtual ports. For instance, FIFO communication is realised using high-level communication primitives. In our design flow, the virtual architecture is described using an extension of SystemC. More details about this model are given in [1].

The memory wrapper may include controllers and buffers when needed. At this level, the time unit becomes the clock cycle and the memory wrapper behavior corresponds to finite state machines (FSM), where each transition is realized in a clock cycle. To resolve the heterogeneity problem (in terms of communication protocols and abstraction levels), the memory wrapper has channel adapters that perform conversion protocol and abstraction level adaptation between logical ports and physical ports. The architecture of the memory wrapper is more detailed in the next section.

3.2.2 Micro-architecture model

The micro-architecture abstraction level gives the detailed RTL architecture (Figure 3.b). In this model, existing global memory modules are encapsulated within an interface (memory wrapper) in order to accommodate the final protocol and to isolate memory from the communication network. This wrapper adapts memory protocols to the communication network. The communication between the memory module and other modules of the architecture is made through the memory wrapper by using physical wires that implement the final protocols.

4. AUTOMATIC MEMORY WRAPPER GENERATION BY ASSEMBLING LIBRARY COMPONENTS

The key idea behind this work is to allow the automatic memory wrapper generation. In order to achieve this, we use a generic wrapper architecture that can be customized according to the architecture under design.

4.1 Generic memory wrapper architecture

Figure 4 shows a generic model of a memory wrapper that connects a global memory shared by several processors through the communication network. Its generic architecture is based on the wrapper architecture described in [13]. It is composed of two main parts. The first part, called Memory Port Adapter (MPA) is specific to the memory. The second part depends on the communication protocol used by the communication network and it contains several modules called channel adapters (CA). These two components communicate through an internal communication bus.

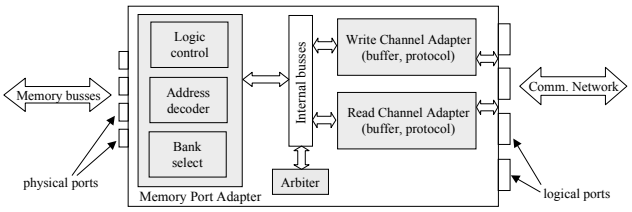


Figure 4: Generic memory wrapper architecture

- Memory port adapter: the memory dependent part includes a controller and several memory-specific functions such as control logic, address decoder, bank controller and other functions which depend on the type of memory (refresh management in DRAM). In addition, the MPA performs type conversion and data transfer between internal communication bus and memory bus. The complexity of the MPA behaviour depends on the memory used.

The communication network dependent part is made of three types of basic elements:

- Channel adapter (CA): CA implements the communication protocol. Its implementation depends on many parameters such as communication protocol (FIFO, burst, etc), channel size (int, short, etc), and port type (in, out, master, slave, etc). The channel adapter manages read and write operations requested by the other modules accessing the memory. The number of channels gives the number of CA.
- Internal communication bus interconnects the two parts of the memory wrapper (MPA part and the CA part). This internal bus is usually composed of address, data and control signals. The size of this internal bus depends on the memory bus size and the channel size. This size is determined for each instance of memory wrapper at the wrapper generation step.
- Arbiter: the arbiter manages the conflicts generated by multiple connection of CAs to the internal bus of memory wrapper. In the case of multi-access to a shared memory port, the arbiter must give the access permission to only one channel. In our model, this is managed using a priority access list.

4.2 Basic elements needed for wrapper generation

In order to generate memory wrapper implementation at the RT Level from the virtual architecture level and to simulate it, three basic libraries are used.

1. The library of physical memories contains several memory models (SRAM and SDRAM abstract memory models).
2. The memory port adapter library contains several specific MPA in relation with memories from the library of physical memories.
3. The communication protocols library contains several types of channel adapters (read and write channel adapter with handshake, FIFO or burst protocol).

4.3 Memory wrapper generation flow

We assume that the virtual architecture is fixed by the designer, or following a high level design flow [14, 15, 16]. The virtual architecture is annotated with parameters that will guide the wrapper generation step.

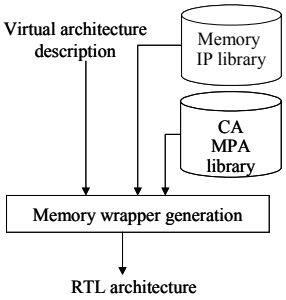


Figure 5: Memory wrapper generation flow

The memory wrapper generation step consists of choosing (1) the channel adapter and (2) the memory port adapter from the libraries (Figure 5). We use parameters to configure selected components. For instance, CA configuration uses the following allocation parameters: read/write type, master/slave operation, type of transmitted data, access mode. Parameters used for the MPA configuration correspond to the address/data bus size, the memory interface signals and static or dynamic. After configuration, customized components are instantiated. The CA component(s) is (are) connected to the MPA through internal busses. Finally, the entire wrapper is connected to the rest of the system. Every time we add a new component in the library of memory IP, we have to write all the specific parts related to this new memory, such as MPA and some other functionalities (refresh, bank selection, …).

5. LOW LEVEL IMAGE PROCESSING FOR DIGITAL CAMERA

5.1 Principle and System Specification

Our application is about a JamCam digital camera. We use the version detailed in [17]. This system architecture is memory rich and processor-poor. It contains 2 Mbytes Flash image storage, 2 Mbytes ROM and 256 Kbytes static RAM. It uses an 8-bit RISC processor with embedded USB and other interface functionalities.

The input image is stored in the memory system before the image-processing step done by the processor. We are interested in the filtering image part that consists of blurring the image by applying a Gaussian convolution mask. It has optimal properties in a particular sense: it removes some aberrations introduced by the sensor, including noise.

To accelerate the image processing, we use two ARM7 processors sharing a memory module. Each processor communicates with three communication channels: address and data channels, and a buffer channel for read operations. In the rest of this section, we study the architecture generation of the image processing. To show the flexibility of the approach, we tried two memory types (a single port SDRAM and a double port SRAM) in two different experiments. The goal is to show that we can change a memory module without changing the rest of the architecture.

5.2 Virtual architecture model

The virtual architecture of the application includes three virtual modules named VM1, VM2 and Virtual memory module. The basic module behaviors are described as SystemC processes (thread). Figure 6 shows the specification for the two experiments. The only difference between these models is the content of the memory module. The specifications of the two other modules are the same.

5.2.1 Experiment 1: using a double port memory

In Figure 6.a, the memory module corresponds to a SRAM memory that has **two physical ports**. Both module VM1 and VM2 access the virtual memory module simultaneously. Each module (VM1 and VM2) is connected to the virtual memory module via one abstract channel.

The virtual memory module is composed of the SRAM module and its wrapper. It contains **two virtual ports**. One is connected to VM1 port, the other is connected to the VM2 port. Each virtual port is composed of the SRAM physical port (internal ports) and one logical port (external port). Operations on these ports are fixed data type assignments (integer 32 bits).

5.2.2 Experiment 2: using a single port memory

The memory module (Figure 6.b) corresponds to a single port SDRAM memory that has **only one physical port**. The virtual port is composed of one physical port connected to the SDRAM and two logical ports (external ports) connected to the virtual ports of VM1 and VM2 via two abstract channels. VM1 and VM2 may access the logical ports in the same time, but only one access to the physical SDRAM port is accepted. So, the SDRAM wrapper performs the arbitration of the multi-access to its single port.

5.3 Micro-architecture model

The micro-architectures corresponding to Figure 6.a and Figure 6.b are shown in Figure 7.a and Figure 7.b respectively. To obtain the RTL architecture, the wrapper is generated. The wrapper establish the RTL communication between the two ARM7 processors and the memory. The physical memory access needs several explicit signals (RAS, CAS, BS, etc), numerous operations and protocols (data conversion, address decoding, refresh, burst protocol, etc).

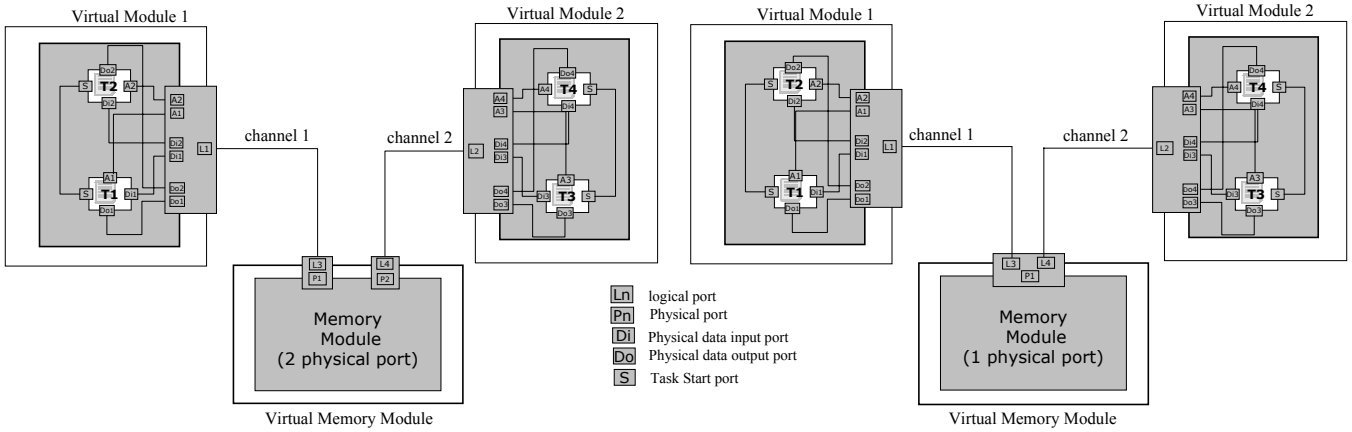
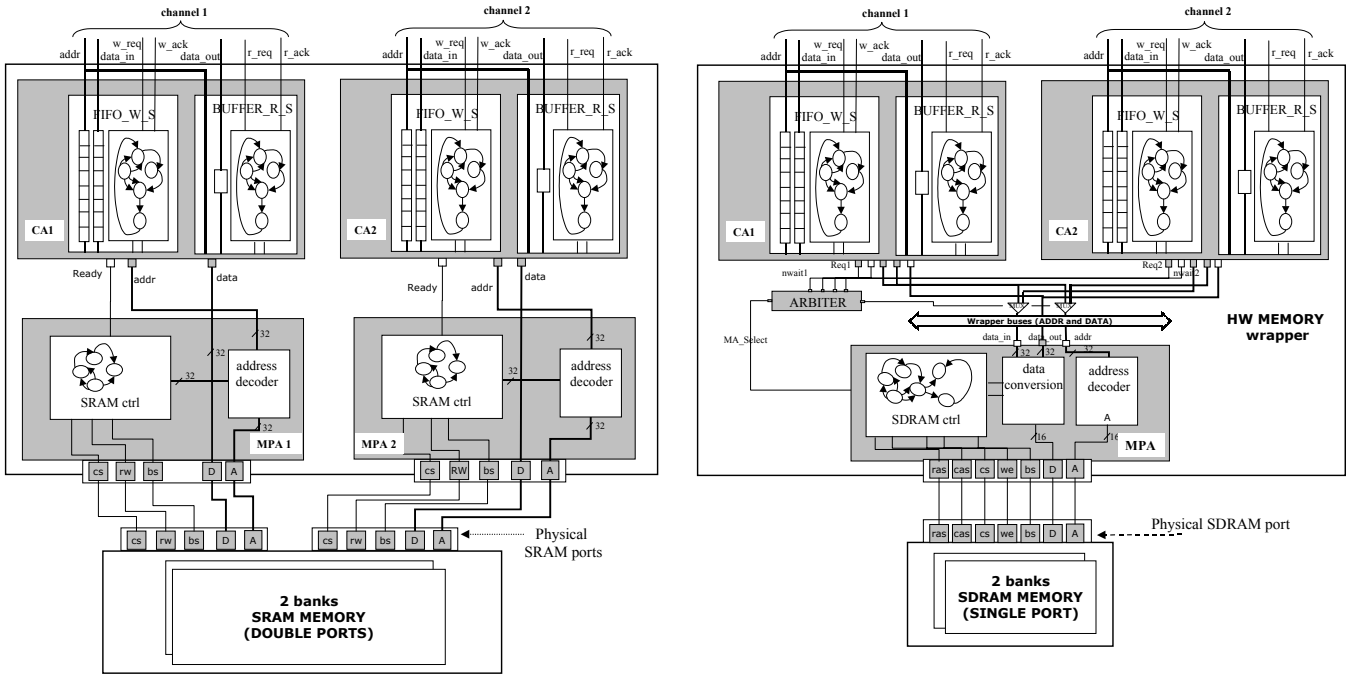


Figure 6: The virtual architecture of the image filtering application using (a) double port memory (b) single port memory



(a) double port memory

(b) single port memory

Figure 7: RTL generated architecture of the filtering application

Communication protocols are refined:

- For write operations, we use two FIFOs (for the address and data) that allows processors to send several write requests to the memory. The size of each FIFO is 32 words of 32 bits wide. Both FIFOs are controlled by only one FSM.
- For read operations, the processor gives the data address, and we use a buffer (one word of 32 bits wide) to send back the read data.

5.3.1 Experiment 1: using a double memory port

In this experiment, we use a dual port SRAM with two banks (Figure 7.a). The size of the memory buses is 32 bits.

The generated memory wrapper ensures the interconnection between the SRAM and the ARM7 wrappers. To adapt the memory protocol to the point-to-point network protocol, we instantiated two channel adapters. To adapt the two SRAM physical ports to the logical ports of the system, we instantiated two MPA specific to the SRAM. MPAs are composed of memory controller which generates control signals and address decoder.

5.3.2 Experiment 2: using a single memory port

In the Figure 7.b, we use a single port SDRAM. The size of the SDRAM busses are 16 bits (data and address). To interconnect this memory to the network, we are not obliged to modify all the wrapper. Indeed, we instantiate only one MPA specific to the SDRAM. MPAs are composed of 3 parts: the memory controller which generates control signals, address decoder and data converter. The data converter performs address and data size conversion between the wrapper internal bus (32 bits) and the

memory bus (16 bits). The CAs used for the first architecture (Figure 7.a) and all the other part of the wrapper remain intact. As multi-access is not accepted, we implement an arbiter to control the memory multi-access. We use a HSK protocol to synchronize the memory accesses.

5.4 Results

The automatic generation of these wrappers allows a fast design space exploration of various types of memories. We generate SystemC model (for cosimulation) and we are currently working on the VHDL generation for synthesis. Memory models are also written in SystemC and VHDL.

The memory architecture and the generated wrapper have been validated with a cycle accurate co-simulation approach based on SystemC. Two ISSs of ARM7 core (40 MHz) are used.

We note that there is a small difference in the code size of the memory wrapper in the two RTL architecture models. In fact, CAs are not changed. Only the MPA is changed (10% of the wrapper code), all the rest remains intact. As the SDRAM requires complex control signals, the two controllers implemented in MPAs are more complex than the one implemented in the SRAM wrapper and it explains the small difference in the code size. This easy integration of memory proves the flexibility of the architecture.

6. CONCLUSION

This paper described the advantages of using memory wrapper to connect a shared memory to processors or IP in a system-on-chip. This memory wrapper separates the logical access (by processor or IP) to the physical access to the memory, which depends on the memory type. It allows designers to try easily several kinds of memory without modifying the rest of the system. We have given also a generic architecture of the memory wrapper in order to generate it automatically. We proved the applicability of this method by applying it on an image processing system.

REFERENCES

[1] W. Cesário, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A.A. Jerraya, M. Diaz-Nava "Component-Based Design Approach for Multicore SoCs", DAC 2002.

[2] J.A Rowson and A. Sangiovanni-Vincentelli, “Interface-Based Design”, Proc. of DAC, pp. 178-183, 1997.

[3] C.K Lennard, “Standards for System-Level design: Practical Reality or Solution in Search of a question?”, Proc. DATE, Mar. 2000.

[4] D.D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S.zhao, “SpecC: Specification Langage and Methodology, Kluwer Academic Publisher., 2000.

[5] Synopsys, Inc., http://www.systemc.org/.

[6] Coware, Inc., http://www.coware.com/cowareN2C.html.

[7] D.A Culler, J. P Singh, “Parallel Computer Architecture”, Morgan Kaufmann Publishers, 1999.

[8] D.A. Patterson, J.L Hennessey, “Computer Organization and Design-The Hardware/Software Interface”, Morgan Kaufmann Publishers.

[9] L. Sémira and A. Ghosh, “Methodologie for Hardware/Software Co-verification in C/C++”, Proc. ASP DAC, Jan. 2000.

[10] K. Takemura, M. Mizuno, and A. Motohara, “An approach to System-Level Bus Architecture validation and its Application to digital Still Camera Design”, Workshop SASIMI, pp. 195-201, Apr. 2000.

[11] J-Y. Brunel, W.M. Kruijtzer, H.J.H.N. Kenter, F.Petrot, and L. Pasquier, “COSY Communication IP’s”, Proc. DAC, pp. 406-409, June 2000.

[12] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, and A. A. Jerraya, " A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design ", CODES/CASHE 2001.

[13] D. Lyonnard, S. Yoo, A. Baghdadi, A.A. Jerraya, “Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip”, Proc. of DAC 2001, June 2001.

[14] F. Catthoor & al, Custom Memory Management Methodology, Kluwer Academic Publishers, 1998.

[15] P. R. Panda, N. Dutt, A. Nicolau, “Memory Issues in Embedded Systems-on-chip, Optimization and exploration”, Kluwer Academic Publishers, 1999.

[16] S. Meftali, F. Gharsalli, F. Rousseau, A.A. Jerraya, "An Optimal Memory Allocation for Application-Specific Multiprocessor SoC", proc. of ISSS 2001.

[17] http://www.pixelphysics.com