Advanced Materials Research Vols 383-390 (2012) pp 2222-2230 Online: 2011-11-22
© (2012) Trans Tech Publications, Switzerland
doi:10.4028/www.scientific.net/AMR.383-390.2222

An Automatic SoC Design Methodology for Integration and Verification

De Ma, Kai Huang, SiWen Xiu, Xiaolang Yan, Jiong Feng, JianLin Zeng and Haitong Ge

Institute of VLSI Design, Zhejiang University, Hangzhou, China

Hangzhou C-Sky micro-system Company, Hangzhou, China
email: (made,huangk,xiusw,yan)@vlsi.zju.edu.cn,
(jiong_feng,haitong_ge,jianlin_zeng)@c-sky.com

Keywords: SoC, Platform, RTL, Integration, IP-XACT

Abstract. The increasing complexity of current SoC design brings a great challenge to SoC designer for fast SoC RTL integration and effective verification. In this paper, an automatic SoC integration methodology based on IP-XACT standard is proposed as a complete and effective solution for low-level RTL simulation, FPGA emulation and ASIC implementation. A bottom-up approach is adopted for design integration and verification from component level, to SoC core level, and then to final chip level. The three-core MPSoC case study not only gives the detailed usage and analysis on the proposed methodology, but also shows its efficiency to integrate a complex SoC design and its feasibility for correct SoC implementation.

Introduction

With the rapid growth of silicon process technology, the increasing requirement of embedded application drives IC designers to integrate a multi-million gates system on a chip (SoC). However, implementing such complex SoC from the given architecture to its RTL implementation is still a great challenge owing to more complicated architecture and shorter design period. To address this problem, designers are seeking a more efficient and reliable SoC design methodology. The key ideas to solve this problem are IP reuse and integrating automation. Several technologies and standards for IP reuse and SoC design have been proposed by industry and academia,such as OSCI[1],OCP-IP[2][2],and IP-XACT[3]. Among these ideas, the IP-XACT Standard [3] is one of the most attractive solutions. The SPIRIT consortium specified IP-XACT standard to enable IP reuse from various vendors and automation of design activities, i.e. IP configuration and SoC integration. IP-XACT standard provides a well-defined XML schema as meta-data that documents IP blocks and system designs, and defines an interface to make this meta-data directly accessible by automation tools[3]. Many design methodologies have been developed based on it[4], [7] and most EDA tools have already been compliant with it as well, i.e. Synopsys SoC integration platform coreAssembler is able to import an IP-XACT packaged IP; CoreConsultant will generate both IP's RTL code and IP-XACT XML file as the input of those third vendor EDA tools.

The previous methods based on IP-XACT standard mainly package the IP and system for IP hardware resource reuse and system transfer between different EDA platforms. However, the IP software resource, sub-system reuse, verification automation and FPGA emulation are not or little covered. In this paper, we proposed an effective SoC design methodology, which adopts the bottom-up approach for the target design integration and verification from component-level, to SoC core level, then to chip level. Three different platforms are generated respectively for RTL simulation, FPGA prototype emulation and logic synthesis.

This paper is organized as follows: section 2 gives a brief introduction to related work of SoC integration based on IP-XACT. Section 3 presents the proposed SoC integration flow. In section 4, we used a three-core MPSoC (Multi-Processors SoC) case study to show the efficiency and feasibility of complex SoC chip RTL integration, especially for those time-consuming and error-prone operations. Section 5 gives the conclusion.

Related Work

Recently, a number of SoC design methods have been proposed based on IP-XACT. A design flow for interchanging design components among different SoC design environments is described in[4] [5], which use the IP-XACT meta-data as the medium. However, the IP blocks are not configurable, and its system verification is not considered. In paper [6], the chip-level I/O fabric method is introduced, but its verification is not available as well. Kwanghyun cho [7] proposes a reusable platform design methodology for SoC integration and verification. It uses the sheet tables for SoC configuration, and then packages IP into IP-XACT XML file during SoC integrating process. It can generate SoC test bench automatically according to the design's IP-XACT XML file. However this methodology depends on the IP RTL library and packages IP into IP-XACT XML file during SoC integration flow, which is time-consuming and also reduces interoperability of different vendors' IP blocks. Industrial IP integration flows based on IP-XACT are presented in [8] [9] [10]. They expand the design to ESL and propose the IP-XACT based verification software code generation method. However, most of their verifications focus on system-level modeling and simulation, while low-level implementation, such as chip-level simulation and FPGA platform emulation, is not available.

Compared with previous work, in this paper, we present a more effective and complete SoC Design methodology. First, we refine the SoC platform from two aspects: system integration and verification platform generation. Second, we expand the design reuse from IP level to sub-system level and from hardware resource to software resource in order to improve the efficiency of SoC design. At last, three different platforms are generated respectively for RTL simulation to verify the correctness of chip integration, FPGA prototype emulation for real-time HW/SW co-simulation, and logic synthesis environment to compile RTL codes to gate-level net list.

Proposed Intergration Methodology

Architecture and work flow. The proposed SoC design method is composed of three parts: SoC Configuration Input interface to manually import SoC architecture information, IP Library of basic components to compose a complete SoC system, and Design Environment with a series of tools to generate the target SoC RTL codes and the corresponding verification platform automatically.

As shown in Fig 1, there are two sources for SoC Configuration Input: one is to use graphic user interface, the other is to import existing design's IP-XACT XML file to improve design reusability. In the former way, users can specify chip implementation parameters such as silicon process and library path, select and configure IP, and complete bus interface and signal connections. Then user specifying information is transferred into design's IP-XACT XML file for interoperating with Design Environment or other EDA tools. In the latter way, all the SoC information is contained in the imported design's IP-XACT XML file which can be generated from architecture exploration tools directly.

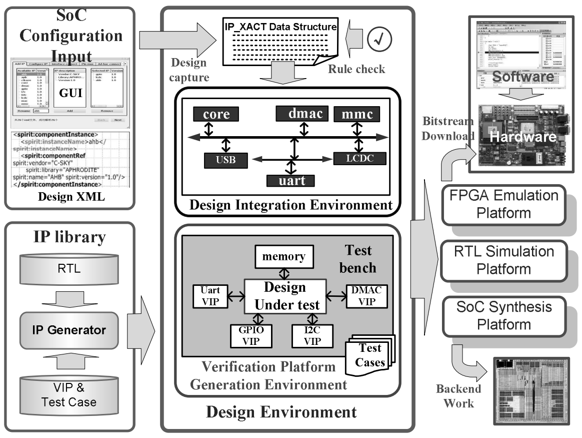

Fig. 1. SoC integration flow

Design Environment is responsible for importing design‘s IP-XACT XML file, analyzing its information, and translating it into internal IP-XACT data structure. To find incautious mistakes of IP-XACT XML file, static checks will be done automatically according to some special rules, such as signal’s multi-driving, overlapping of registers’ address and so on. Based on this data structure, two environments, Design Integration Environment and Verification Platform Generation Environment, are used to generate RTL codes of target SoC chip, as well as create its verification platform. During Design Environment’s working, it will call IP generator to create IP’s RTL, test cases and Verification IP (VIP) according to IP configuration information from IP-XACT data structure. IP library consists of four parts: component’s IP-XACT XML file, IP RTL codes, IP generator, and IP Verification environment with test case templates, verification IP, and vSW (verification software).

As shown in Fig 1, Design Environment is able to generate three different-purpose platforms for RTL simulation, FPGA emulation and ASIC implementation respectively. Based on RTL simulation platform, all SoC integration can be verified by automatically generated test cases. Based on FPGA emulation platform, automatically generated FPGA bitstream can be downloaded into the FPGA development board, and real-time applications can be run on this board to estimate system’s performance. Therefore, hardware and software co-simulation can be run efficiently on this platform. Based on ASIC implementation platform, Synopsys Design Compiler scripts can be generated automatically, which can compile RTL codes to gate-level netlist. Then backend engineers can implement the SoC design into chip.

Reusable IP library. In order to completely separate IP provider from SoC integration tools and increase IP reusability, the proposed method supports both IP components and sub-systems complied with IP-XACT standard. All of the components in the IP library are packaged into IP-XACT XML files so that each IP contains two parts: IP-XACT coded XML file and IP generator. The IP-XACT XML file includes the following parts:

1) IP configuration parameters

IP configuration is essential for reusable IP which means it can be used by different system environment requiring different features. All of the reusable IP’s configuration parameters are packaged into component’s IP-XACT XML file. SoC designer will modify the parameters’ value through configuration interface as shown in Fig 1 according to the application. The availability of IP configuration improves flexibility in SoC design.

2) Memory map

Any IP has one or more registers for communication with others and some IP even has internal memory for data buffering. Both registers and memory information is encoded into component’s IP-XACT XML file, which contains address offset, data width, reset value and each bit’s accessibility. This information can be used on address allocation checking, coverage monitoring, which are important for verification automation.

3) Interface definition and mapping:

Each IP component has larger number of physical ports. The connection of these ports will be a timing-consuming and error-prone job. Fortunately, most time they are connected to other IP in groups with several signals, where each signal is predefined like AMBA bus and DMA hand-shaking ports. Some physical ports have special functions for software like the interrupt port, which will be treated as interface as well to improve verification automation. Interface definition represents bundling specified signals into one group and giving them proper logical name as well as connection mechanism. After the interface has been defined, component’s physical ports are mapping to the certain interfaces. During SoC integration, if two interfaces are connected, the corresponding physical ports will be connected automatically, which improves the SoC integration efficiency greatly.

4) Other packaging information

Besides the common IP attributes described in IP-XACT standard, some vendor expanding information, such as IP’s process technology, system’s FPGA board selection, extra interfaces definition etc, is packaged as well. These extensions are used to support ASIC verification automation, FPGA emulation, and ASIC synthesis environment generation.

The IP generator is invoked by Design Environment and responsible for performing a variety of tasks, including accessing the components' IP-XACT XML file, configuring IP, creating the IP's test cases and RTL etc.

IP library also includes some packaged stable subsystem for reuse. The subsystem is a pre-designed system which links a number of components to fulfill some specific functions like audio application, for example. By introducing the subsystem, the IP reuse capability is extended from component level to system level for higher design efficiency. The design reusability is greatly improved when Design Environment integrates several subsystems into one complex SoC.

Design Integration Methodology. A complex SoC platform contains a lot of IPs and each IP supports different kinds of functional configuration, which boosts platform flexibility while increases SoC integration complexity because deep understanding on the usage of these configurable IP is demanded in order to generate component's required RTL codes. Besides that, integrating the IPs to form a top-level RTL design needs tremendous signals connections, which is time consuming and vulnerable to human errors. Automation of these two bothering jobs is inevitable for the SoC integration.

In the proposed method, the Design Integration Environment is responsible for generating component RTL code according to IP's configuration and forming the top-level RLT design automatically. The complexity of the integration flow is increasing rapidly along with the larger system architecture. In order to decompose the complexity of SoC integration, a bottom-up approach is adopted for proposed design integration methodology, refining from component level to SoC Core-level, and then to Chip level. Just as Fig 2 shows, Design Environment imports the design's IP-XACT XML file and abstracts each level's information for next SoC implementation.

Component is the essential element that the architecture specification instantiates. In component-level, Design Environment abstracts all the IP's configuration information and then calls specified IP generators from IP library to get the necessary component environments automatically for next platform generation. These components may be specific-configuration IP, or fully verified subsystems. As Fig 2 shows, the sub-system is implemented in recursive way. During the iteration of subsystem generation, each IP and its connections are generated as those specified in subsystem's IP-XACT XML file.

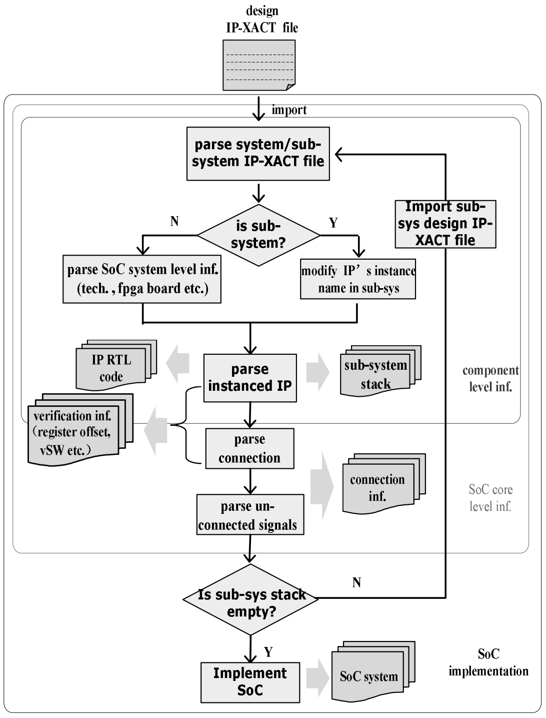
design IP-XACT file
import
parse system/sub-system IP-XACT file
is sub-system?
N
Y
parse SoC system level inf. (tech., fpga board etc.)
modify IP's instance name in sub-sys
Import sub-sys design IP-XACT file
IP RTL code
parse instanced IP
sub-system stack
component level inf.
verification inf. (register offset, vSW etc.)
parse connection
connection inf.
parse un-connected signals
SoC core level inf.
Is sub-sys stack empty?
N
Y
Implement SoC
SoC system
SoC implementation

Fig. 2. Design IP-XACT parsing process

In SoC core-level, Design Integration Environment is responsible for generating subsystem and design's top-level RTL codes. Those given systematic configuration is implemented, for example, address space allocation, interruption vector port number, DMA handshake port number, and bus number for each IP. All the interface connections are mapped to physical ports' connections automatically, which decreases the signals' connection work greatly. Based on the information, all selected components are connected into one module.

In chip-level integration, owing to the limited PAD resources, many functional IP ports may be connected to one PAD with multiplexer logic. Besides, Design for Test (DFT) function also demands PAD multiplexer because of different chip work modes. Therefore, many multiplexers may be generated automatically, and the port reusing relationship information is stored for generating test cases and configuration codes to verify these logics. After this stage, all SoC RTL codes have been generated automatically.

A. Verification Platform Generation Methodology

One of the major benefits that we can get from IP-XACT based design methodology is verification automation. Verification Platform Generation Environment generates the testbench and stimulus for ASIC simulation platform and bitstream and verification software (vSW) for FPAG emulation automatically, using the information from component or design's IP-XACT XML file.

There are two kinds of test cases in ASIC simulation platform: register or memory access test cases and function test cases, used to verify various aspects of SoC design. Address decoding or bus architecture, interconnection are tested to be sure that core level design is well implemented while ports' reuse and pads' assignation need to be verified in order to make sure that the chip level design is expanded from the core level design correctly.

Register access test is one of main tests to check whether the IP has been integrated into SoC design correctly. It will find the bus memory mapping bugs. Although it is not a difficult job, it is tedious and cumbersome to write test case for all registers in a complicated SoC since there are so many registers and you have to understand each component's registers property. In the proposed method, all the register access test cases are generated automatically using register's information abstracted from IP-XACT XML files like address offset, data width and reset value, for example. The memory access tests are treated in a similar way. The other kind of test cases is function test cases. Since SoC integration is based on existed IP with fully verified, the main responsibility of the function test case is to check connection of the IP to pads and to surrounding IP like DMA handshake signals' connection, interrupt signals' connection. For verification platform generation environment, it is easy to modify the IP-specific test case templates to generate these test cases according to the connection and configuration information contained in design or component's IP-XACT XML file.

Besides the test cases, the testbench is generated automatically as well. The ASIC simulation platform is shown in Fig 3.

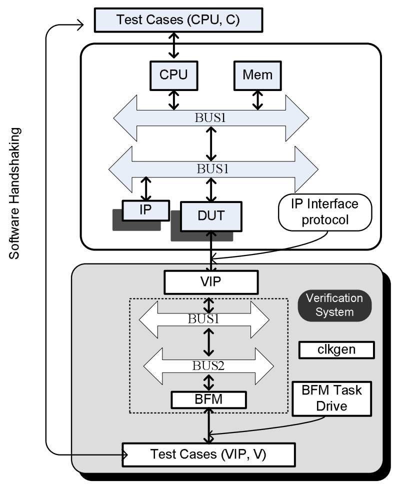

Fig. 3. ASIC simulation platform

For most test cases like register access cases, function test cases used to check internal IPs' connection, the verification system is useless and all these cases are run on CPU with self-result check. The verification system is mainly used to test the connection between IP and pad, especially for the IPs having communication protocol with off-chip device such as USB, UART, and so on. As Fig.3 shows by communicating between VIP (verification IP) and DUT (design under test), the IP's connection to pad will be checked. Verification IP is the behavior module which is DUT's operation object or has the same function as DUT, for instance, when DUT is a DRAM controller, the VIP is just a DRAM behavior module while DUT is UART, the VIP is an UART as well. If one pad is reused by more than one IP or IP's port is connected to several pads, more than one case will be generated to make sure that each pad's reuse mode is covered.

FPGA emulation provides a more accurate verification for target design in both function and performance than ASIC simulation. It also gives the designers more confidence about the hardware's physical feasibility of the RTL codes. The generated RTL code of design integration is imported by FPGA generation and some hard IPs of ASIC process are replaced with those of FPGA technology. For example, ASIC PAD should be replaced by FPGA PAD before implementing the system on FPGA. The FGPA version RTL codes are compiled into gate-level Netlist by Synplicity Synplify and then implemented into Bitstream by Xilinx ISE (FPGA board is based on Virtex 4 FPGA). The FPGA information abstracted from design IP-XACT XML file is used to generate the UCF. Different FPGA Boards require different UCF File Templates to generate targeted UCF File. Owing to PAD multiplexer, more than one UCF file may be generated for different PAD functions.

Case Study for MPSoC Design

A complex MPSoC with three heterogeneous CKCore processors [11] is used as a case to show the efficiency of the proposed methodology. The main function of this three-core SoC involves audio/video multimedia application on Linux Operation System (OS). As shown in Fig 4, the kernel system consists of four subsystems.

CPU1 Subsystem works as the main controller for the whole system. Linux OS can be run on this subsystem for supporting more sufficient applications. In this subsystem, main CPU is configured as CK510m CKCore processor [11] with MMU. Timer is configured to support general-purpose timer and watch-dog timer for OS and high-level application usage.

CPU2 Subsystem is designed for audio application. In this subsystem, main CPU is configured as CK510s CKCore processor [11] with 8K Cache and 16K SPM so as to meet different kinds of audio algorithm process requirements. Two digital audio interfaces, AC97 and I2S, are equipped to support most audio applications. This subsystem is able to meet different requirements of digital audio codec or audio effect process application.

CPU3 Subsystem is designed for video application. In this subsystem, main CPU is configured as dual-issue superscalar CK610 CKCore processor [11] with 8K Cache so as to be capable for massive data computation. A video post-process accelerator is also used for image scaling, color translation and pixel interpolation. One 8K SRAM is also integrated in this subsystem for buffering several lines of pixel data for video process.

Peripheral Subsystem is composed of all modules for inter-subsystem communication, external memory controller and data transfer interface. Mailbox module is designed for communication message passing between three CPU subsystems. Ethernet MAC and USB Host/Device modules are used to support usual massive data transfer for audio, video and other applications.

As is specified, there are totally 27 different kinds of functional IP used in this MPSoC case and thousands of interconnections. Most of these IPs support different kinds of functional configuration. It will be a challenge for the SoC designer to configure each IP and complete the tedious ports connection without introducing any errors. Fortunately, the proposed methodology supports sub-system integration and interface connection.

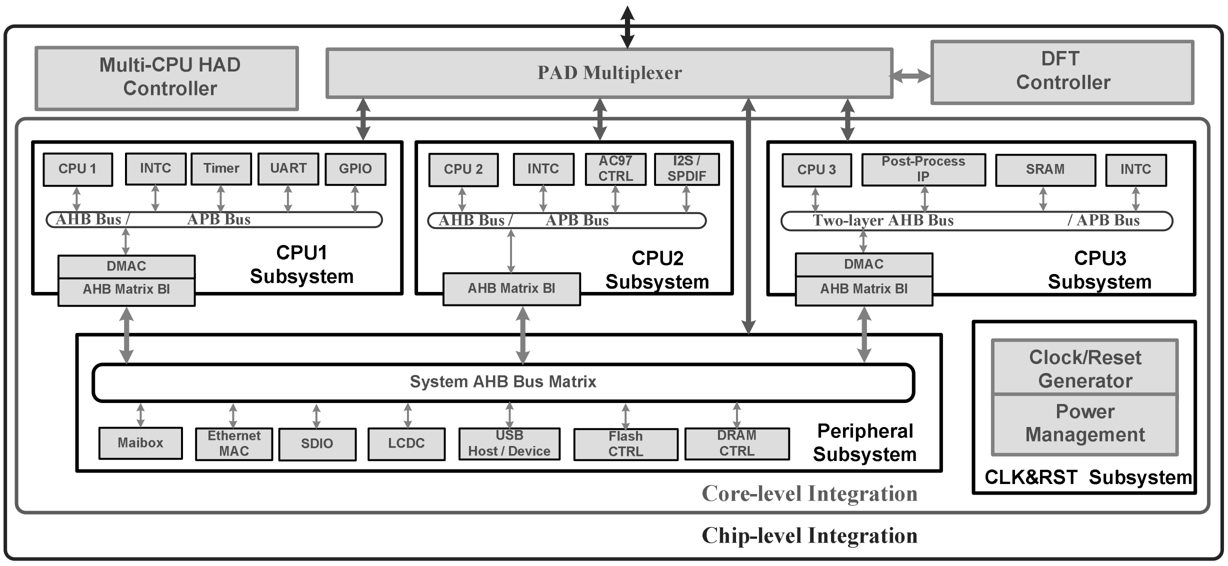

Fig. 4. The architecture of the MPSoC

Table 1. THE INTERFACES LIST OF MPSOC

Subsystem	HMI	HSI	IntI	DHI	BI
CPU1 Subsystem	2	9	17	4	0
CPU2 Subsystem	2	4	12	2	0
CPU3 Subsystem	4	4	11	0	0
Peripheral Subsystem	4	11	0	0	0
SoC Core System	0	0	27	0	4

HMI: AHB Master Interface connection
HIS: AHB/APB Slave Interface connection
IntI: Interrupt Interface connection
DHI: DMA Handshake Interface connection
BI: Subsystem Bus Interface connection

Table I shows each kind of interface's connection number in the proposed MPSoC. As is shown, it's found that there are totally 12 AHB master interface connections, 28 AHB slave interface connections, 67 interrupt interface connections, 8 DMA handshake interface connections and 4 subsystem bus interface connections. Each interface has predefined signal configuration and it is relatively easy for the designer to connect them due to it regularity. Fig 5 gives the interface connection and port connection's comparison. As it is shown, the proposed methodology could save the connection up to 90% at most.

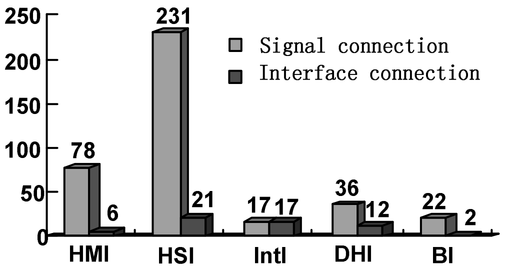

Fig. 5. Interconnect comparison between interface and signals

Further, higher-level subsystem reuse in our proposed methodology can reduce unnecessary IP configuration and connection work and improve design efficiency. As Fig.4 shows, MPSoC consists of four sub-systems and all of these sub-systems are common in audio and video application SoC. CPU2 sub-system, CPU3 subsystem and peripheral subsystem have been existed in previous SoC with similar architecture and could be reused. Based on the sub-systems, we only need to integrate the CPU1 sub-system and modify some configuration parameters and connections of the existed sub-systems, and then integrate the four sub-systems together to get the MPSoC. Table II shows the sub-system reuse technique could save configuration work up to 84% and connection work up to 73%.

Table 2. INTERGRATION IMPROVING BASED ON SUB-SYSTEM RESUE

Sub-system name		No sub-system reuse	Reuse sub-system	Reduction (%)
CPU2 sub-system	Configuration parameters	85	12	84.8%
	Interface connection	15	4	73.3%
CPU3 sub-system	Configuration parameters	95	20	78.9%
	Interface connection	20	7	
Peripheral Subsystem	Configuration parameters	135	30	77.8%
	Interface connection	36	20	44.4%

During MPSoC's integration, according to the configuration information from design's IP-XACT XML file, the proposed methodology generates the ASIC simulation and FPGA emulation automatically as well. In ASIC simulation platform, there are totally 138 test cases to test core-level connection (1 test case for register access for all IP, and 19 test cases for interrupt and DMA handshake connection), and chip-level connection (105 test cases for all IP function ports test, 5 test cases for functional port MUX test, 3 test cases for multi-CPU HAD MUX port test and 5 test cases for DFT MUX test). Based on these test cases, automatically generated regress test script can run RTL simulation based on RTL verification platform. The whole MPSoC is implemented into Virtex-4 XC4VLX160 FPGA chip, by using 50724 Slices and 102035 LUT. Then, we can run the complete application software on this FPGA platform for hardware/software co-simulation.Linux OS can be run on CPU1 Subsystem using UART as user interface, with MP3/OGG audio decoder running on CPU2 Subsystem, and JPEG decoder running on CPU3 Subsystem. It showed reduction of verification time more than 60% comparing with conventional testbench writing and FPGA emulation methodology.

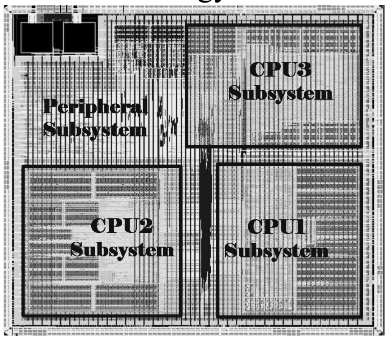

Fig. 6. The physical implementation of the MPSoC

In ASIC chip implementation stage, the proposed IP-XACT method is able to generate Synopsys Design Compiler (DC) scripts and its working environment automatically according to silicon process related parameters specified in design's IP-XACT file. The designer can modify the generated DC script according to requirements. After gate-level Netlist is generated, backend designers implemented the whole MPSoC into this chip, as shown in Fig 6. In SMIC 130nm silicon process, the whole chip area is about 19.2 mm2 (CPU1 subsystem: 3.9 mm2, CPU2 subsystem: 4.1 mm2, and CPU3 Subsystem: 3.7 mm2).

Conclusion

A new IP-XACT based SoC design methodology was proposed for automatic code and platform generation at three different levels: component level, SoC core level and chip level. The proposed methodology is able to provide a complete and effective solution for low-level RTL simulation, FPGA emulation and ASIC implementation. Three-core MPSoC case study gives the detailed usage and analysis on the proposed platform, and also shows its efficiency to integrate a complex SoC design and its feasibility for correct SoC implementation. According to the next-generation IP-XACT standard for ESL design, we will keep on improving CKSoC design method to cooperate with Virtual Platform of ESL tools using Design IP-XACT XML file as medium data.

Reference

[1] Open SystemC initiative. SystemC standard 2.2[OL]. (2007-3-14) [2010-4-11]. www.osci.org.

[2] OCP-IP. Open Core Protocol[OL]. [2010-4-11]. www.ocpip.org

[3] The SPRIRIT Consortium, “SPIRIT 1.4 specification”, http://www.spiritconsortium.org/home/, March, 2008

[4] Kamil Synek Sonic, Inc. “Using SPIRIT Cores in SonicsStudio”, System-on-Chip, 2006. International Symposium on, PP. 1-4 13-16 Nov. 2006

[5] Victor Berman, Marino Strik, et al. “Industrial Proving the SPIRIT Consortium Specifications for Design Chain Integration”, Design, Automation and Test in Europe, 2006. DATE’06, PP. 1-6, 6-10 March 2006.

[6] Geoff Mole, Marino Strike, et al. “Philips Semiconductors Next Generation Architectural IP Reuse Developments for SoC Integration”, http://www.design-reuse.com/articles/, May, 2009

[7] Kwanghyun Cho, Jaebeom Kim, et al., “Reusable Platform Design Methodology for SoC Integration and Verification”, International SoC Design Conference, 2008.ISOCC’08, International, PP. 78-81, 24-25 Nov. 2008.

[8] Wido Kruijtzer, Pieter van der Wolf, et al., “Industrial IP Integration Flows based on IP-XACT Standards”, Design, Automation and Test in Europe,2008. DATE’08, PP. 32-37, 10-14 March 2008.

[9] Marcin Zys, Emmanuel Vaumorin, et al. “Straightforward IP Integration With IP-XACT RTL-TLM Switching”, http://www.techonline.com/learning/techpaper, June 2008.

[10] IP-XACT USER GROUP, “Using IP-XACT in Complex SoC I/O Integration and Register Management”, Design Automation Conference, 2008. DAC08, June 10, 2008.

[11] C-SKY CKCore Processor, www.c-sky.com

Manufacturing Science and Technology, ICMST2011
10.4028/www.scientific.net/AMR.383-390

An Automatic SoC Design Methodology for Integration and Verification
10.4028/www.scientific.net/AMR.383-390.2222