IP integration methodology for SoC design

F. Abbes[1,2] , E.Casseau[2] , M. Abid[1] , P. Coussy[2], J.B.Legoff[2]
1 GMS, ECOLE NATIONALE D'INGENIEURS DE SFAX, 3038 SFAX, TUNISIE
2 LESTER, UNIVERSITE DE BRETAGNE SUD, RUE SAINT MAUDE - 56100 LORIENT, FRANCE

Abstract

Integrating Intellectual Property (IP) components into System-on-Chip (SoC) designs requires the use of a generic parameterizable hardware/software interface to increase reuse efficiently, quality and productivity of SoC design. In this paper we propose a design approach for wrapping the cycle accurate bit accurate (CABA) interface of hardware IPs. This interface integrates many communication and synchronization mechanisms with respect to the virtual component interface VCI protocol from VSIA to fulfill IP designer and IP integrator requirements.

1. Introduction

In order to manage the system-on-a-chip (SoC) increasing complexity, a promising way consists of the reuse concept of preconceived hardware or software blocks, which are checked functions of the systems [5]. An important aspect of a core's marketability is its ability to be easily integrated into a SoC since IP must be usable in many different application contexts. Thus, the design team can concentrates the effort on the innovating parts of the system and exploits the already existing components.

The integration of the existing blocks, known as "IP" cores (VC: Virtual component in VSIA taxonomy), constitutes, actually, an essential phase of the SoC design. IPs facilitate and accelerate the design cycle. However, these components must be able to communicate with the system and between them. Generating automatically an IP wrapper –communication interface –is needed for a given black-box entity. To allow an easy interconnection of components from various sources, it is necessary that available virtual components respect a standardized communications protocol [11]. The high level abstraction of these components is the key of their flexibility and genericity, i.e. of their reuse opportunities and optimization. Currently, simulating a system at functional level does not guarantee an automatic generation of the associated RTL (Register Transfer Level) description. There are many manipulations made manually especially communication ones. In this work, we define the design technical details of a novel approach to ensure the encapsulation (the generic interfacing) of a RTL VC in a SoC. Few works have addressed the problem of IP integration and interface synthesis in a global way. This work focuses on the simulation of the interface before affecting architectural description for a generic SoCLiB IP interface.

The idea is to associate the behavioral model of the IP (under input/output (I/O) and timing constraints) to an I/O cycle accurate bit accurate (CABA) interface description model. It allows firstly simulating the IP in a plug and playing communication environment and secondly to easily synthesize it according to a generic interface architecture.

This paper is organized as follows. Section 2 analyzes the state of the art of the IP interfacing. Section 3 discusses structure of the generic interface for an IP design. Section 4 describes the SoCLiB platform, as a SoC simulation platform for the novel IP encapsulating approach and the technical details of the experimentation. Finally, we conclude this work.

2. IP Interfacing

In order to be reused in many different SoC contexts, IPs must be parametrizable, easily testable and especially interconnectable. Easy and quick assembling of various cores to obtain a fully functional SoC has not yet become reality. In fact, the integration of cores into a SoC is widely a manual and error-prone process because it forces the designers to fully understand the functionality and interfaces features of complex cores. Furthermore, a successful IP core integration requires the designer to take into account synchronization; protocol conversion and I/O buffer synthesis tasks [6]. Much work has been done on interfacing with cores basing on the key idea that explicitillyexplicitly separate the functional behavior of IP from interactions aspects between IPs. This separation allows (1) choosing the appropriate communications models for the IP design space exploration, (2) reusing the system behavior independently of the communication, and (3) reusing the existing communication models [7].

2.1. State of the art

Various techniques (bus synthesis, automatic synthesis of bus interface logic, rapid prototyping…etc) and standards aim at reducing system design time when designs become too large for the available software tools. They dealt recently with two approaches in basic design methodologies [8]. The first ones defines a standard bus protocol: for example using CoreConnect from IBM to connect various components to the bus architecture and a wrapper is indicated to adapt the protocol of each component to this CoreConnect. The second ones define standard components protocols that interface between a bus wrapper and the core internals [VCI [10], OCPIP [9]] and A core can be retargeted to any bus simply by providing an appropriate bus wrapper.

As communication interface is an intelligent block that can change the scheduling control and update parameters of modules in systems, it must work independently of the IP computation part. These components can be reused and arranged freely in a maximum of SoC context therefore the importance of standard component protocols. In this context, both VCI (virtual component interface) and OCP (On Chip

0-7803-8656-6/04/$20.00 ©2004 IEEE

Bus) initiatives allow that with simple connection management (one master and one slave unicast) as well as more complex connection (implying a master and several slave in broadcast or multicast) [11]. The goal of these proposal standards is to facilitate quick retargeting of a core from one system to another.

2.2. Proposed Interface model

IP design interface can be actually much longer than the internal IP design. Furthermore, communication problems and timing issues can cause failure of SoC design. That is why we consider that going from simulation down to synthesis is the best way to solve the communication problem of IP core reuse. The novelty of proposed approach of IP integration exploits both IP designer and SoC integrator constraints while considering the application requirements. The basic points are both specification of the I/O schedule of the IP and satisfaction of the user requirements. We aim having an IP's encapsulation model which communication interface may be predefined with user constraints.

Our approach deals with interfacing hardware IP to VCI. It starts with defining bandwidth, I/O timing constraints of the desired interface. Our approach manages communication under low level interfacing constraints details relative to the IP synthesized from a high level specification (computing latency, data synchronization transfer etc.). The main idea is to associate the behavioral of the IP model under I/O and timing constraints to an I/O CABA model. The RTL interface constraints are implemented at a cycle-true and a bit-accurate signal level. To automate the process of generating IPs communication interfaces, we formulate the problem under consideration as follows:

1- The configuration choice of the interface components leads to different communication and synchronization mechanisms: for a given IP, we focus on satisfying a variety of the pre-specified requirements of an application**.**
2- The parameters of the communication interface include clocking, address definition and VCI parameters.

The approach favorites the design at a higher level than RTL of HW interfaces and exploits both IP provider and SoC integrator constraints.

3. *CABA* Communication Interface

An IP can be viewed as a cycle accurate "black-box": only the communication interface is known. It has a number of I/O ports each of which having a certain bit width. SoC can supports multiple models of computation. In this work, we associate the IP behavioral model (under *I/O* and timing constraints) to an I/O *CABA* interface description model. The *CABA* abstraction level is defined so that having a *CABA* hardware interface without necessary describing the component hardware architecture [3]. This *CABA* definition specification tends to simulate faster than the *RTL* model due to less implementation details.

To connect the IP to others SoC components, we have to respect refinement and specification of the I/O protocols, data sequence orders and timing information of data transfers. The information needed is for identifying without ambiguity which data is input (res. output) at each clock cycle for each input port (res. output). The information is exploited for correctly interfacing the I/O of the IP with VCI protocol. *VCI* is a handshake data oriented protocol that allows memory accesses operations [2]. It defines a generic cycle-based address mapped point-to-point communication protocol with fixed initiator and target.

3.1. Interface communication mechanisms

Communication interface is composed of a software part and a hardware part [11]. For the reuse, only the software part changes: the hardware control is the same. The system data production (that can be the other SoC components production) does not usually correspond to the IP data consumption and inversely: the IP scheduling and I/O constraints is often different from the system scheduling and I/O constraints. We have to consider four sides according to the communication scheme (figure 1): the system sending data order and the IP consumption data order (for input controller) and the IP production data order and the system receiving data order (for output controller). In the other hand, the production of data can exceed momentarily its consumption by the IP. That is why the interface is based on the FIFO mechanism. This structure is considering in order managing the temporal difference and the memorization of the succession of the I/O variables. In fact, FIFOs store data, which are read in the same order in which they were written in. The FIFO size defines burst length tolerance.

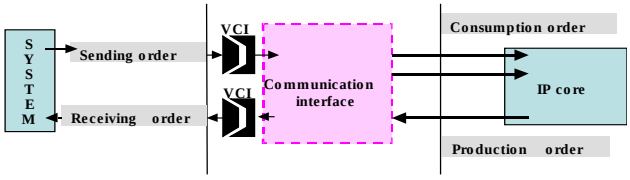

Figure 1. Communication scheme

Figure 2 shows the input interface structure with a 2 input ports case and 2 bytes data width. The output interface structure is a symmetrical. Input communication channel mechanism transforms the request from an external slave VCI port into a signal dedicated to the IP communication interface. Output interface mechanism translates the data back from the interface into a VCI response. The interface sub-modules follow the rules of the design reuse: they are instantiated according to the software script describing the spatial and temporal I/O constraints. When data are read or written, the address being accessed is checked to see if the desired data block is present in sequential FIFOs inserted between the VCI interface and the IP. The communication is made by passage of address and there is as many FIFO as address. The address of the data is used by the first input

controller (res. second output controller) to correctly de-multiplex (res. multiplex) data for the associated input (res. output) ports. The software script of the communication interface drives also memory mapping and data layout. It is writing to manage user requirements. However, the order in which data are sending from the system is the same as the one data are consuming by the data (res. the order in which data are producing is the same as the one data are recovering by the system). This restriction allows a generic form of the controller to be defined.

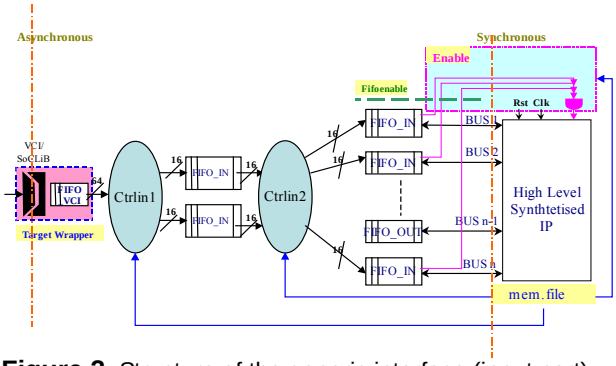

Figure 2. Structure of the generic interface (input part)

In the case of an IP that makes bi-directional buses, second part of the figure is considered. A second controller in the input interface is instantiated to dispatch data to the appropriate FIFO-bus. FIFO-bus is a process that synchronizes all FIFOs of the component interface with the *Enable* mechanism. Dispatching data in the appropriate FIFOs is carried out through the interface controllers detailed in the follow subsection.

3.2. Interface Controllers

The Sw corresponds to C code running on the processor and the Hw corresponds to the hardware accelerator. Interface controller is the Hw part of the interface. It handles the data flow/control according to the values specified via the IP interface. There are an input and an output controller. A single VCI is port as input (res. as output) for the interfacing mechanism. Target wrapper provides word of 8 bytes (size of the FIFO_VCI cell) as bit width size of the interconnect: 4 bytes for the address and 4 bytes for the corresponding information. As many IP architectures can consumes different bit-width data, optimising the bandwidth is useful for an efficient communication. For example, if the IP manipulate 16 bits as input and output, Ctrlin1 dispatches data in the appropriate FIFO_IN by address decoding and filters the data bits from the information part. In that case, it is assumed that two 16-bit data are transferred from the system each time. The information is stored into the interface input and output controllers in a set of configuration variables using information from mem.file. FSM Controllers are configured before data are transmitted. A sub-module *Enable* is responsible of the synchronization of the data traffics according to the I/O FSM required. *Enable* sub-module (figure 2) works with a counter. This counter is maintaining a virtual clock that cadences the IP behavior. When it is not set, IP behaves as if the clock is freezing. The referred virtual clock allows the controller to know the clocks cycles number executed in the IP. *Enable* is reset for new computation iteration. The iteration is a repetitive execution of a basic motif or data sequence. For example, with an FFT (Fast Fourier transform) IP, the treatment of a new matrix corresponds to a new iteration.

The *Enable FSM* shown in figure 3 synchronizes the data transfers between *FIFO_IN* (figure 2) and the IP according to the I/O constraints. It checks at least that a data is in each concerned *FIFO_IN* in the iteration step to stimulate IP behavior. The *INIT* state stimulate enable signal to launch the first iteration. *WAIT* state consists in switching off enable signal and to wait the arrival of data. Once this condition is filled, *RUN* state is active. *RUN* state transits the data to IP behavior and gets back output data from it. This FSM is piloted by I/O data scheduling. It drives synchronously the component with the implementation of the synchronous communication interface.

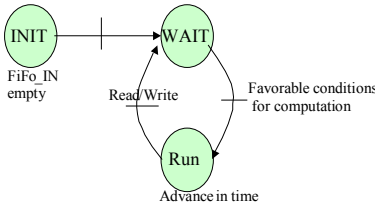

Figure 3. Automate Enable FSM

When the controllers receive the system-reset signal, the interface mechanism initializes the virtual clock counter and check that the FIFOs are empty (i.e. the IP consumption –resp. production – process is finished) to change the configuration of the interface behavior.

4. Design environment and experimentation

The communication interface design starts in the early phases of SoC design when the design specification is being developed. It uses a high-level language to speed to design flow: SystemC. SystemC [4] -a hardware modeling platform based on C++ language- can define both functional and timing characteristics to describe the interface. SystemC methodology relies on the speed and efficiency of the high-level languages to decrease verification through a combination of co-design techniques, co-simulation, and co-synthesis. Indeed, the use of a high level specification allows real reuse possibility for a particular function to be implemented and allows timing, memory and communication aspects to be taken into account.

The communication interface is specified as a slave basis SystemC Mealy/Moore Finite State Machine (*FSM*). Mealy Moore FSM contains:

1- A transition function, that computes the next state from the current state and the input values on the clock edge,

2- A generation function that computes the outputs as a function of the current state, for Moore machines, or as a function of the current state and the inputs, for Mealy machines.

Using this strategy is easy to apply with the concurrency nature of SystemC simulation. In this paper, we consider the particular aspects of hardware design with SystemC: high-level design using designs patterns, and generalization using templates. We present the design of a generic using the "state" design pattern and templates. The *sc_method* and *sc_thread* processes supported in SystemC help in modeling concurrency. The monitoring and tracing of the various ports and values is facilitated by *sc_trace* option. We use the C++ templates to implement variables that we keep parameterized for providing genericity. The bit width and the number of the I/O ports as interface parameters are template parameters. In the other hand, sizes of FIFOs, for example, are passed as SystemC parameters to the sc_module implementing the concerned sub-module.

In order to validate our approach, we have used the SoCLiB platform [3]. SoCLiB proposes the use of an open modeling and simulation platform for SoC design. It aims at developing a library of simulation models for virtual components (IP cores) to allow to model complex systems multiprocessors in the context of the VCI protocol use [1]. To test our proposed encapsulation methodology, a monoprocessor platform has been conceived from the existing SoCLiB IP cores. The interface specification is then used throughout the design flow to verify the design functionality at each step.

A slave generic communication interface has been evaluated with the hardware specification of a mean filter block. The platform contains a standard memory (RAM), a MIPS R3000 processor with its cache, and the Hw co-processor (mean filter). All these components are connected via VCI ports to a simple network (SoCLiB crossbar in figure 4). The processor initiates the communications. It sends/receives the data according to the software script scheduling. The co-processor reads the data according to the consumption order from the appropriate FIFO then writes back the result in the output interface.

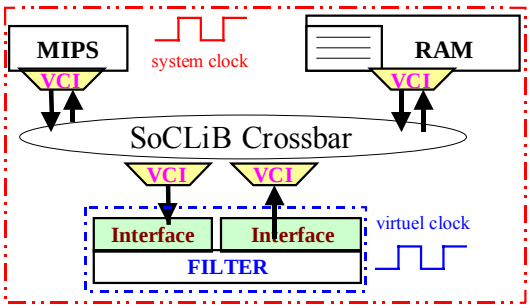

Figure 4. Platform design

The FIFOs are directly connected to the ports of the virtual component. The MIPS processor and the filter co-processor can then work concurrently in the platform thanks to the interface communication mechanisms. From the description files, a script is run with user parameters to obtain the desired specific interface. The overall Hw/Sw design is then co-simulated using the SoCLiB platform. Every transfer is done on a rising edge of the system clock. The transfers at the IP interface are cadenced by the virtual clock controlled by the Enable sub-module.

Simulations have been successfully realized with various numbers of busses (hardware IP input/output number) to verify different interface sub-module functionality. Simulations have been realized and validated under a Linux environment, SystemC-2.0.1 simulator and gcc 3.3.1 for compilation. The controller synthesis has also been realized with ISE/Foundation from Xilinx in order to evaluate the interface performance. For the mean filter constraints we used, ctrl1 and ctrl2 can operate up to 40 Mbits/s with a Virtex 1000E technology.

5. Conclusions

SoC design complexity can be managed by raising the abstraction level through the definition of new design methodologies for the reuse and the integration of the pre-designed cores. In this paper, the architecture of a communication interface that facilitates the integration of a hardware component in a system has been presented. This generic interface can be easily applied to different IPs constraints in different hardware system contexts and using different data protocols. The design has been successfully validated by simulation and its hardware synthesis is in progress.

6. References

[1] F. Petrot, P. Gomez « Lightweight implementation of the posix threads API for an on-chip MIPS multiprocessor with vci interconnect. » In: Design, Automation and Test in Europe Conference and Exhibition (DATE'03 Designers' Forum), Munich, Germany (2003)

[2] G. Cyr, G. Bois, M. Aboulhamid « Generation of processor interface for SoC using VSIA recommendations », submitted to IEE Proc. - Computers and Digital Techniques, 11/2002.

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

[4] http://www.systemc.org

[5] M. Keating, P. Bricaud, "Reuse Methodology Manual for System-on-a-Chip Design", 3rd edition, {Kluwer Academic Publishers}, 2003.

[6] P. Coussy, A. Baganne, E. Martin, « A Design Methodology for IP Integration », IEEE International Conference ISCAS 2002, Scottsdale, USA, May 2002.

[7] R.Damaševičius, V.Štuikys. «Wrapping of Soft IPs for Interface-based Design Using Heterogeneous Metaprogramming», informatica, 2003, Vol. 14, No. 1, pp. 3-18. ISSN 0868-4952

[8] R. Lysecky & F. Vahid « Prefetching for improved Bus Wrapper Performance in Cores » ACM-Transaction Design Autom. Electr. Syst. 7(1): 58-90, Janv. 2002

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

[10] Virtual Component Interface Standard VSI Alliance™, OCB 2 2.0, On-Chip Bus Development Working Group April 2001

[11] Y. Cho, G. Lee, Sungjoo Yoo, K. Choi, and N.Zergainoh, « Scheduling and Timing Analysis of HW/SW On-Chip Communication in MP SoC Design » (DATE'03 Designers' Forum), Munich, Germany (2003)

< Back