Unifying Memory and Processor Wrapper Architecture in Multiprocessor SoC Design

Férid Gharsalli Damien Lyonnard Samy Meftali Frédéric Rousseau Ahmed A. Jerraya
Laboratoire TIMA,
46 av. Félix Viallet
38031 Grenoble cedex (France)
{ferid.gharsalli, damien.lyonnard, samy.meftali, frederic.rousseau, ahmed.jerraya}@imag.fr

ABSTRACT

In this paper, we present a new methodology for application specific multiprocessor system-on-chip design. This approach facilitates the integration of existing components with the concept of wrapper. Wrappers allow automatic adaptation of physical interfaces to a communication network. We also give a generic architecture to produce these wrappers, either for processors or for other specific components such as memory IP. This approach has successfully been applied on a low-level image processing application.

General Terms

Design, Experimentation.

Categories and Subject Descriptors

C.0 [**General**]: *system architecture, Hardware/software interfaces*

Keywords

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

1. Introduction

Multiprocessor SoCs (MP SoC) are more and more used to meet the ever increasing performance requirements of application domains such as xDSL, game applications, etc... As these systems require heterogeneous processors (for application specific optimizations), complex communication protocols, and IP or application-specific memory components, this architecture generation demands significant design efforts.

One way to reduce this effort and to comply with the shorter time-to-market is to reuse some components. This means that we need to adapt specific physical accesses and protocols of those components to the communication network that may have other physical connections and other protocols. The protocol and physical adaptation are made with an interface called wrapper in the literature. The integration of several existing components makes specification and implementation of the wrapper a major design problem.

To facilitate the design space exploration and to allow the designer to try different components or communication protocols, we need to generate automatically these wrappers, based on parameters given by the architecture (processor types, protocols, etc.) and by the designer. The wrapper generation is actually done by assembling components caught in a library. We defined a generic architecture, which is suitable for most components. We implemented unified library components that can be used for processor or memory wrapper generation.

In our approach, we defined the wrapper at two abstraction levels. At the first level, we defined an abstract wrapper architecture that hides the low level communication. The second level corresponds to the RT level where the abstract wrapper is implemented.

This paper presents a systematic approach for existing component integration into multiprocessor SoC. Section 2 presents a multiprocessor SoC architecture and related work. In section 3, we describe the basics of our methodology and our architectural models at different abstraction levels. In section 4, we present 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. MP SoC Architecture and related work

Figure 1 shows an example of a conventional multiprocessor SoC composed of CPU, DSP and/or memory or IP components. 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 or IP components need to be connected to processors and/or networks [6][7]. To reduce the complexity of a design, most of current system design methods adopt design reuse [2][3][4][5]. In such design methods, system architecture specification consists of heterogeneous modules in terms of communication protocols and abstraction levels interconnected through a set of abstract ports called also logical ports.

The key issue is then to adapt the physical interface of these components to the logical interface. Wrappers have been widely used to solve this problem for simulation and/or synthesis

[4][8][9][10][13]. In the simulation step, a BFM (bus functional model) encapsulates a memory functional model with a cycle accurate interface [4][8]. In [9], a wrapper for mixed-level cosimulation between FIFO channel and cycle-accurate models is presented. In system synthesis, a protocol transducer is used to adapt a communication protocol of IP to the communication protocol of on-chip bus [1][3][4].

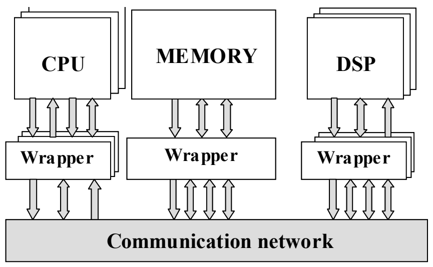


Figure 1: a typical Multiprocessor SoC

In [1], a communication wrapper is automatically generated from a VHDL package that implements the communication protocols (rendez-vous based on CSP model). [15] addresses the problem of external memory interfacing between a datapath and a memory. The internal architecture of this interface is composed of two main parts, one is memory dependent, and the other one is datapath dependent. This interface is optimized for aggressive scheduling of memory operations. Palmchip [17] provides a shared memory processor that corresponds to high performance configurable memory controller subsystem for a DDR SDRAM. This controller can be configured with DMA buffers. The DMA buffer matches transfer speeds of the on-chip bus with the controller by means of a FIFO. In this approach, an architecture template of this memory controller can be configured only on the AMBA bus, CoreFrame and Mbus, but it can't be used for different specific bus structure. This wrapper template architecture does not support large bus architecture and several cores and IPs.

Our contribution is (1) the introduction of a generic wrapper architecture that can be applied to several types of components and (2) the definition of a method to generate automatically these wrappers.

3. Models used in our SoC Design Flow

This work is an extension of the work presented in [12] and [14] to cover wrapper generation. In [12], an automatic generation of processor wrapper was detailed. [14] describes the automatic generation of memory wrapper in order to interconnect different memory IPs to a communication network. The wrapper generation in these works is based on two different libraries. In order to master the complexity of design, we do need to define a common generic architecture for all wrappers and then to build a common library.

3.1 Architectural model at different abstraction levels for SoC design

3.1.1 Virtual architectural model

The virtual architectural model represents an abstract architecture. This abstract architecture is composed of a set of virtual modules interconnected through logical wires (Figure 2.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 component 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).

Each virtual module communicates with the others through abstract channels connected to its virtual ports. For instance, FIFO communication is realized using high-level communication primitives. In our design flow, the virtual architecture is described using an extension of SystemC.

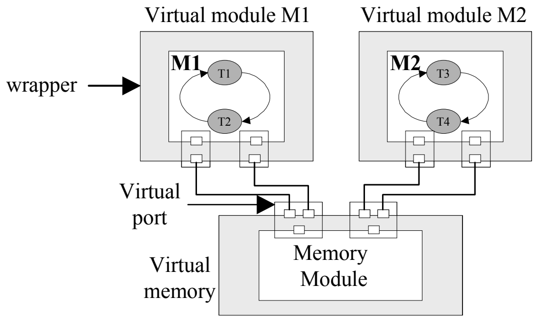
The wrapper performs conversion protocol and abstraction level adaptation between logical ports and physical ports.

3.1.2 Micro-architecture model

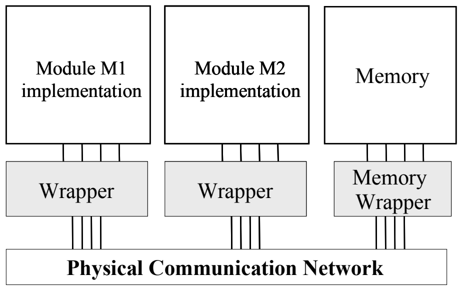
The micro-architecture abstraction level gives the detailed RTL architecture (Figure 2.b). In this model, existing components are encapsulated within an interface (wrapper) in order to accommodate the final protocol and to isolate the behavior from the communication network. This wrapper adapts protocols to the communication network. The communication between modules is made through wrappers by using physical wires that implement the final protocols. At this level, the time unit becomes the clock cycle and the wrapper behavior corresponds to a behavioral finite state machine (FSM), where each transition is realized in a clock cycle.

4. Unified Wrapper Model

The key idea behind this work is to allow the automatic wrapper generation based on a common library. In order to achieve this, we use a generic wrapper architecture that can be customized according to the architecture under design and can be used for both processors and memories.



(a) Virtual architecture model



(b) Micro-architecture model

Figure 2: architectural models

4.1 Unified wrapper model

The generic wrapper architecture is made of two main parts as shown in Figure 3.a: the module specific part called Module Adapter (MA) and the network specific part called Channel Adapter (CA). The two parts are interconnected through an internal bus (IB).

The Module Adapter implements services requested by the module. Generating the Module Adapter consists of assembling all the basic components that provide services.

The Channel Adapter implements the communication protocol (FIFO, DMA controller, etc) and controls the communication between the module and the network. Its implementation depends on many parameters such as communication protocol, channel size, and port type (in, out, master, slave, etc). The CA manages the data transfer between the communication network and the internal bus of the wrapper. The number of CA is given by the number of channels.

The internal communication bus (IB) interconnects the two parts of the memory wrapper (MA 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 bus size of the module and the channel size and it is determined for each instance of wrapper at the wrapper generation phase.

4.2 Application to processor

Processor wrapper (Figure 3.b) is obtained by the customization of the generic wrapper model described in the previous section. We customize the module adapter part which is specific to the processor. In this case, the MA is called Processor Adaptor (PA).

The PA performs channel access selection by address decoding and interrupt management. The PA is a master, whereas the CAs are slaves. The transfer of data between PA and CA is done through an internal bus by using a synchronization protocol. To do that, it has four kinds of signals: address, data, enable and interrupt. Data signal is bi-directional and has a generic type whereas the address signal is uni-directional. A specific address/data type (e.g. int, short, logic vector, etc) of address/data signals is determined for each instance of the generic wrapper architecture. Enable signals are set/reset by the processor adapter. They select one channel adapter and enable it to read/write data to/from data signal. Each channel adapter sets its interrupt signal when it receives data on its port.

The processor wrapper frees processors from executing communication code and separates computation from communication. A good example of processor wrapper has been presented in [12], an ARM7 and an M68k wrapper was implemented for IS95 and packet routing switch applications.

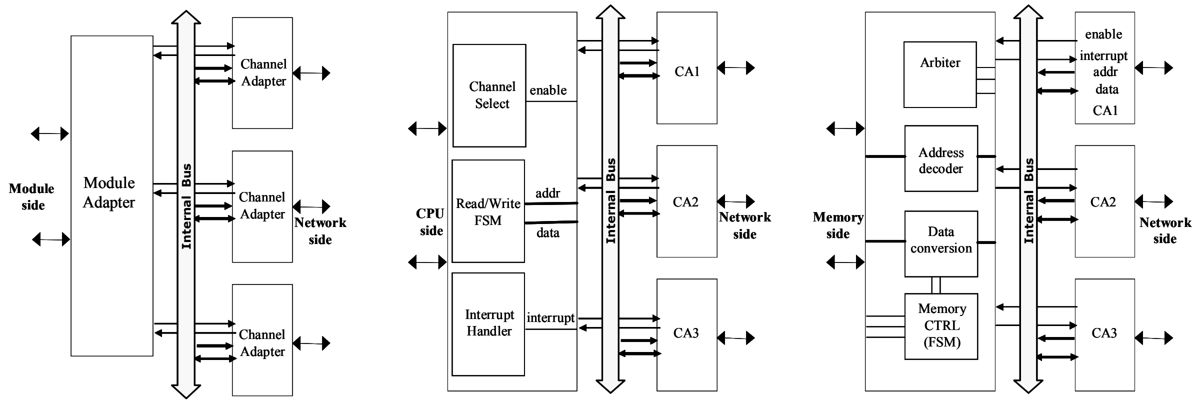
4.3 Application to memory

The generic wrapper architecture can also be used as a memory wrapper in order to adapt the physical memory interface to the communication network (Figure 3.c). For memory module, we customize the module adapter which is specific to the memory. In this case the module adapter is called Memory Port Adapter (MPA).

The MPA includes a memory 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 data type conversion and data transfer between internal communication bus and memory bus. In case of multiple connections of CA to the internal bus, MPA includes an arbiter that manages parallel access. The arbiter must give the access permission to only one channel adapter at a time. In our model, this is managed by using a fixed priority technique. The complexity of the MPA behavior depends on the memory used.

4.4 The unified libraries needed for wrapper generation

Generic wrapper architecture has been defined in the previous section. In order to facilitate the wrapper generation a library of basic components should be built. This library includes several macro-models of channel adapters and module adapters. There are three types of CA depending on the direction of the channel (in-out, out, in). The in-out CA is used during port read operations, the out CA is used during port write port operations and the in CA can be used for module configuration. The generic architecture allows us to use a common library to generate the two types of wrappers.



(a) generic wrapper architecture (b) processor wrapper architecture (c) memory wrapper architecture

Figure 3: wrapper architecture

CA can be slave or master. In the case of processor wrapper, the CA is used as a slave controlled by the processor adapter where it is used as a master that controls the memory port adapter in the case of memory wrappers.

The second element of this library is a generic MA part of the wrapper. Depending on the kind of the module, we generate either the specific processor adapter or the specific memory port adapter.

In the case of memory port adapters, several memory-specific services are implemented, which correspond to physical memory modules. We have written generic models of MPA associated with several memories (SDRAM Micron 256 Mbits, SRAM, etc). MPA services such as burst access, type conversion, refresh, address decoder are written according to the datasheets provided by manufacturers.

In the case of processor, we have implemented an ARM7 and M68K adapters. Our library is limited to two processor adaptors for availability reasons, but the generic module adapter can support other kinds of processors.

We also need a second library implementing the components of the architecture (processor core, memory). This library is composed of processors or memory components and their protocols. The processor part contains simulation models and synthesizable codes of available processors and their local architectures (i.e. processor core, local memory). The memory part of this library is made of generic memory codes used for simulation and synthesis written in SystemC and VHDL. The protocol part contains simulation models and synthesizable codes of available communication protocols.

4.5 Wrapper generation

As shown in Figure 4, the inputs of the wrapper generation flow are composed of processor and memory library, and MA and CA library. This generation step needs also parameters coming from a virtual architecture described in a SystemC like specification. For instance, CA configuration uses the following allocation parameters: read/write, master/slave, type of transmitted data, etc.

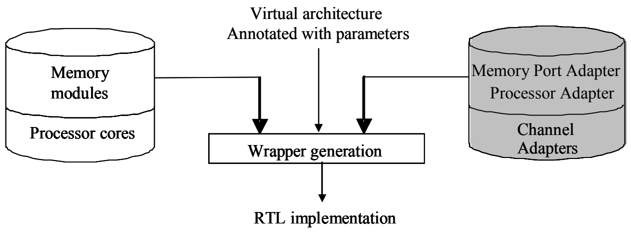


Figure 4: wrapper generation flow

Parameters used for the MPA configuration correspond to the number of ports, the memory bus size, the memory word width, the memory interface signals, and the access mode. The wrapper generation algorithm consists of customizing the generic CA and MA selected from the library by using these architectural parameters extracted from the virtual architecture. After this, the customized wrapper components are instantiated and the entire wrapper is connected to the rest of the system. Finally, the output of this flow is an RTL implementation.

5. Memory Wrapper Generation in Image Processing Application

In order to show the effectiveness of our approach and to validate the correctness of the memory wrapper, we performed a low level image processing for a digital camera application [16]. We implemented this algorithm using two processors (ARM7) and a global shared memory.
We performed two experiments in order to prove the memory flexibility ensured by the wrapper. In the first experiment, we used a dual port SRAM and in the second we used a single port SDRAM. In both cases, the memory wrapper is automatically generated. In these experiments, we used several wrappers for processors and memory, but we detail only memory wrappers.

5.1 Experiment 1: using a double memory port

At the virtual architecture level, the virtual memory module is composed of the SRAM module and its wrapper (Figure 5.a). It contains **two virtual ports**, each one being connected to an ARM7 processor. The architectural parameters extracted from this virtual architecture, which will be used for the memory wrapper generation are: 2 channels composed of 2 FIFOs (32 words x 32bits) and 2 buffers (1 word x 32 bits), the data type is integer, the memory access type is a burst mode of 4 words and the memory type is a dual port SRAM of 32 bits. The memory bus size and the IB size are 32 bits.

At the micro-architecture level, we use these parameters to customize the wrapper components. In our case (Figure 6.a), we instantiate:

- two CAs, each one is composed of two FIFOs (32 words x 32 bits) with one controller and one buffer (1 word of 32 bits),
- two specific SRAM port adapters. Each one is composed of one address decoder and one SRAM controller that provides the following services: SRAM control, burst access and test operation used during co-simulation,
- two parallel internal buses of 32 bits.

5.2 Experiment 2: using a single memory port

In the second experiment, as shown in Figure 5.b, we change the dual port SRAM by a single port SDRAM of 16 bits width and we use a classic Read-Write access.
At the micro-architecture level (Figure 6.b), we modify the last architecture of the memory wrapper by using

- a specific SDRAM port adapter that provides a dynamic refresh operation, a classic read/write access, address decoder, data type conversion (32 to 16 logic vectors) and an arbiter that manages multiple accesses.
- a shared internal bus of 32 bits.

5.3 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 soon for synthesis). Memory models are also written in SystemC.

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

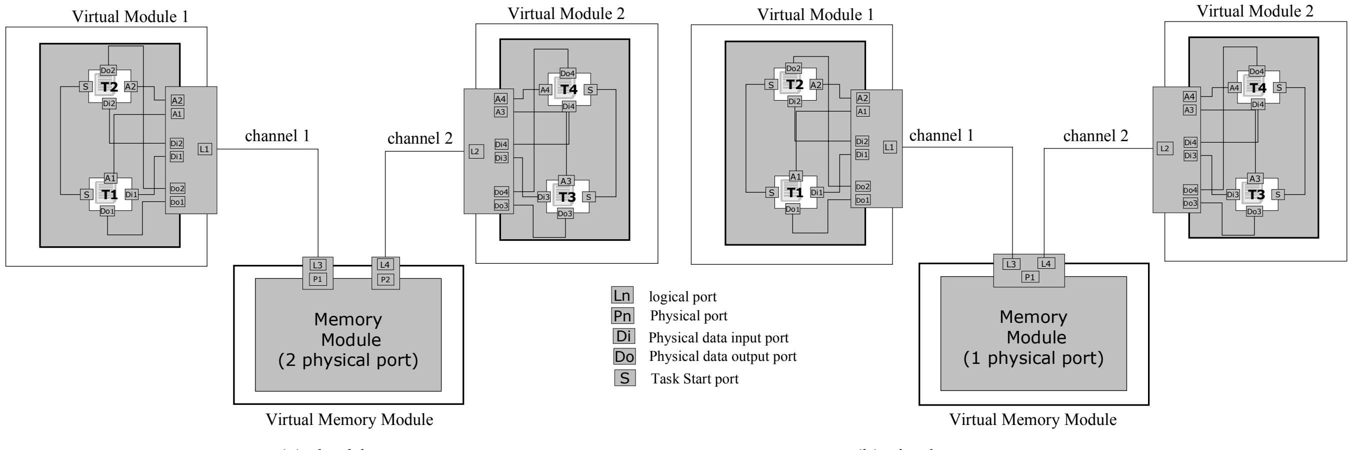

(a) double port memory (b) single port memory

Figure 5: The virtual architecture of the image filtering application

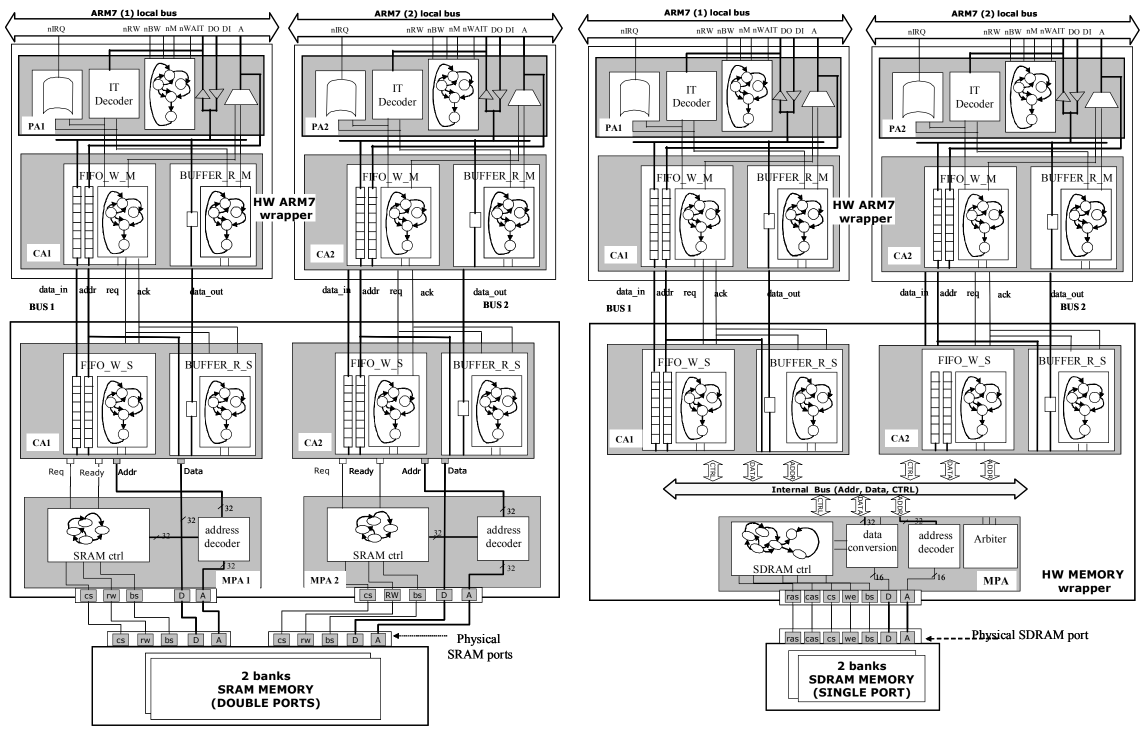

(a) double port memory (b) single port memory

Figure 6: RTL generated architecture of the filtering application

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). As the SDRAM requires complex control signals, the two controllers (SDRAM controller and data conversion controller) implemented into MPAs are more complex than the one implemented in the SRAM wrapper and it explains the small difference in the code size.

The write latency is 3 CPU (without memory latency) cycles whereas the read latency is 7 CPU cycles (send/receive).

The simulation cycle which corresponds to the processing of an image of 387x322 pixels is 2.05×10^6 CPU cycles in experiment one and 2.97×10^6 CPU cycles in experiment 2. Thus, with the assumption that code ratio leads to the area ration, we conclude that the first memory wrapper architecture is more optimal than the second one. This is due

(1) to the first wrapper architecture that supports the parallel accesses to the memory through two parallel internal buses and (2) to the burst mode used in the first experiment.

For both experiments, the area cost of the synthesized HW memory wrapper (AMS 0.35 µm CMOS) takes less than 5% of the overhead in total system area.

6. Conclusion

This paper describes the need of wrappers in multiprocessor SoC design and the requirement of unified libraries for the wrapper generation step. The automatic generation of wrapper is made by assembling basic components from a library. A generic architecture of wrapper is provided, either for processor wrapper or for memory wrapper. It could be extended to IP wrapper.

REFERENCES

[1] S. Vercautern, B.lin, and H. De Man, “Constructing Application-Specific Heterogenous Embedded Architectures from Custom HW/SW Applications”, DAC 1996.

[2] J.A Rowson and A. Sangiovanni-Vincentelli, “Interface-Based Design”, DAC 1997.

[3] C.K Lennard, P. Schaumont, G. De Jong, A. Haverienen, and P. Hardee, “Standards for System-Level design: Practical Reality or Solution in Search of a question?”, DATE 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] D.A Culler, J. P Singh, “Parallel Computer Architecture”, Morgan Kaufmann Publishers, 1999.

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

[8] L. Sémira and A. Ghosh, “Methodology for Hardware/Software Co-verification in C/C++”, Proc. Asia South Pacific DAC, Jan. 2000.

[9] K. Takemura and all, “An approach to System-Level Bus Architecture validation and its Application to digital Still Camera Design”, Workshop SASIMI 2000.

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

[11] P. Gerin, S. Yoo, G. Nicolescu, A. A. Jerraya, Scalable and Flexible Cosimulation of SoC Designs with Heterogeneous Multiprocessor Target Architectures, Proc. of Asia South Pacific Design Automation Conference, Jan. 2001.

[12] 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.

[13] 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.

[14] F. Gharsalli, S. Meftali, F. Rousseau and A.A. Jerraya, "Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC", DAC 2002.

[15] J. Park, P. C. Diniz, "Synthesis of Pipelined Memory Access Controllers for Streamed Data Applications on FPGA-based Computing Engines", ISSS 2001.

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

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