Interface Architecture Generation for IP Integration in SoC Design

Fatma Abbes[(1),(2)], Mohamed Abid[(1)] and Emmanuel Casseau[(2)]
[(1)]CES Laboratory, ENIS engineering school, Sfax, Tunisia
[(2)] Lester Laboratory, Research Center, UBS University, Lorient, France

***Abstract*-Designing component-based SoC (System On Chip) has become a communication design problem. The reuse of Intellectual Property (IP) cores in Multiprocessor SoC is facilitated by the concept of packaging and wrapping. In this paper, we present an approach to automate the integration process of hardware accelerators/ coprocessors. This approach gives an interface modelling considering communication adaptation concepts/context throughout the integration steps. Graph formalism has been established to specify the interface considering the IP execution cycle accurate behaviour. This allows for automatic generation of interface architecture for simulation towards its synthesis. We illustrate the utility of the proposed framework that enables faster simulation times compared to existing methodologies which allow the designer to quickly evaluate alternative system implementations.**

I. INTRODUCTION

Easy and quick assembling of various cores to obtain a fully functional system has not yet become reality. In fact, the core integration widely remains a manual and error-prone process. Despite the key idea of dissociating communication and computation [1], designers are still forced to fully understand the functionality and interface features of the components they want to integrate in their design.

In order to resolve this problem, several methodologies called "Communication-based Design" emerge and open the door for innovative solutions for SoC design. Techniques and tools [2] relative to those new methodologies aim at reducing the system design time when complexity becomes too large. Unfortunately, such tools do not efficiently manage low-level details relative to IP interface synthesis (computing latency, I/O timing constraints etc.). In order to facilitate plug-n-play style IP reuse, two approaches have recently been dealt with. The first one defines a standard bus protocol like CoreConnect from IBM [3]. The spreading of the "platform-based design" methodology imposing particular buses on the chip includes this tendency. The second one defines standard protocols that interface between a bus wrapper and the core internals. For example, VCI [4] and OCPIP [5]] are interconnect-independent protocols. Unluckily, the use of these standard protocols does not exempt the wrapper task of the data time control (beginning time and end time). Other work has been done on interfacing with cores being based considering the whole system execution: COSY [6], Fast Prototyping [7], Polis [8], Coware [9]. These approaches represent dedicated approaches for specific application field. Furthermore, they require having a perfect knowledge of the protocols of both the sender and the receiver. Other tendencies rest on the capability of languages. For example, SystemC methodology talks about transaction level modelling in [10] that aims at communication modelling to optimize simulation speed. However, it does not address automatic generation of such models. Moreover, the emergence of networks on chip requires new interface features like intelligent scheduling control and update parameters capabilities of modules in systems [11].

So, methodologies based on the idea that a virtual component can be reused without additional design effort are not valid any more. This is particularly true with applications dominated by data processing. Indeed, the systems working on strong volumes of information require a viable solution of implementation to take into account the data exchanges organization [12]. Besides, automatic IP assembly for SoCs design involves compatibility checking between the IP protocols, system level simulation of the solution and interface synthesis considering protocol mismatches and component composition. These additional steps increase the designers' effort and time required for chip design. Different formalisms can be used for modelling communicating hardware for automation [13].

In this paper, the proposed formalisms of interface and input/ output (I/O) automata control present significant differences. It deals with low-level details for IP core integration process. The established formalism has been developed to specify the interface adaptation based on Finite State Machine (FSM) based framework considering its cycle accurate behavior. The interface specification can then configure a generic interface architecture structure. So, the approach allows 1) to obtain an IP encapsulation model for simulating, which hides low IP functionality behind a high level interface forwarding its adaptation during its lifetime; 2) to enable a systematic interface architecture generation towards its system verification and its synthesis. This research targets "component-based design" and "platform based design" focusing on automatic IP integration into multiprocessor SoC (MPSoC) context.

The paper is organized as follow. Section 2 presents the proposed design flow to model interface for IP integration steps. In Section 3, architecture modelling is detailed. In section 4, we give the experimental results following the application of the considered integration approach to the Object Motion Detection algorithm. Finally, in section 5, we conclude this work and we present the future work.

II. INTERFACE MODELLING

The goal of the specification method is to abstract system communication to a level that is independent of the protocol

module and the technology details, while being easy to use and expressive [14]. At this level what is known is the I/O signals of interacting modules and their inter-module data and control flow requirements, which is equivalent to the interface behavior. This section presents an overview of the proposed design flow describing IP integration steps. It details spacio-temporal hypothesis that will be considered.

A. Method Overview: Design flow for IP integration steps

Target SoC architecture includes hardware accelerators and coprocessors which represent computing units that have not enough control logic to be autonomous. These accelerators need external control unit to manage the communication transfers. They are considered as slaves and controlled by the master processor through the architecture interconnect.

In our approach, we define the wrapper at two abstraction levels. At the first level, we define an abstract wrapper architecture that hides the communication details (encapsulation model). The second level will corresponds to the RTL level where the abstract wrapper is implemented.

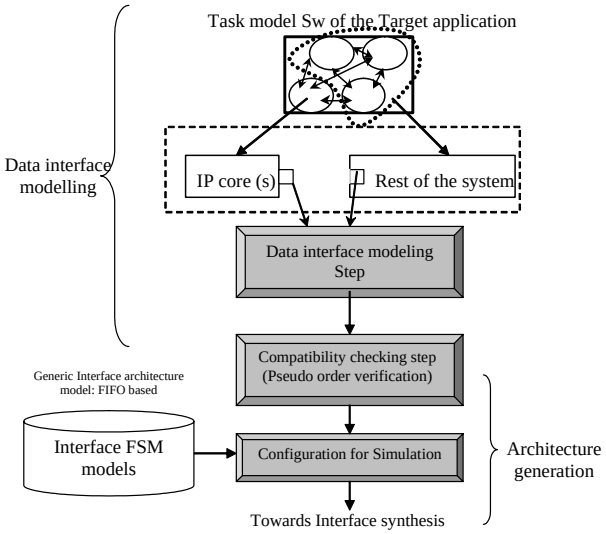

Figure 1. Design Flow steps overview

Figure 1 shows IP integration steps starting from the system specification as interconnected tasks. Mapping the specification to SoC architecture is performed by the partitioning step. Hardware system tasks are considered in their synthesized form. The data interface-modelling step considers spacio-temporal data scheduling to generate correspondent abstract models. Thus, our integration methodology is built around two main steps: interface behavior modelling for checking compatibility and interface architecture generation (figure 1). Compatibility is verified if the integrator system (rest of the system) verifies data scheduling at IP interface. Generic interface architecture is defined as FSM models. The I/O interface modelling is then needed for configuration to automatically generate the correspondent interface architecture structure.

The approach targets both the simulation and the synthesis since the IP interface can be simulated and is synthesizable. The work presented is (1) the definition of a method to generate wrappers automatically and (2) the introduction of a generic wrapper architecture that can be applied to coprocessor and accelerator components.

B. Integration constraints

The communication interface generation is based on the IP interface (characterization information), system requirements (task model priorities that define the scheduling of the whole execution of the target application) and the designer specification exigencies (interconnect style, use of Direct Memory Access (DMA)...). The integration mechanism is strictly target data-independent communications. So that, the information needed for the interface modelling allows for the unambiguous identification of which data is input (respectively. output) at each clock cycle for each input port (respectively. output). The interface block disposes a local memory with first-in first-out (FIFO) semantics for storing data. If the production order and the consumption order are different in a point-to-point communication, a simple FIFO in this case is not enough for a transformation of N_dimension data structure to 1_dimension data structure. It is necessary to envisage more complex mechanism for intermediate memorization according to [15].

For this work, we suppose that the system knows the information related to the data succession in each port but it is unaware of the sending order and the sending time of the data to the input interface. The system is locally synchronous at the IP interface side but globally asynchronous. So that, the production order and the consumption order are the same for a flow transiting on a given port. This restriction allows the definition of a pseudo order that guarantees the data order for each structure. The data are assumed a power of 2. The inputs and the outputs are supposed uninterrupted with beginning and end virtual times defining the delay of an IP iteration computation. An iteration of computation is a repetitive execution of a basic motif of data sequence considered by the IP. The communication interface encapsulates the IP architecture (adaptation of the IP interface to the interconnect) and guarantees data transferring. This kind of IP integration problems are accentuated in a context of multiprocessor SoC design.

To model the data exchanges between the environment and the virtual component, we use a graphic modelling defined in the next subsection. These abstract graphs based model completely express the input/output communication behavior of the system interface and the IP interface relying on the I/O data scheduling behavior.

C. Data InterfaceModelling: Abstract Graph Models

IP cores can be categorized into three main types: Soft, Firm or Hard cores [16]. The approach we propose can be applied whatever the type of IP. The information needed describes the I/O signals and their data and control flow requirements, which is equivalent to the IP interface behavior relating to an iteration of computation. It should be provided by the IP designer and constitute a key element of successful integration.

The proposed graph models definition illustrated in figure 2 rests on the work of Ku and Michel [17]: the IOCG graph (Input Output Constraint Graph) and the IPERM model (IP Execution Requirement Model) [18]. These models support the modelling of (1) the type of transfers, (2) the temporal variations of the data arrival times, (3) the data exchanged sequencing and transit ports etiquette, (4) the related mechanisms to the communication protocols.

Definition 1: Input Output Scheduling Graph (IOSG)

IOSG is a hierarchical polar directed graph IOSG (V, E) where the nodes set V = {v0,… ,vn} represents data transfers stages containing the whole of the consumed/produced data equipped with the number of transit channel. v0 and vn are respectively the node source and the node destination and symbolize the beginning and the end of a data transfer sequencing. The arcs set E= {va, vb} represent the transfers sequencing specified by deadlines. A weight noted "wab", associated to the arc noted "Eab", represents the transfer time separating two transfers from data va and vb. IOSG represents the IP interface behavior considering that the system works without interruptions.

Definition 2: Structure Input Output Scheduling Graph (SIOSG)

Eliminating the time constraints and information on the indexing in the data structure from IOSG, SIOSG is defined for each data structure. It is used to know the sequencing order for each data structure.

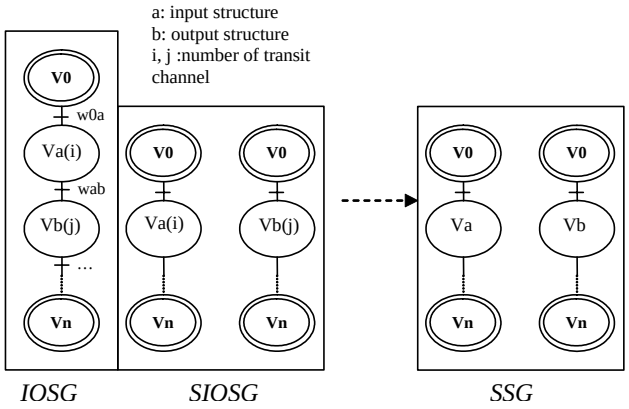

Figure 2. Graph models

Definition 3: System Scheduling Graph (SSG)

For system interface behavior, we propose a graph resulting from the IOSG: the SSG. It is a third type of graph which is used to specify the data produced/consumed by the system. The SSG is obtained by eliminating the indexing information from the data in graphs SIOSG. The system data transfer must validate SSG to satisfy the hypothesis of the pseudo-order. The pseudo-order granted to the data sent/received by the system (SSG) checks the data order for each structure. For the SSG, we do not use time information. So that, the encapsulation model can target the integration of synchronous and asynchronous architectures. Indeed, the interface time constraint of the IP is expressed by a delay (wab in figure 2) weighting the transition arcs.

D. Compatibility Checking Step

SIOSG, SSG result from the same model of graph: IOSG. This compatibility enables integrating IP core automatically in the design work flow. SSG in figure 3 is used to define the order definition in which the system is sending/receiving the data. According to this order, the software driver is generated. It is used to pilot the interface. This software part of the interface must order the communication of the data between the IP and the memory of the integral system. It must be generated automatically starting from the specification of the entries left to the interface the IP. It is carried out by the processor which orders the material accelerator (IP) via the hardware interface. IOSG configures the generic interface architecture modules defined as Interface FSM models. Compatibility checking step verifies the pseudo order. For that, two sides are considered: the behavior at IP interface and system data transfer to ensure their compatibility. If the graphs compatibility is not validating, the data interface modelling step is re-studied. The key idea of the proposed approach is to model data constraints behavior as abstract graphs considering the integrator system and the IP execution.

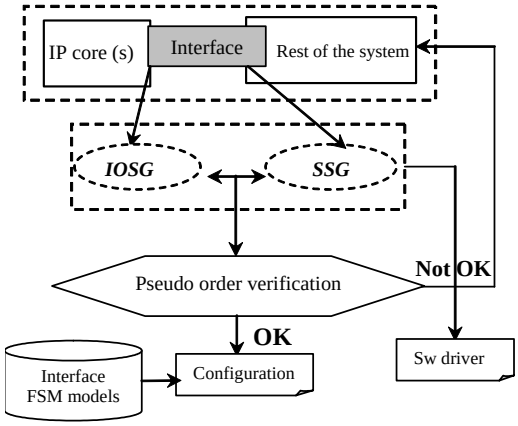

Figure 3. Compatibility checking

This specification is considered with a generic interface architectural modelling allowing automatic IP integration in a context of re-use. The interface FSM models in figure 3 are defined as a library to generate the interface architecture modules. This library and graphs interaction are detailed in the following section in the context of the proposed generic interface architectural structure.

III. GENERIC ARCHTECTURAL MODELLING

An IP is considered as a cycle accurate "black-box". Only the communication interface is known. It has a number of I/O ports, each having a bit-width, latency and data ports scheduling. A key strategy for minimizing the system design effort and managing growth of the application complexity is using modules library. This holds design information and simulation models related to each module, making it reusable among several applications. So, the generic communication interface relies on FSM as a model of computations interacting with interface graph models and communication with the rest of the integrator system.

A. Architecture Interface Structure

Input communication channel mechanism transforms the request from an external slave port into a signal dedicated to the IP communication interface. Output interface mechanism translates the data back from the interface into a slave response. The interface sub-modules follow the rules of the design reuse: their FSMs are instantiated according to the spatial and temporal I/O constraints described as graph models. We use a network of FSMs describing the communication interface while abstracting its implementation. It is composed of: 1) an FSM for the input FIFO, 2) an FSM for the output FIFO, 3) an FSM for the controller unit. Figure 4 illustrates the interface FSM and graphs interdependency. SIOSG allow the correct operation of the controller_FSM in the following FSM network

designing the interface modules behavior. IOSG and SSG describe and manage the behavior with the interface.

We consider the particular aspects of hardware design and metaprogramming capabilities with SystemC [8] for simulation: high-level design using design patterns and generalization using templates C++. Encapsulation resides in putting methods into C++ classes. The technical details which will be explained deal with the communication interface of the input channel. We consider the output part in the interface structure design as the symmetrical.

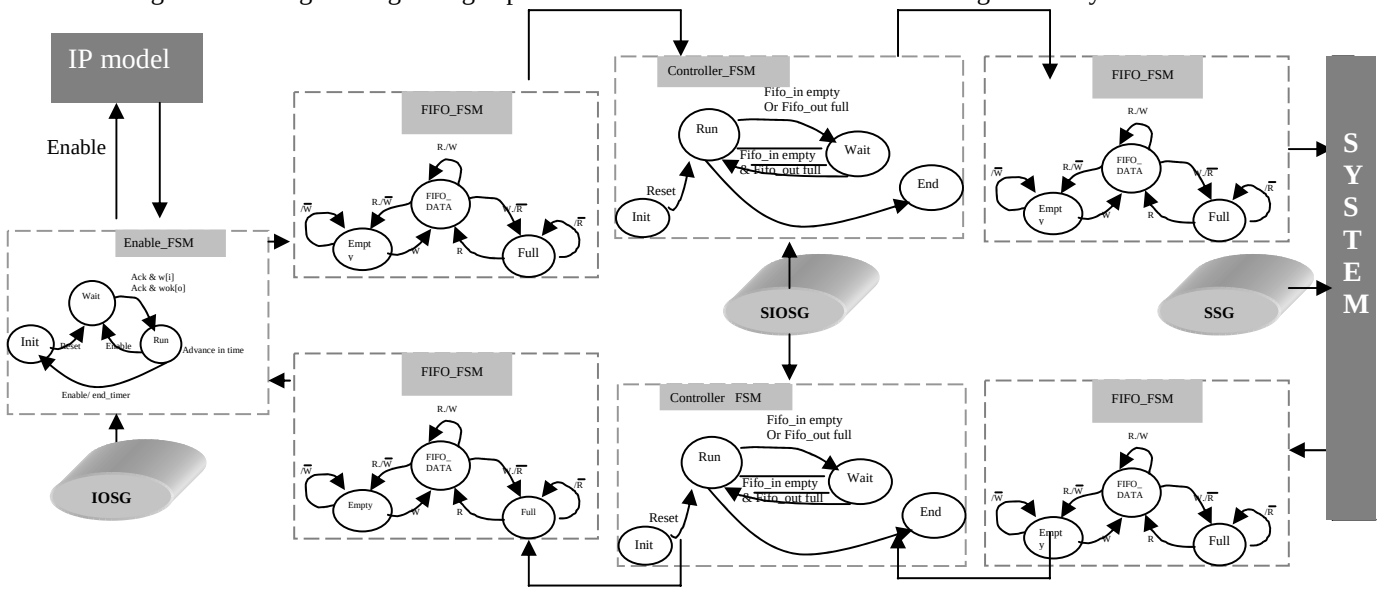

Figure 4. Interface FSMs and graphs interacts

In order to connect the IP to other SoC components, the interface has to respect refinement and specification of the I/O protocols, data sequence orders and timing information constraints of data transfers. To be able to integrate the IP, more information in particular the size of the interconnect and the size available to buffer data (FIFO size and FIFO depth) are needed. There are as many FIFO_IN as the IP has input ports. The Controller_FSM dispatches data information according to the real size and their corresponding structure. It is configured according to the graph models before data are transmitted. The Enable_FSM is cadenced with a virtual clock (simple counter) to control IP temporal execution. It is reset for new computation iteration and can freeze the behavior of the IP if data are not ready (for example, if the system has no yet provided the data).

We distinguish two design solutions in order to dispatch serialized data to the corresponding ports of the architecture interface structure. In one hand, transferring data can be realised by decoding address structure from FIFO input. This relies on the initiators addresses (addresses allowing knowing the structure). We suppose that the interconnect protocol vehicles only the initiator address for an input/output structure. This assumption allows the controller decoding addresses to differentiate the data port. In the other hand, the conception of the interface architecture doesn't consider addresses. A sentence is preliminarily decided as a set of data that will be sent/ produced to/by the architecture to feed the architecture /system during computation iteration. A pattern describing data scheduling can gather several motifs in order to take advantage of the interconnect burst mode. Input pattern and output pattern must be carefully merged in serialized form taking into account available cells to avoid deadlocks. We consider a pattern describing the order in which the serialized data arrive to the controller FSM. This order is the same as the system data sending order. It is written in the form of software driver piloting the architecture of the IP.

The experimentation of this IP integration approach using a generic interface structure is detailed in the following section. Discussions deal with simulations results differentiating between the two ways considered to dissociate serialized data by the communication IP interface.

IV. Experimental results

The automatic interface generation has been tried on the Object Motion Detection (OMD) algorithm. This algorithm allows the detection of motion objects in a video sequence using mathematical and morphological operations on a sequence of successive images [20] as shown in figure 5. The "Mean" function computes the mean between the input and N old images in the algorithm. This function is used to be as a hardware accelerator in order to experiment the proposed encapsulation methodology. A "Mean" function computes the mean between the input and N old images in the algorithm.

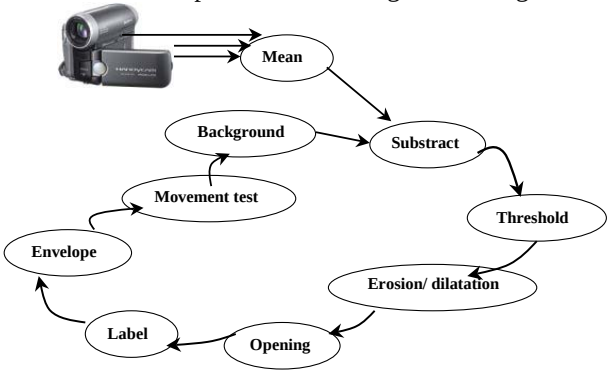

Figure 5. OMD algorithm

This IP is synthesized with the high level synthesis tool: GAUT [21]. In this case, the "Mean" IP core requires

extracting data line by line, from every image (i.e. data structure). In order to simulate the whole system, we have used the SoCLiB platform. SoCLiB proposes an open modelling as a simulation platform for SoC design to allow modelling of complex systems VCI (Virtual Component Interface) compliant [22]. The simulation platform we have used contains a standard memory (RAM), a MIPS R3000 processor with its cache, and the hardware accelerator ("Mean" IP). Both the software part (“driver”) and the hardware part (the interface) of the communication channel are generated from the same data transfer description. The host processor initiates the communications. It sends (respectively. receives) data according to the software script scheduling. The script has to respect the order by structure (pseudo order).

Simulations have been carried out and validated under Pentium M Centrino (1,5 GHz, 512 MB RAM) with Linux environment as O.S, SystemC-2.0.1 simulator and GCC 3.3.1 compiler. The overall hardware/ software design is then co-simulated. The throughput of this system, measured in frames per second, is 12,5 for frames in gray levels Bmp format of 155x235 pixels (Design 1 in figure 5). The system should be able to process 25 frames per second as a real-time standard. The problem is in the slow communication link. Clearly, the bottleneck is the system 32-bit width of the SoCLiB platform. However, the platform is used to validate our interfacing approach. Simulation results performance will be compared with real prototyping of the interface.

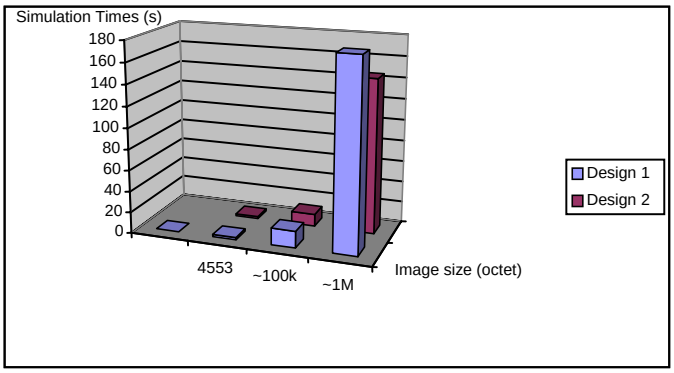

Figure 6. SoCliB OMD Simulation times

To compare performances of the proposed interface designs, communication interface decoding data address (design 1 in figure 5) is flexible since it can target the multiprocessor context. It is independent from the interconnect protocol and its software driver is simple to generate. However, it is not optimal in terms of memorization units: FIFO_IN contains the data and the sender address. The controller of the interface contains an address decoder. On the other hand, communication interface dissociating data according to a fixed pattern is optimal. It is independent from the interconnect protocol and reduces the simulation time (figure 5). However, it can’t target MPSoC context without a scheduling arbiter and its driver is more complex to generate.

The specification tends to simulate faster than the RTL (Register Transfer Level) model and favors systematic interface synthesis. Moreover, it guarantees more safety in building interface architecture in a SoC/MPSoC context. In fact, all the interface sub-models are generated as CABA (Cycle Accurate Bit Accurate) models defined by the Mealy Moore FSM. This style of implementation generates simulation models that are semantically similar to final RTL implementation.

V. CONCLUSIONS AND FUTURE WORKS

This paper proposes an approach of encapsulation to integrate lower level IP in a multiprocessing context. The solution is based on a generic configurable interfacing model. Abstract graph models illustrating low-level constraints for IP execution details have proved to have a high utility value for the integration task where internal features of the IP core are hidden. The design methodology relies on library which contains interface modules and a strategy to capture the protocol and timing information necessary for interface generation. Various choices allowed in the interface design have been described. The communication of the system is validated using SystemC fast simulation. In addition, the associated generic architecture modelling allows an easy hardware implementation of the interface.

We are currently focusing on providing tool supporting the methodology described here. We expect in the future to design a tool based on the presented design models and formal illustration in order to automate the RTL wrapper generation before the logic synthesis process.

REFERENCES

[1] J.A. Rowson, A.S. Vincentelli., “Interface Based Design”, DAC 1997, California, USA.

[2] P. Chou, R. Ortega, K. Hines, G. Borriello, “IPChinook: An Integrated IP-Based Design Framework for Distributed Embedded Systems”. DAC’99.

[3] IBM CoreConnect Bus Architecture Available at: http://www-3.ibm.com/chips/products/coreconnect

[4] Virtual Component Interface Standard VSI Alliance™, OCB 2 2.0, April 2001.

[5] Sonics Inc, "Open Core Protocol Specification 1.0", 2000

[6] J-Y Brunel, et. al., COSY Communication IP’s, Proc. DAC, 2000.

[7]F.. Pogodalla, R.Hersemeule and P.Coulomb, “FastPrototyping: a system design flow for fast design, prototyping and efficient IP reuse”, CODES 1999.

[8] F. Balarin, et. al, “Hardware-software Co-Design of embedded system : The Polis Approach”, Kluwer 1997.

[9] S. Vercauteren, Harhuore/So/iwore co-design o/ opplicofionspkfic Heteregeneous orchilecrures, IMEC, PhD thesis, DEC.1998.

[10] K. Svarstad, G. Nicolescu, A.A. Jerraya, “A model for describing communication between aggregate objects in the specification and design of embedded systems», DATE’01, Munich, Germany, March 2001.

[11] L. Benini, G. De Micheli, “Networks on Chip: A New SoC Paradigm”, IEEE Computer, January 2002.

[12] P. Coussy, A. Baganne, E. Martin, “A Design Methodology For IP Integration” , ISCAS 02, Phoenix. USA

[13] A. Baganne, J.L Philippe, E. Martin “A Formal Technique for Hardware Interface design”, In. IEEE Trans. On Circuits And Systems, Vol.45, N5, 1998.

[14] Ahmed A Jerraya, “Long Term Trends for Embedded System Design”, CEPA 2 Workshop, Brussels, Belgium, March 15-16, 2005

[15] C. Z.-Ianculescu Alexandru, T. B. Kienhuis and E. Deprettere, “Solving Out of Order communication using CAM memory an implementation” , ProRISC 2002

[16] Virtual Socket Interface Alliance, http://www.vsi.org

[17] G.De Micheli, D.ku, “ High level synthesis of ASICs Under Timing and Synchronization Constraints”, 1992, ISBN 0-7923-9244-2.

[18] P. coussy, A. Baganne, E. martin “A Design Methodology for IPIntegration”, ISCAS’O2, Scottsdale, USA, 2002.

[19] Systemc Synopsys Inc., SystemC, available at http://www.systemc.org

[20] G. Moroz “Optimisation d’un algorithme de compression d’images”, training course SYEL group Pierre et Marie Curie University ”, 2002,

[21] http://web.univ-ubs.fr/gaut

[22] http://soclib.lip6.fr/