IP Cores Integration in DSP System-on-Chip Designs

Philippe Coussy, Adel Baganne, Eric Martin

LESTER – Université de Bretagne Sud – BP 92116 - 56321 LORIENT Cedex, France
{firstname.lastname}@univ-ubs.fr

ABSTRACT

Successful integration of IP/VC blocks requires a set of view that provides the appropriate information for each IP Block through the design flow for an IP-integration system. In this paper, we present a methodology of IP integration in a System-on a chip (SOC) design, that exploits both IP designer and SOC integrator constraints. First, we describe a method to extract and specify IP functional and timing constraints (I/O sequence transfer constraints) from the IP core. Second, we propose a modeling style of the integration constraints and a technique for merging them with IP constraints. This technique allows the specification and design of an optimized IP interface unit required for IP-Socketization. The synthesis output is synthesizable VHDL RT of the interface, a detailed Bus-Functional model of the IP core towards Cosimulation.

I. INTRODUCTION

The complexity of modern embedded systems design requires designers to leverage the reuse of both software and hardware modules. Reuse is done at the chip level called *cores*, *VC* (Virtual Component) or *IP* (intellectual Property) available in various forms ranging from soft cores to hard cores [1]. These components represent functions of specific domains like signal processing (DCT, FFT), telecommunication (Viterbi, Turbo codes) multimedia application (MPEG2, MPEG4, JPEG) etc. The IP core are integrated in a system-on a chip *(SOC)* which a typical architecture is depicted in Fig. 1. Such architecture includes digital signal processors (DSP), shared memory, bus controller and a set of hardware IP blocs connected to the system bus through specific interfaces or Wrappers. IP cores can be, previously or not, created internally by the SOC designer team but can also be bought from an external source. Despite efforts oriented on IP core exchange and IP core catalog development ([1]), communication problems and timing issues can cause SOC design to fail. A successful IP core integration requires the designer to take into account the main following tasks:

1. **Synchronization:** the components have to be synchronized on different aspect such as global execution, data exchanges and protocols.
2. **Protocol conversion**: Assure the protocol conversion between blocks that use incompatible protocols. Wrapper can be used for this purpose but introduce overhead that should be taken into account with the timing constraints.
3. **I/O buffer synthesis:** data may be buffered to ensure the system behavior and to meet timing constraints.

In practice, the vision of easy and quickly assembling a SOC using cores has not yet become reality for many reasons. Actually, even if cores are pre-verified, it does not mean the whole system will work when they are put together. The integration of cores into a SOC is widely a manual and error-prone process because it requires the designers to fully understand the functionality and interfaces features of complex cores. Besides the protection of the internal IP block architecture can lead the designer to hide some information that may be essential for the IP integration.

Different approaches attempt to ease IP integration today by defining design methodologies or techniques to solve specific problems. Virtual

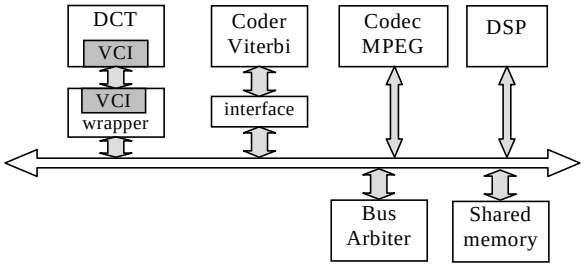

Fig. 1 A Typical SOC Architecture

Socket Interface Alliance (VSIA) [2] focussed on defining a standard on-chip bus, but this soon appeared to be difficult [3]. In [4] authors proposed an interface-based design methodology that attempts to ease integration by separating the communication from the behavior. VSIA [2] provides a Virtual Component Interface (VCI) standard that defines a generic cycle-based address-mapped point-to-point communication protocol. The use of this kind of standard interface can add communication overhead [5]. Some EDA companies provide a set of tools that allows incorporating IP cores for high level specification and system cosimulation. Coware N2C provides a Virtual Bus [6] to connect each system block and allows the HW/SW cosimulation at the conceptual and architectural level. VCC (Virtual Component Co-design) [7] proposed by Cadence is a system-level environment for HW/SW co-design and IP reuse. This tool allows specifying the system functionality, defining the system architecture, performing the partitioning, refining communications between blocks and analyzing system performances. However, such tools require the system designer to have an efficient IP core modeling adapted for the co-simulation and system-level performance analysis steps. Furthermore, they can not manage low-level details relative to IP interface synthesis (computing latency, I/O timing constraints etc.). Few works have addressed the problem of IP integration and interface synthesis in a global way. Some of them addressed the problem of interface synthesis between standard components that have incompatible protocols [8]. In [9] authors describe the problem of IP wrapper synthesis and overhead delays to be considered for integration. Others addressed the problem of interface synthesis from hardware I/O transfer sequences in a co-design approach [10].

In our point of view a global methodology of integration, going from the system level performance analysis down to the synthesis step, is the best way to solve the problem of IP core reuse. In this paper we present a methodology of IP integration that exploits both IP designer and SOC integrator constraints. The paper is organized as follow: First in section 2, we give the formulation of the IP integration problem. In section 3, the proposed integration flow is presented. As illustration, section 4 describes an integration example of an FFT core and the synthesis results obtained.

II. PROBLEM FORMULATION

Let us consider a SOC architecture composed of an IP core and a DSP(see Fig. 2). This IP core receives data X,Y,Z from the DSP and sends its result W to the DSP over a single bus. Two functional units compose the IP core: one memory management unit and one processing unit that exchange data over two busses. All the data used

in the processing unit are read from the memory management unit in a fixed order sequence $S_{IP} = (X,Y,Z)$ i.e. $t_x < t_y < t_z$. The produced output signal W is also stored in the memory management unit. The memory management unit includes a fixed address generator. The order of the data transfer sequences is therefore completely deterministic. Let us consider the I/O sequence constraints imposed by the DSP to be the sequence $S_{SYS} = (X,Z,Y)$. The produced result will be false because of the wrong data sequence order presented to the IP core interface.

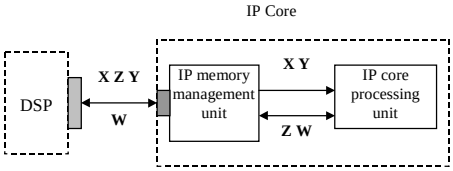

Fig. 2: IP core integration problem

Let us now consider the following DSP data transfer sequence $S'_{SYS} = (X,Y,Z)$. If the timing requirements imposed by the IP core are not respected the integration process will fail. Successful integration of IP blocks requires a set of views that provides the appropriate information for each IP block through the design flow of an IP-integration system. Hence, a methodology of IP integration has to exploit both IP provider and SOC integrator constraints. In our work, we consider the real time implementations of computing intensive applications such as image and signal processing. In our work, we consider the real time implementations of computing intensive applications such as image and signal processing. So, the functions processed by the IP cores are supposed to be deterministic.

III. DESIGN FLOW

An overview of our design methodology is described in Fig. 4. The design flow concerns on the one hand *IP design* tasks and on the other hand *System design* and *integration tasks*. The point of contact is done by means of an **IP Execution Requirements Model** (**IPERM**) and an **IP Delay Model** that describe low-level details for IP core integration. These models should be provided by the IP designer and constitute a key element of successful integration from the performance analysis task to the synthesis step. As it will be seen in the next sections these models offer to the IP designer an efficient protection of the internal description by hiding architecture details while keeping clear the description of the functionality requirements.

A. IP Design

The design of an IP core begins by a functional specification that describes the behavior of the component. The IP core is then described with hardware language more suitable for implementation. Usually, IP core architecture is based on four main functional units:

- Processing Unit *(PU)* releases all the arithmetical operations
- Memory Management Unit *(MMU)* stores data during executions.
- Control Unit *(CU)* drives all the precedent described units
- Interface Unit *(IU)* manages and controls the communications between internal architecture and external environment.

The functional units previously described can be designed by means of manual RTL description or high-level (behavioral) synthesis tools such as SystemC Compiler from Synopsys. Based on these descriptions, we can extract the IPERM model for the IP integration. This design step is discussed in the next section.

B. IPERM Model Generation

At this stage of the design flow, the functional units of the IP core are described at the RTL level. The processing unit is modeled with a Finite State Machine with Data-path *(FSMD)* model described in [8]. An FSMD differs from the FSM in that it may include variables with various data types. This modeling of the processing unit will be named M_{PU} in the rest of the paper. The memory management unit is modeled with a set of FSMD $M^{MU} = \{M_{MU1},\ldots, M_{MUi}\}$ where M_{MUi} represent the i^{th} storage element with $1 \leq i \leq N_{IP}$ and N_{IP} represents the number of busses that connect the processing unit to the memory management unit. The memory management unit and the processing unit are therefore modeled by a set of communicating FSMD.

The **first step** of the IPERM generation we merge the M^{MU} states with M_{PU} states in order to obtain a single FSMD M_{IP}. The **second step** merges sequential M_{IP} states without I/O data dependencies into a new state called *Super State* (see Fig. 3). Thus this super state represents a set of computations and memory accesses that are released between two I/O data transfers.

First phase

```
1)  For all the state in M_PU
2)      For each data dependency of the current state
3)          Merge the data dependent M^MU states with the current state
            in a new M_IP state
4)      End for
5)  End for
```

Second phase

```
6)  While M_IP state without I/O data dependency
7)      For all the state in M_IP
8)          If (the next state has no I/O data dependency)
9)              Merge it with the current state
10)     End if
11)     End for
12) End while
```

Fig. 3: Pseudo code of our IPERM design algorithm

Since the IP core is described at the RTL level and that all the *PU* I/O transfer sequences are fully specified. Timing information can therefore be extracted and added to the generated model of the IP core such as data lifetime $L_T(d)$, input data latest arrival date and output data earliest emission date $T_{PU}(d)$. The transfer delay due to the data exchange protocol between the processing unit and the interface unit is expressed by Δ in cycles (Fig. 5).

The final IPERM model is an annotated FSMD:

- A set of super states output by states merging steps
- Timing frames where the data transfers can occur

Fig. 9 depicts the set of communicating FSMD that represents the IP core described in section 2. The MMU is composed of two memories. The PU reads X, Y in the first one reads Z and writes W in the second one. The PU FSMD includes five states. Fig. 9, 10 respectively show the result of the first and second phase.
Finally, the obtained IPERM Model M_{IP} is composed of three super states.

C. IP delay Model

Embedding intellectual property models into high-level system description allows the system designer to simulate and evaluate appropriate virtual components during the performance analysis phase. For this purpose, the functional description of the IP core is associated with a delay performance model that describes its timing requirements. This enables the system designer to anticipate the synchronization problems between the different components of the system and the IP core. Taking care about the timing requirements of the IP core early in the system design flow allows an optimized integration. The IP Delay Model is generated from the IPERM model since it describes the functional and timing requirements of the IP core. For instance, in [7] IP core can be integrated at the system level for performance simulation. For this purpose, the IP functional model is associated with a DSL performance model using the Delay Script Language.

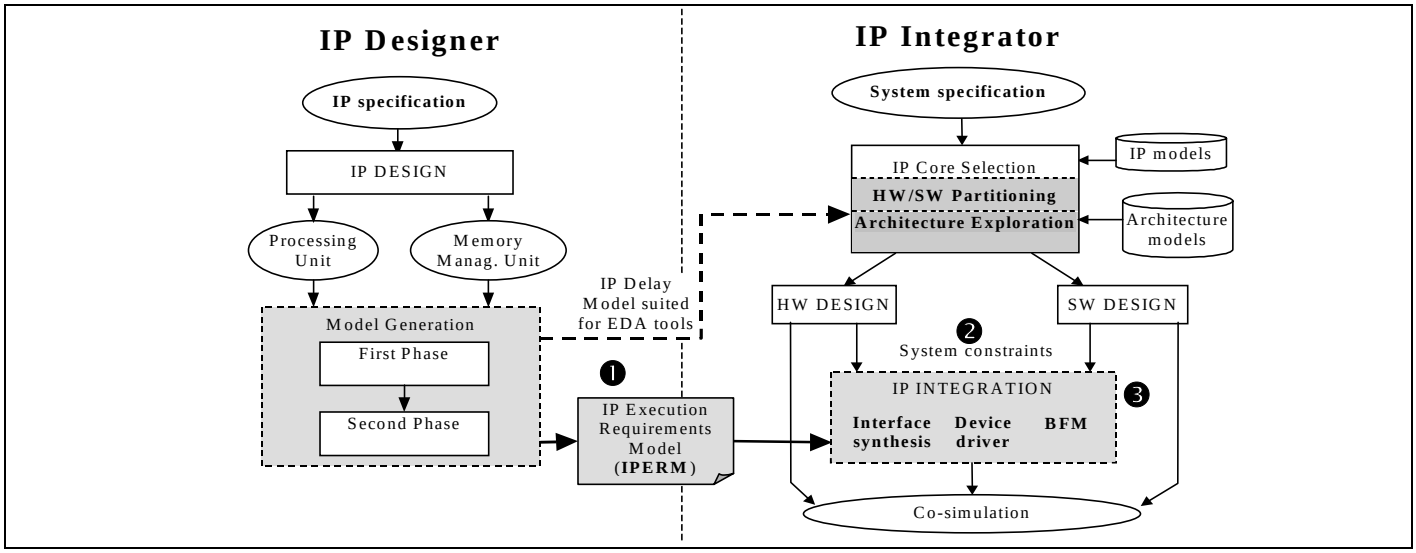

Fig. 4: Design Flow for IP integration

C. System Design and IP Integration

The system design begins by a specification capture of the desired application. The system designer select IP cores from a database considering constraints criteria e.g. speed, area, or power etc. Follows an architecture exploration concurrently with a set of co-design techniques [12] (HW/SW partitioning, system performance analysis, communication synthesis HW,SW and interface generation). The performance analysis task allows the designer to explore independent dimensions of behavior and architecture to reach optimal design performance within the given constraints. The hardware and software design tasks generate respectively an RTL description of ASIC/FPGA blocs and C/C++ code. To satisfy the integration constraints and to carry out the IP-Socketization, the system designer can incorporate the low-level details of IP provided by the IPERM model (Latency, I/O sequence transfer, I/O timing constraints etc.).

1. Integration Constraints

Definition:

All the following parameters specify the communication features between the system and the IP core.

- δ_a : probability of access to the communication medium when the communication is done via a shared On-chip-bus
- δ_p : constant which depends on the used transfer protocol
- γ_w : overhead introduced by the bus wrapper
- δ_t : data transfer delay

Integration constraints can be of three major types: *(1) fully specified* by precise dates of data transfer and data sequence order; *(2) partially specified* (timing frame of data transfer and data sequence order or partially or not ordered data transfer)- *(3) Unspecified*. These constraints are specified for the N_s busses that connect the IP core to the rest of the system. Each one is modeled with an FSMD that describes the bus transactions. Hence the set $M^S = \{M_{S1}, \ldots, M_{Si}\}$ models integration constraints for each external bus: M_{Si} represent the i^{th} bus with $1 \le i \le N_S$. The dependencies set between the M_{IP} and M^S are represented by a hierarchical links set. Each link can be decomposed in two subsets: data links and control links. The control links hence model a data exchange protocol as handshake for example. The timing frame of a data associated to the hierarchical links take into account the data transfer latency and the data exchange protocol delay between the system and the interface unit of the IP core.

For each data the transfer delay between the system and the interface unit (see Fig. 5) is expressed as $\delta = \delta_a + \delta_p + \gamma_w + \delta_t$. Fig. 11 depicts

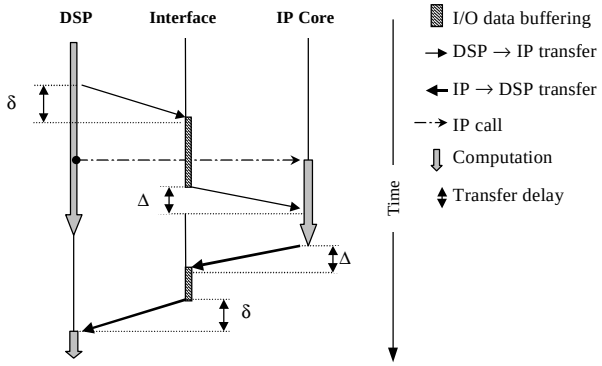

Fig. 5: Integration Constraints Specification

the constraints imposed to the IP core of our example. The DSP and the IP core exchange data over a single bus modeled by the M_S FSMD. The overhead added by wrapper is currently not supported and is left for future work.

2. IP Interface Synthesis

Merging integration constraints and IP constraints allows the design of an optimized IP interface unit required for IP-Socketization. Each I/O data is characterized by two timing frames: $\boldsymbol{T_{IU}}$ that represents the interval in which the transfer can occur; $\boldsymbol{L_T(d)}$ that represents the data lifetime in the interface unit. These information are generated by merging: (1) IP functional and timing constraints provided by the IPERM model (2) system integration constraints N_s and data transfer sequences, (3) transfer delays Δ, δ. The generic interface unit targeted by the synthesis is composed of buffers for storing I/O data and an FSM based controller. The hardware synthesis step uses algorithm working from timing requirements and data ordering information. Interface hardware synthesis generates a synthesizable VHDL RT description. A BFM [13] can be generated manually based on the system bus specification. It is written in VHDL and will drive the simulation with the core's bus response. A new IP delay model can be generated at this stage taking into account the interface unit effects on the timing constraints of the interfaced IP core. This is not supported in the actual design flow and is left for future work.

IV. IP INTEGRATION EXAMPLE: FFT CORE

The presented method has been applied to an IP core that implements an 8-points complex FFT optimized on area. The system is composed of one DSP and one IP core that communicate through a point to point

link (see Fig. 6). Real and imaginary parts are sent in parallel over one data bus that connects the DSP and the FFT IP core.

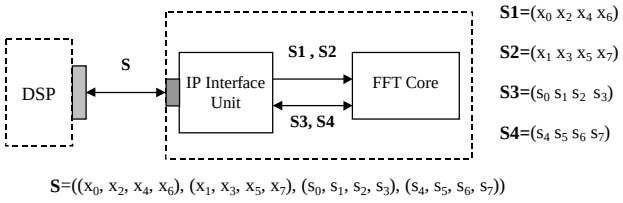

Fig. 6: Integration Constraints of the FFT core

The memory management unit implements eight 16 bits width registers containing the real and imaginary parts of the data each one coded with 8 bits. The *PU* exchanges data with the *MMU* over four 16 bits width busses for intermediate results. The *PU* reads input data and writes final results from or to the system on its I/O ports. The data are exchanged serially between the DSP and the FFT core and in parallel between the interface unit or the memory management unit and the processing unit. A simple handshake enable protocol synchronize the DSP and the FFT core (δ_p=1, δ_t=1). The processing unit and the memory management unit are synchronous: the PU reads and writes data on internal busses at fixed dates. Fig. 7 shows a piece of IP delay model script of the FFT core written with the Delay Script Language. DSL is a C-like language used in the VCC tool [7] to describe the DSL Performance Model of hardware components. This IP delay model associated with the system-level description of the FFT core can be used for the performance analysis.

```
…
delay_model () {
input (ai); input(ar);input(bi);input(br);        /*Read the inputs*/
run();                                            /*computing part*/
delay('9.0e-9');                          /*Wait before posting output*/
output(wi);oupout(wr);                            /*Post the outputs*/
input (ai); input(ar);input(bi);input(br);   /*Read the next inputs*/
…}
```

Fig. 7 : IP Delay Model of the FFT Core

Table 1 shows the parameters of the systems and the results output by the interface synthesis. Fig. 8 depicts the synthesized interface that allows the integration of the IP core into the system design.

Constraints				Interface			
MMU	δ_p	δ_t	Δ	FSM	Registers	Mux	Demux
16x16 bits registers	1	1	1	11 states	4x16 bits	4→1	1→4

Table 1:Experiments parameters and Results

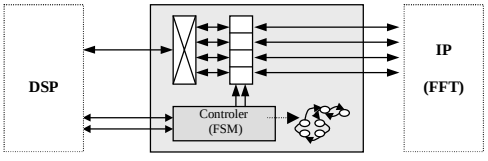

Fig. 8: Interface Unit of the FFT example

The synthesized interface unit is optimized for the data sequence transfers imposed by the DSP. One multiplexor and one demultiplexor process respectively the parallel-serial and serial-parallel transfer mode translation on the DSP side. Registers are directly connected to the processing unit ports. This interface unit allows the integration of the IP core into the system design.

V. CONCLUSION

In this paper we presented a design methodology of IP integration in a SOC design that exploits both IP designer and SOC integrator constraints. The integration task is based on the IPERM and IP delay models that describe low-level details of the IP execution constraints. These models can be deliverable since the internal features of the IP core are hidden. We also presented a method for IP interface synthesis that can be easily automated. As a future work, we plan to refine the method by incorporating timing overhead added by bus wrapper, and handling the stochastic nature of applications where predicable behavior can not be guaranteed.

VI. REFERENCES

[1] Inventra, http://www.mentor.com/inventra/
[2] Virtual Socket Interface Alliance, http://www.vsi.org
[3] A. Cataldo, "VSI abandons plans for system-chip bus", *EETimes*,, 1997
[4] J.A. Rowson and A.L. Sangiovanni-Vincentelli, "Interface-Based Design", *in Proc. of DAC*, June 9-13 1997
[5] R. L. Lysescky, F. Vahid, T. D. Givargis, "Techniques for reducing Read Latency of Core Bus Wrapper ", *in Proc. of DATE*, March 2000
[6] K. Van Rompaey, D. Verkest, I. Bolsens, H. De Man, "CoWare A design environment for heterogeneous hw/sw systems", in Proc of EURODAC, 1996.
[7] Cadence VCC 2001 http://www.cadence.com/ datasheets/vcc.html.
[8] R. Passerone, J.A. Rowson, A. Sangiovanni-Vincentelli, "Automatic Synthesis of Interfaces between Incompatible Protocols", *Proc. of DAC*, 1998
[9] G. Cyr, G. Bois, M. Aboulhamid, "Synthesis of communication Interfaces for SOC using VSIA recommendation", i*n Proc. of DATE*, 2001
[10] A. Baganne, J-L. Philippe, E. Martin, "A Formal Technique For Hardware Interface Design", *in Proc of ISCAS*, 1997
[11] D. Gajski, N. Dutt, A. Wu, S. Lin, *"High-level synthesis Introduction to Chip and System Design"*, Kluwer Academic Publishers, Boston, 1992.
[12] J. Staunstrup, W. Wolf :"Hardware/software Co-design Principles and practice", Kluwer academic publishers 1997.
[13] M Keating, P Bricaud, *"Reuse Methodology Manual for System-On-A-Chip Designs"* Kluwer Academic Publishers, Boston, 1998

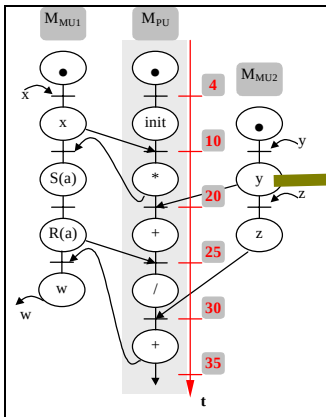

Fig. 9: IP Core Modeling

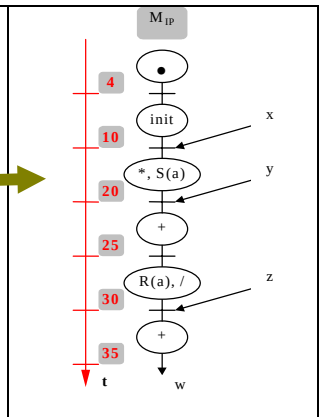

Fig. 10: 1st Phase Result

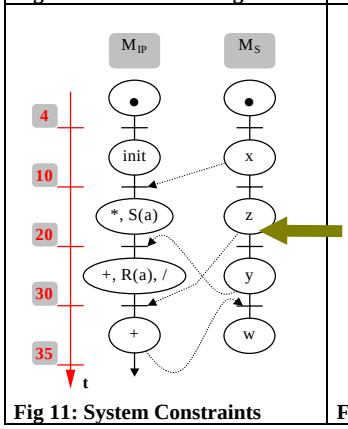

Fig 11: System Constraints

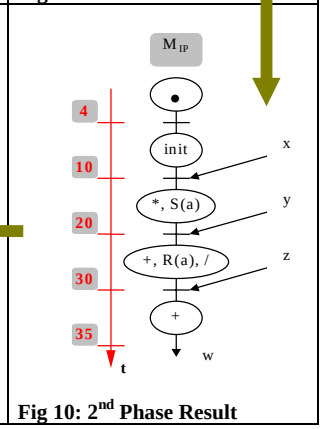

Fig 10: 2nd Phase Result