

Bus-centric SoC Architecture Generation Tools

Jordi Carrabina
CEPHIS. UAB
Dpt. Informàtica. ETSE.
08193 Bellaterra. Spain
+34935813082
jordi.carrabina@uab.es

Màrius Montón
CEPHIS. UAB
Dpt. Informàtica. ETSE.
08193 Bellaterra. Spain
+34935813534
marius.monton@uab.es

Ricardo Martinez
IMB-CNM (CSIC)
Campus UAB
08193 Bellaterra. Spain
+34935947700
ricardo.martinez@cnm.es

Jaime Joven
CEPHIS. UAB
Dpt. Informàtica. ETSE.
08193 Bellaterra. Spain
+34935813082
jaime.joven@campus.uab.es

Oriol
CEPHIS. UAB
Dpt. Informàtica. ETSE.
08193 Bellaterra. Spain
+34935813534
ioseporiol.font@campus.uab.es

Raul Ruiz
EPSON Electronic Europe GmbH
c/Alcalde Barnils, 64-68, C-2
08190 Sant Cugat, Spain
+34.935442490
raul.ruiz@epson-electronics.de

Pere Garcia
EPSON Electronic Europe GmbH
c/Alcalde Barnils, 64-68, C-2
08190 Sant Cugat, Spain
+34.935442656
pere.garcia@epson-electronics.de

Lluis Teres
IMB-CNM (CSIC)
Campus UAB
08193 Bellaterra. Spain
+34935947700
lluis.teres@cnm.es

Abstract — System-on-a-chip (SoC) architectures are driving an important part of the Digital Signal Processing technology. System architects need new methodologies oriented to efficiently build different SoC systems oriented to different target applications. This paper presents a methodology oriented to real and virtual platforms compatible with system-level design tools and including IP integration. This methodology is supported by a new set of Java tools, that generate the Hardware (Verilog™) and Software (C) files required to synthesize and simulate an entire AMBA™-based SoC. This methodology has been validated for a set of examples using simulation tools and a prototyping board.

I. INTRODUCTION

Current silicon technologies integrate complete systems on a single chip, introducing the Systems-on-a-chip (SoC) architecture concept. This promotes the use of system level design methodologies that receive strong support from new languages (SystemC, system verilog, JHDL,...) and tools (Cocentric, Cossap, SPW, Matlab, ConvergencSC, ...) developments.

SoC architectures were initially proposed as a central processor, an on-chip bus (OCB), standard components like memory, peripherals, interrupt units, etc. plus some application specific components. Among the advantages of this approach we have power savings, higher integration density, lower systems costs, easier procurement, etc

Among the different approaches to the SoC concept, Virtual socket initiative (VSIA™) [1], merits special remark for the effect in standardizing criteria for concepts, methods and allowed interoperability.

The most popular SoC approach has been the ARM™ processor strategy [2]. ARM™ disposes of a complete family of RICS processors, with the AMBA™ OCB[3].

AMBA™ (Advanced Microcontroller Bus Architecture) is currently one of the most widely used systems bus architectures for SoC applications (even for processors other than ARM™).

SoC approach together with component reuse, through virtual components (IPs) management, coming from different design teams (either in-house or third party), is greatly increasing the design productivity and therefore pushing the creation of new low-power high-performance applications.

This works starts from the SoC concept and tries to establish a design methodology for building SoC architectures embedding IPs, for bus-centric systems.

II. SOC PLATFORMS

SoC concept requires support at levels of abstraction in order to map system level specifications into chips, either ASICs, ASSPs, ASIPs o CPLDs...

A. SoC Architecture

Computer architecture has been for years constrained by manufacturability restrictions (i.e. pin count of ICs and PCBs). This lets to structural techniques inherent to architectures (i.e. 3-state buffers to manage buses). Silicon integration evolution

together with increasing number of metal layers let to architectures without these restrictions, producing new criteria for designing systems (separation of address and data buses, format of control signals, clock and reset management, etc.) most of them included in the VSIA recommendations.

One of the consequences is that SoCs have multiplexed OCBs, so that every shared signal is routed from masters to slaves with help of multiplexers and decoders. These Bus Modules must be customized in order to take into account the number of master and slave modules, and their base addresses. Like in more complex computer systems, a SoC can implement more than one bus, being the typical architecture composed of one high-performance bus for memory access and high speed peripherals and one low-speed bus for low-speed peripherals like UARTs.

B. SoC tools

New tools are emerging in the market to help SoC design. System Express [4], DesignWare & System Studio [5] and Platform creation [6] provide the designer the capability to easily create SoC architectures, allowing faster design explorations.

These tools enable designers to specify a given SoC architecture using a GUI (Graphical User Interface) and Drag&Drop technologies.

Proprietary IPs can have a huge integration, since they required new models (sometimes as complex as a SystemCTM models) or special wrappers.

When the whole design is specified, designers can verify its functionality (using simulation), and, if specs are not reached, modify the architecture and iterate the process.

Some of these tools work at SystemC level, with cycle accurate simulation but do not generate any HDL file for final implementation. Some other tools can generate HDL files and these files can be used to simulate entire system, but they do not use high level language executable specifications.

C: AMBATM based SoC

The AMBATM bus has been widely used in SoC development. Three different buses are defined within the AMBATM specification:

- 1) Advanced High-performance Bus (AHBTM).
- 2) Advanced System Bus (ASBTM).
- 3) Advanced Peripheral Bus (APBTM).

AHBTM bus is devoted to high-performance communication, for system modules requiring high clock frequencies. This bus acts as the backbone bus. It is intended for connection of processors and co-processors, DSP units, on-chip memories and off-chip external memory interfaces.

The ASBTM is mainly deprecated and it has been substituted by AHBTM. The APBTM bus is devoted to low-speed peripherals and it is optimized to minimize power consumption and to reduce interface complexity.

APBTM is designed to be used in conjunction with a system bus (AHBTM/ASBTM).

D. Processor IPs and OCBs

There are different approaches for building SoCs according to given application purposes. Our main interest is to help system specification in order to quickly build any used defined architecture. This will help architectural exploration and verification either at simulation levels (functional or structural) or through rapid prototyping. Due to the fact that we reaching prototyping platforms, we will validate our approach with examples mapping to ARMTM-FPGA solutions.

Among those solutions we analyzed several approaches:

1) Altera SOPC [7]

The NiosTM embedded processor is a user-configurable, general-purpose RISC embedded processor. It was designed to be a flexible and powerful processor solution. NiosTM is a Soft-core (or Soft Ips), what means that RTL code is generated according to user specifications. Designer can choose data path width (16 or 32 bits), multiplier type (software based, one-cycle, n-cycles), cache size, memory map, interrupt priorities, etc.

Altera offers another SoC solution, based on the ARMTM processor. This system is not as flexible as Nios systems from the architectural point of view, but it is more powerful. ARMTM, Interrupt Controller and single and dual-port RAM are implemented as Hard-cores (or Hard IPs), so user cannot choose all options available on Nios systems.

For both systems Altera brings also complementary IPs (either proprietary or third party) ready to be used.

2) Xilinx [8]

MicroBlazeTM is the 32-bit soft processor from Xilinx. It is a standard RISC-based engine with a 32 register of 32 bits each, LUT RAM-based Register File, with separate instructions for data and memory access.

Smaller and less performing, PicoBlazeTM and its derivatives are fully customizable 8-bit soft microcontroller macros that provide 49 different 16- to 18-bit instructions, 8 to 32 general-purpose 8-bit registers, 256 directly and indirectly addressable ports, reset, and a maskable interrupt.

Xilinx disposes also of a solution with the IBM PowerPCTM 405 core integrated into Virtex-II Pro devices using the IP-Immersion architecture, that allows hard IP cores to be diffused at any location deep inside the FPGA fabric [9].

E. Prototyping Systems

To validate and test the entire system, we use a specific prototyping board. In this prototyping system, there is an ARM720TTM CPU connected to a Xilinx Virtex 2-6000 FPGA, SDRAM and FLASH and a set of ICs for physical interfaces to Ethernet, USB and

PCMCIA/CF. The entire SoC design is built inside the FPGA, and any bus configuration is possible.

We propose a SoC design methodology able to support this flexibility, with the help of our own tool, **UltraWizard**, oriented to easily build SoC architectures.



Fig. 1. ARM + FPGA Prototyping platform

This platform is configured, through a process in which the ARM™ program memories are filled downloading the code from a PC to SDRAM through a UART running in special mode. When then a reset sequence is generated the board allows all download-related peripherals running in normal mode of operation.

FD. Hierarchical Bus-centric SoCs

As an extension concerning traditional SoC architectures, hierarchical bus-based systems stand for systems with no limitations about number of buses and its hierarchy. These systems are more flexible and powerful than processor-centric systems (they allow any number of CPUs). To achieve that goal, we need to deal with any kind of bridges between buses (AHB-AHB, AHB-APB, APB-APB).

III. SPECIFICATION OF ELECTRONIC CIRCUITS USING XML™

During last years, XML™ has been proposed and progressively used for electronic design specification, becoming a de-facto standard for interoperability in IP management [10], [11], [12], [13].

XML is a mark-up language for documents containing structured information [14]. Structured information contains both content (words, pictures, and in our case the more relevant characteristics of hardware IPs of our system) and some indication of the role that those contents play (we use this indications to save the attributes we need for each characteristic of our Hardware IPs). Data can be identified using tags.

A mark-up language is a mechanism to identify structures in a document, whereas XML specification

describes a standard way to add mark-up to documents.

XML seems to be a good choice because it is a very easy language to describe all kind of information and, because of its flexibility, allows building the right architecture to describe our hardware IPs and the top view of our SoC (it is a hierarchical language). These descriptions will be easily created, modified and parsed.

Also, all the documents must fulfill all the restrictions described in the specification of the document. This specification is known as Document Type Definition (DTD) that can be viewed as a dictionary for the document (also described in XML). It specifies the possible elements, attributes, entities, its properties and relations between them.

A. IP specification

In order to describe an IP, it is necessary to specify it initially as a black-box, so without any functional or structural description. In our proposal, XML files describing components are divided in two parts, one for HW information and one for SW information.

This means that the XML file corresponding to a given IP describes following items:

1) HW information :

- Ports: name, direction (input, output or bidirectional), width and type (bus function, irq, or extern).
- Bus type that IP is attached to (AHB or APB).
- Behavior: Master or Slave.
- IRQ related info: enabled or not, default assignment.
- Address base and depth.
- Files needed to synthesize an IP.

2) SW information

- Name of C structures.
- Name of C header files.
- Libraries directory.

B. System specification

To describe the entire SoC, we propose to use also a XML description. In this case, the XML file contains only buses, and references to IPs connected to them.

At this level, concept of connection is introduced. With one connection, the designer can interconnect two IPs (i.e. UART_TxReady pin to DMA_request pin).

The designer can even interconnect two buses (in this case the corresponding bridge is created) and connect an IP to another bus (in order to be visible in both buses).

It is important to note that XML system specification contains only IPs and buses, avoiding explicit declaration of other necessary components of AMBA buses (Muxes, Arbiters, Decoder). These components are automatically generated by our system level tool: UltraWizard.

C. Software Technology: JAVA™

Java technology includes both a programming language and a platform. Advantages of Java include

its portability, so using it as the programming language for our tools and methodology, will not determine the operating system it should run on.

Another advantage of Java is that it offers a whole set of facilities to parse XML documents giving a very simple interface to work with them.

IV. ULTRAWIZARD

An essential part of our methodology is composed of a tool that build the SoC architecture corresponding to the user specification.

UltraWizard (UW) is composed of two pieces, front-end, to interact with the designer with a GUI, and back-end to generate the complete system. The GUI allows the designer to graphically build the entire system. Once done, front-end generates the XML file containing all the system information stored on it (system.xml). With this information, back-end tools generate all files needed for synthesis and simulation.

With the GUI, the designer can describe the system to be implemented, selecting IPs from a catalog, and the way to connect them (assign IRQs, base address, etc).

As any other graphical interface with the designer, front-end UW manages the verification of correctness of the system description. (Unique IRQ, address range conflict ...).

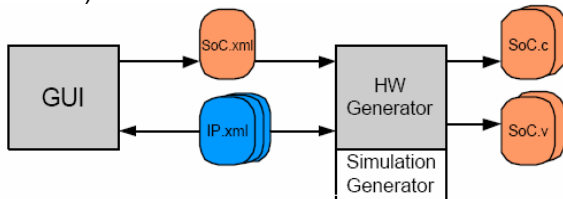


Fig. 2. UltraWizard Structure.

We call UW back-end, the code that generates the whole set of files starting from the information generated by front-end and designer. This information is archived in another XML file.

Several files are created at this stage that build the system, specified by system XML file:

- 1) HDL files (Verilog) to describe complete system. By creating bus interconnection modules required for all buses present in the system, instantiating all modules and properly connecting them.
- 2) C files to give a set of Software libraries in order to access registers of each module. At this time Back-end creates a single header file (system.h) that contains the definition of the set of pointers to each module or structure that has registers visible by the processor.

The structure used for our systems is the following:

- 1) One XML file for each hardware IP.
- 2) One XML file for the system.
- 3) A DTD file defining all information on each module in the system.
- 4) A DTD file defining all information on our system.

SoCs built following our methodology and tools, will be composed of several interconnected buses and, at

last in one of them, there will be a master module. The simplest example we can think about, is composed of one AHB with an ARM processor and a memory controller.

We allow our systems to grow in complexity in order to produce more complex bus structures, since UW was built trying to be as general and flexible as possible.

The following classes represent the system:

- 1) System: Stores the whole set of buses and connections in system.
- 2) AMBABus: Represents one AMBA bus (AHB or APB) and all IPs connected to it.
- 3) HWModule: Stores information for every IP.
- 4) Connected: Represents interconnection between IPs and/or buses.

For the SoC automatic generation, we implemented the following methods:

- 1) Generate_from_system: Writes the top HDL file.
- 2) Generate_from_top: Generates HDL files involved in AMBA bus (arbiter, decoders, MUXs).
- 3) Generate_from_template : Used by previous classes to generate HDL from template files.

D. Templates

Verilog files required to synthesize the SoC system are created using templates. Templates must follow the AMBA architecture specification again. The main goal for these templates is to be generic (they should work in any system that fulfills the restrictions of an AMBA system). This means enabling any number of buses (AHB or APB), any allowed interconnection among them and any number of modules attached to them. These templates must be oriented to build Verilog HDL descriptions (with a certain structure and respecting some syntax rules).

Templates contain Static Verilog sentences with tags for the parts that we want to generate automatically. Those tags will be substituted (if required) by specific IP ports that allow correct port-mapping when instantiating IPs in the top system.

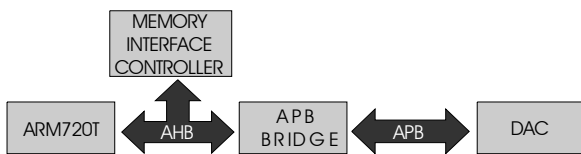
V. EXAMPLE

In order to show the capabilities of this new methodology, we use a simple problem for an easy understanding of UW power.

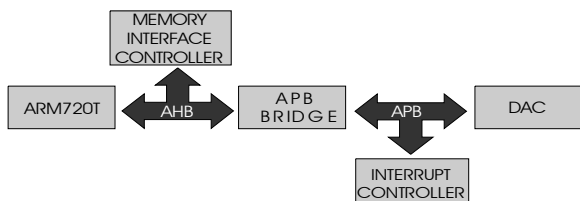
The example consist in the evaluation, in terms of bus occupation and HW resources usage, of three different architectures designed to map a system able to generate a sinusoidal waveform through a DAC (20 KHz), connected to the APB bus.

The proposed solutions are:

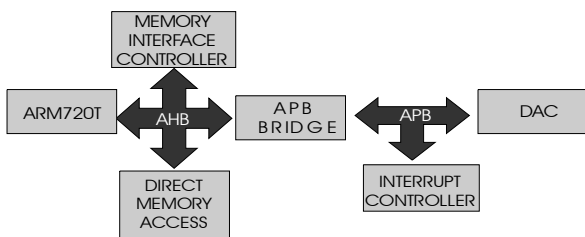
- 1) "Stand-alone DAC". The SW will poll the DAC status. When ready, data is sent to the processor. Obviously, we foresee a high bus occupation with minimum amount of HW resources.



2) For the second approach, an interrupt controller is added. The bus is used only in the case that an interruption were generated by the DAC. This should end up in a higher HW resources usage but decreasing the number of bus cycles needed.



3) Direct Memory Access scheme is added in the third case. In order to optimize DMA capabilities some buffering has to be added to the DAC, i.e. FIFO memory. The size of the FIFO is now another parameter that can be evaluated.



The three different approaches have been generated by Ultrawizard. The obtained RTL code was synthesized using Leonardo Spectrum™, simulated using VerilogXL™, and downloaded to the ANDIII board, with the following results:

Architecture Type	HW Resources (Xilinx CLBs)	Bus Occupation (%)	Processor Occupation (%)
Polling (1 st)	1103	100% ¹	100% ¹
Interrupt (2 nd)	1341	7%	7%
DMA (3 rd)	2236	5%	3%

This table shows the usefulness of Ultrawizard. A quick evaluation of different HW-SW solutions, corresponding to different SoC architectures, can be carried out in a short period of time, using either simulation or downloading the HW and SW compiled codes into the Andill board.

In this way, the system architect disposes of performance measurement for different architectural

¹ This percentage will depend on SW

choices that solve a given system application. As a consequence, it is easier to find the optimal implementation that fits with specifications in early stages of the design cycle.

VI. CONCLUSIONS

We believe that the design of complex SoCs has to analyze a large and diverse set of architectures in order to find optimal implementations. Architectural exploration have to deal with restrictions coming from application domains and performance requirements.

Moreover, application domain drives the use of 3rd party IPs for which a restricted set of models is available, so that performance measurements becomes more complex.

Ultrawizard is oriented to help designers in both aspects for bus-centric SoC architectures based on the AMBA bus. IP management is supported through XML (de-facto standard) specification.

Production of cost measurements for the architectural exploration phase is accelerated though its capabilities for building complete systems with complex bus architectures and its link to both simulation and prototyping facilities.

Copyrights:

VSIA™ is a trademark of Virtual socket Interface Alliance; ARM™, AMBA™, AHB™, ASB™, APB™ are trademarks of ARM Ltd; SystemC is a trademark of the Open SystemC Initiative; NIOS Nios is a trademark of Altera Corp.; MicroBlaze, PicoBlaze, Virtex are trademarks of Xilinx, Inc; PowerPC is a trademark of IBM Corp; Java is a Trademark of Sun Microsystems, Inc; XML is a trademark of MIT and a product of the World Wide Web Consortium; Verilog-XL™ is a trademark of Cadence Design Systems, Inc; Leonardo Spectrum™ trademark of Mentor Graphics.

REFERENCES

- [1] VSI Alliance. <http://www.vsia.com/>
- [2] ARM Limited. <http://www.arm.com>
- [3] AMBA Specification (Rev 2.0). <http://www.arm.com>
- [4] TM of Mentor Graphics <http://www.mentor.com>
- [5] TM of Synopsys. <http://www.synopsys.com>
- [6] TM of Coware. <http://www.coware.com>
- [7] Altera SoPC Builder. <http://www.altera.com>
- [8] Xilinx EDK, <http://www.xilinx.com>
- [9] Dev. Syst. Ref. Guide <http://toolbox.xilinx.com>
- [10] EDAXML DTD. <http://www.e-tools.com/xml/dtds>
- [11] <http://xml.coverpages.org/xmlAndEDA.html>
- [12] W.O. Cesário et al. An XML-based Meta-model for the Design of Multiprocessor Embedded Systems.
- [13] J.Zhu, W.S.Mong.Specification of Non-Functional Intellectual Property Components.
- [14] Extensible Markup Language (XML) 1.0. <http://www.w3.org/TR/xmlbase/>