INTERFACE DESIGN APPROACH FOR SYSTEM ON CHIP BASED ON CONFIGURATION

Issam MAALEJ[1-2], Guy GOGNIAT[2], Mohamed ABID[1], Jean Luc PHILIPPE[2]*

[1] E N I S - DÉPARTEMENT DE GÉNIE ÉLECTRIQUE, *B.P : W 3038* SFAX TUNISIE

[2]L.E.S.T.E.R., UNIVERSITÉ DE BRETAGNE SUD, FRANCE ; RUE SAINT MAUDE - 56100 LORIENT

* Email: *maalej@ iuplo.univ-ubs.fr*

ABSTRACT

Communication synthesis is an essential step in hardware/software co-synthesis: many embedded systems use automatic generation of interface for point to point communication or use external supports of communication as standard bus or micro network. In this paper, we address the problem of hardware – software interface design in codesign approach for real-time applications. We refer to the hardware component as hardware accelerator and the software component as processor. In our approach, the accelerators are directly integrated into the processor core in order to optimize the transfers of data between the two units. In this paper we describe a configuration approach to design a communication interface.

1. INTRODUCTION

Telecommunication and multimedia computing are the fastest growing segments of microelectronics market today. The market sectors are stimulated by emerging business and consumer application that are now possible with recent advances in wireless communication, multimedia processing, and integrated network technology. In fact, thanks to the micro-electronic advances in integration and the progress made by silicon manufactures, implementation of embedded systems is observing a quick shifting from system on board to System on Chip (SoC) where all components are on the same silicon. Thus, hardware/software SoC codesign is one of the most important electronic application area in future-oriented designs.

The complexity of these embedded systems and their time to market constraint pressure designers to leverage the reuse of both software and hardware modules. These modules are called VC (Virtual Component) or IP (Intellectual Property). These components represent function of specific domain like signal processing (DCT, FFT), telecommunication and multimedia (VLC, Turbo codes) etc. In this area, IP integration and management require new concepts innovative breakthroughs in order to introduce new quality in Electronic Design Automation (EDA).

In the work presented in this paper, we focalize on the problem of communication design for SoC. In fact, communication synthesis is a key step in the integration of embedded system. Interfacing between hardware and software is the bottleneck in many embedded systems, since communication links add chip costs and timing overhead.

To enable flexible low-cost designs in a short design cycle, some emerging designs are based on heterogeneous embedded system architecture that integrate multiple software programmable components, e.g. processor cores, together with dedicated hardware components into a single chip. Programmability is then introduced in these embedded system architectures (offering more flexibility to the designer), while maintaining most of the advantages of customized VLSI and ASIP solutions (such as the potential to optimize processing performance and reduce time to market).

In this paper, we address the problem of automatically obtaining a customized interface between processor and hardware accelerator modules. We present an original approach based on configuration of generic model to design communication interface for SoC.

The paper is organized as follows. Section 2 gives a brief description of previous and related work. Section 3 describes the target architecture and formulates the problem that we consider. Section 4 presents our approach of communication synthesis. Some results concerning the communication between a processor core and some accelerators are presented in section 5. LEON processor core using AMBA bus is considered for this study. Finally, section 6 concludes the paper and presents some lines of future work.

2. RELATED WORK

Different approaches attempt to ease and quick assembling SoC. Some of the designers committee agree that the key approach to enhance the reuse and integration of IPs into a wide range of applications is the one separating communication from behavior. This model called *interface-based design* [1] has two main advantages. The first is that VC creators can easily update their products to customer's architecture by limiting the modifications to IP interfaces, while keeping the IP cores untouched. The second advantage is to promote the definition and the development of standards and design methodologies for IP interfaces. Today, these concerns are being addressed by many industries as SoCiations like Virtual SoCket Initiative Alliance (VSIA).

Some designers treat the problem during high level synthesis thanks to tools or languages of specification like Cosmos [2], Polis [3], SystemC [4] and Coware [5], which allow automatic generation of communication interfaces. [6] treats communication synthesis by an approach based on synthesis of the communication structures (interfaces, arbitration schemes). Others focalize in the automatic generation of interface to point-to-point communication [7] [8]. [9] addresses the problem between standard components that have incompatible protocols. These approaches require having a perfect knowledge of the

protocols of both the sender and the receiver. Furthermore, they represent a dedicated approach: for each new application it is entirely necessary to redesign the interface.

Other designers propose the use of external communication resources as standard bus (e.g. AMBA) or micro network architecture (e.g. Sonics). In this case it is necessary to define wrappers in order to ensure the communication between the hardware accelerators and the bus. The problem that tries to resolve these approaches is the communication with the bus on one-side and with the IP on the other side.

In our approach, we propose a methodology to help the designer during the communication interface definition step. Our method is based on a separation between the communication and the behavior which:

- Do not require detailed knowledge about communication protocol,
- Uses a generic model of communication,
- And, conducts to an efficient implementation of the communication interface in term of time and area.

3. TARGET ARCHITECTURE

In this section, we present a typical SoC architecture and different aspects related to communication problems.

Architecture of embedded system includes generally many types of components as processor cores hardware accelerators and software modules. For these systems, designer can target three types of processors according to constraints on the system: processor specific to an application as ASIP, processor specific to a domain as DSP or VSP (Digital/Video Signal Processor) and processor of a general nature as RISC processor (Reduced Instruction Set Computer). The first and the second types present better performances and flexibility but they require a significant cost of design. Contrary, the last type has the most short design time and presents an acceptable performance. However, in order to meet tight constraints in numerous specific applications it is generally necessary to add some hardware components to the RISC processor cores.

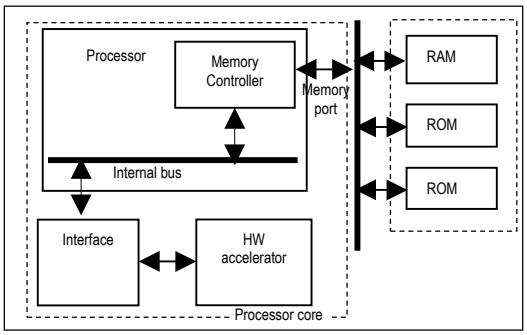

Figure 1: SoC architecture

Target architecture considered in this work is depicted in figure 1. Such architecture includes hardware accelerators which represent computing units that have not enough control logic to be autonomous. These accelerators need external control unit to manage the communication transfers. Thus, they are connected to the internal bus of the processor through a communication interface and are considered as slave while processor core is the master of the system.

Using this architecture, we consider the following assumptions:

- The processor commands transfer operations, since it is the master of the system and all hardware accelerators are slaves,
- Hardware accelerators do not have their own communication logic, they need a communication interface,
- Hardware accelerators are connected to the on core processor bus in order to enhance the communication performances between the processor and accelerators,
- Processor data size may be different from hardware accelerators data size.

4. COMMUNICATION SYNTHESIS

Considering the architecture and the assumptions described before, we propose a communication interface design methodology based on a configuration approach. This approach deals with a generic model of communication interface. The designer configures the communication interface considering the performances and constraints it has to meet.

The objective of our approach is to allow the interface design for SoC implementation without requiring detailed knowledge about communication implementation and generation of optimized communication (performances, constraints, cost etc.).

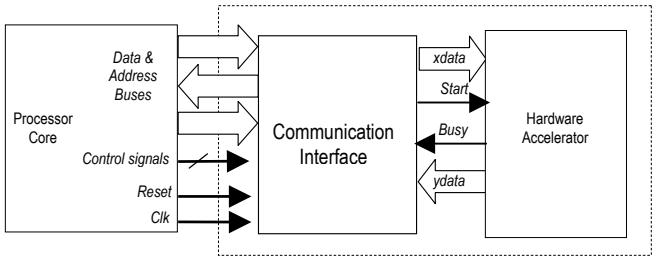

Figure 2: Interface entity

The interface model is presented in figure 2 and has two sides: the first side communicates with processor core, the second communicates with accelerator. Side1 is connected to the on core processor bus that has its own protocol. In this work we have focused on AMBA AHB and APB buses. Side2 is connected to hardware accelerator that can be classed in two types. The first is characterized by a constant computing time, no large memorization (only few registers), a fine granularity, and a constant input/output data size. This type is called FGDTA (Fine Granularity & Deterministic Time Accelerator). Typically, this type of accelerator is DCT, Exponent, FIR … The second type is called VGNDTA (Variable Granularity & Non Deterministic Time Accelerator) and considers others accelerators as variable input/output data size, non deterministic computation time, variable granularity, etc. Note that both types can have different data sizes than the processor data size.

4.1 Communication with the bus

To design a generic communication interface, we consider a generic communication chronogram of simple transfers as shown

in figure 3. For this work, AMBA AHB and APB buses are considered [10].

This chronogram is composed of two clock cycles. On the first clock cycle, address and control signals generated on the processor bus and read by the interface module. On the second clock cycle, data is transferred from/to processor to/from interface. To allow a generic protocol, the transfer is performed only on the second half of the second clock cycles since the majority of existing buses correspond to this model. For example, the APB bus has such a simple transfer, while AHB uses specific protocol as burst protocol. However, the burst protocol is a duplication of two simple transfers where the second transfer begins on the second cycle of the first transfer.

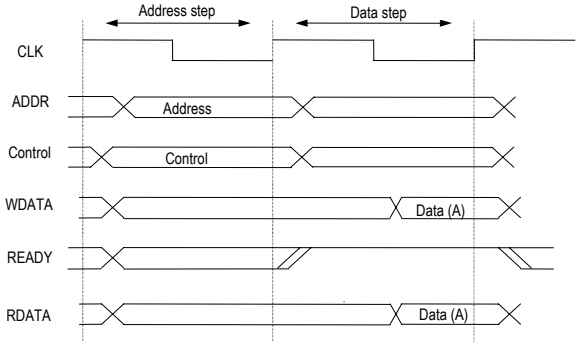

Figure 3: generic transfer chronogram

In order to build the interface control unit is able to communicate with different protocols, the considered protocol model (i.e. shown in figure 3) is designed with parallel units in order to allow the duplication between the address and the data steps.

4.2 FGDTA interface

We consider here FGDTA accelerators where communication can be performed with a simple transfer mechanism. If the target accelerator has the same data size than the processor core, only the selection of the accelerator and the transfer direction are controlled. More sophisticated interface module is used only when data sizes are different. Two cases may be presented: processor data size (*ProcDS*) is larger then accelerator data size (*AccDS*), or *AccDS* is larger then *ProcDS*.

i) If *ProcDS* > *AccDS*:

a. The communication interface detects the selection of the accelerator and begins the transfer,

b. The communication interface reads data from the processor core until the totality of data has been transferred inside the interface,

c. When the accelerator is ready, the interface transfers data to the accelerator and sends a start impulsion to command the beginning of the hardware accelerator computation,

d. When the accelerator has finished data computation the interface reads the result and sends it to the processor core.

ii) If *AccDS* > *ProcDS*:

a. The communication interface reads data from the processor,

b. The communication interface saves data in registers,

c. The communication interface sends the accelerator data with a start impulsion,

d. The communication interface reads the result from the accelerator when the *busy* signal is set to '0',

e. The communication interface transfers the result to the processor.

4.3 FGDTA Interface specification

The FGDTA interface is described in VHDL language and is performed in two files. The first is a package that describes the data types necessary for the interface. It contains four types: one describing the size of the target processor core internal buses, another describing the size of the hardware accelerator data buses, and two others describing the control signals (input and output) associated to the processor bus. The second file constitutes the functional description of the interface. It corresponds to the control unit that performs the communication management. 5 processes grouped in one «process», which is sensitive to the clock signal «clk», ensure the communication. Each process is controlled by specific conditions to its operation. The first process corresponds to the procedure of launching of the transfer of the data. It detects the beginning of the transfer operation when all the operating conditions are validated. This process has as a function to validate the transfer operations in writing and reading. These two operations constitute successively the second and the third process. The operating conditions of these processes are ordered by the first, the fourth and the fifth processes that correspond successively to the processes of reset of writing and of reading.

4.4 Interface FGDTA configuration

The FGDTA communication interface represents a generic module that can be connected with various types of buses and different accelerators. The interface is able to ensure the transfer of data between processor and accelerator having different bus sizes. This propriety is performed in the code description-using notion of package in the VHDL language. Using this approach, the designer does not have to use an automatic interface generation but has to configure the communication interface. Hence, the communication interface definition is summarized in the three following steps:

1. Configure the data types of the buses in the package,
2. Configure the signals of the buses in the package,
3. Update the code description.

This approach has the advantage of making possible the communication without detailed knowledge of protocols and sequencing of the tasks.

5. EXPERIMENTAL RESULTS

The communication interface module is described using VHDL and the generic code has been optimized in order to conduct to an efficient implementation in term of area and frequency. Note that the processor and the hardware accelerators can work with different frequencies. Several examples were done, in the following we present the communication interface between DCT (Discrete Cosines Transform) as hardware accelerator and LEON [10] as processor core.

LEON processor has two internal buses AMBA AHB and AMBA APB. In order to validate the generic connection with buses, the FGDTA interface has been used to communicate between these two buses and the DCT accelerator. The DCT uses 64 bits as inputs/outputs and the two LEON buses are 32 bit widths. Some results about the experiments of the communication interface (throughput and interface area) are shown in table 1.

Bus type	Throughput (Moctet/s)	Interface area (LUT)
AHB	88	196
APB	36	196

Table 1: Bus communication performance

These results confirm that the AHB bus is more adapted for high performance communication. These experiments were conducted with a Virtex FPGA from Xilinx as target technology. The FPGA has a maximum frequency of 100Mhz. The DCT and the communication interface are implemented on the processor core. After synthesis, as shown in table 2 the interface designed uses a minimum area and the frequency of the accelerators is maximal.

	Area (% of total area)	Frequency (Mhz)
Interface	1	100
LEON processor	28	35
DCT	6	100

Table 2: Experimental results (Virtex XCV800)

Hence, the utilization of the generic interface conducts to a negligible area overhead and ensures a high communication speed.
Similar results were obtained using the interface to communicate between APB and PCI buses (also exist on LEON processor) with the same DCT. These buses have also 32 bits as data bus size but have different protocol complexity. This diversity confirms the reliability of the interface module designed.
In order to evaluate the VGNDTA interface, some tests were also done with VLC (Variable Length Coder) and DCT considering different granularities. The VLC presents a variable granularity and non-deterministic properties while the DCT presents a variable granularity but a deterministic computation time (depending of the considered granularity). Experiments with the DCT are presented here and allow studying the impact of the granularity on system costs as area and computing time (figure 5). If the granularity increases the computing time decreases; however, the area cost increases. To help designers to make a choice about the granularity degree figure 5 gives a tradeoff curve between speed and area.

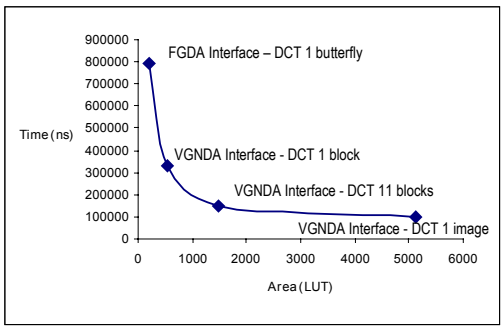

Figure 5: Granularity impact for a DCT accelerator

6. CONCLUSION

This paper proposes a configurable generic interface model allowing the refinement of the communication between processor core and various hardware accelerators. This approach allows the designer to simplify the communication integration step. This model is sufficiently generic in order to be able to adapt to various bus protocols and accelerators data sizes. The experiment results show that this method is efficient for hardware accelerators. Generic chronogram will be generated in future work to have a generic model for more standard bus. Our work will be extended to be more generic for more bus and accelerator type.

7. REFERENCES

[1] J.A. Rowson and A.L. Sangiovanni-Vincentelli, *Interface-Based Design*, *in Proc. of DAC*, June 9-13 1997

[2] Benner T., ENST R., *An Approach to Mixed System Co-Synthesis*, proc. INT'1 Workshop on Hardware/Software Co-design, Braunschweig, Germany, Mars 1997.

[3] Balarin F., Chiodo M., Lavagno L., Passerone C., Sangiovanni-Vincentelli A., Sentovich E., Suziki K., Tabbra B., *Hardware-Software Co-Design of embedded system : The Polis Approach*, Kluwer 1997.

[4] Kjetil Svarstad, Gabriela Nicolescu, Ahmed A. Jerraya *A model for describing communication between aggregate objects in the specification and design of embedded systems*, DATE 2001.

[5] Vercauteren S., *Hardware/Software co-design of application-spécific Heteregeneous architectures*, IMEC, PhD thesis, Dec. 1998.

[6] A. Zitouni, M. Abid, K. Torki, R. Tourki, *Communication synthesis techniques for multiprocessor systems*, International Journal of Electronic 2002, VOL. 89, NO. 1, 55-76.

[7] Bill Lin, Vercauteren Steven, *Hardware/Software Communication and system Integration For Embedded Archtectures*.

[8] Narayan S., Gajski D., *Interfacing Incompatible Protocols using Interface Process Generation*. Proc. Design Automation Conference. San Francisco, CA, USA, June, 1995.

[9] R. Passerone, J.A. Rowson, A. Sangiovanni-Vincentelli, *Automatic Synthesis of Interfaces between Incompatible Protocols*, *Proc. of DAC,* 1998

[10] www.gaisler.com