The Development of Reusable SoC Platform based on OpenCores Soft Processor for HW/SW Codesign

Younghoon Bin and Kwangki Ryoo, *Member, KIMICS*

Abstract**—**Developing highly cost-efficient and reliable embedded systems demands hardware/software co-design and co-simulation due to fast TTM and verification issues. So, it is essential that Platform-Based SoC design methodology be used for enhanced reusability. This paper addresses a reusable SoC platform based on OpenCores soft processor with reconfigurable architectures for hardware/software codesign methodology. The platform includes a OpenRISC microprocessor, some basic peripherals and WISHBONE bus and it uses the set of development environment including compiler, assembler, and debugger. The platform is very flexible due to easy configuration through a system configuration file and is reliable because all designed SoC and IPs are verified in the various test environments. Also the platform is prototyped using the Xilinx Spartan3 FPGA development board and is implemented to a single chip using the Magnachip cell library based on 0.18㎛ 1-poly 6-metal technology.

Index Terms**—**OpenRISC, SoC, Platform, Co-design

I. INTRODUCTION

As billion transistors system-on-chip (SoC) becomes commonplace and design complexity continues to increase, designers are faced with the daunting task of meeting escalating design requirements in shrinking time-to-market windows. Because of this, hardware/ software co-design and high level design reuse emerged as a possible approach to boost design productivity [1]. There are mainly IP-based design method and platform-based design method for SoC HW/SW co-design. In IP-based SoC design, reusing methodology of complex pre-defined design blocks, the system intellectual property modules or virtual component applied to design a system, electronic system engineers continuously are offered ready-made or customizable functional cores which may be added to designs [2].

But to use a pre-designed block, an engineer must understand how it works and how it integrates with other components within the design. Also using a pre-designed IP block cannot guarantee first cut success without extra effort spent on verification and debugging. Because of this difficulty, a platform-based design is issued and several companies are using the platform-based design approach as an effective strategy to address product complexity and time-to-market at all levels. A platform-based SoC design methodology permits reuse of key SoC functional component. So, the platforms provide users with ample room for product differentiation. Derivative design can be accomplished quickly by adding just a few IP components. Moreover, integrated architecture minimizes verification uncertainties that greatly reduce design effort and risk [3].

There are various SoC Platform environments. For example, the Nexperia developed by Philips Inc. is standard platform for video-centric multi media product in company. Also the famous OMAP developed by Texas Instrument Inc. is a typical SoC platform for mobile solution and fully programmable platform such as, Xilinx's Virtex-II Pro and Altera's Excalibur with programmable logic is using for embedded applications. This paper presents a synthesizable SoC platform which is comprised of OpenCore's popular IPs and a set of software development environment built by us. The platform is verified through various test environments and is implemented to single die in range of 2.6mm^2, and can operate at 91MHz frequency.

The reminder of this paper is organized as follows. In section 2, we will describe the platform-based SoC design methodology. Section 3 exposes the SoC platform architecture and introduces main components such as processor and on-chip bus, also software development environment. Section 4 shows the verification result of all the part that composes the platform by applying it to three level verification methods. In the section 5 the platform implementation and its summary of specification will be demonstrated. Finally, the conclusion, result, and future works are presented.

Manuscript received August 1, 2008; revised September 12, 2008. Younghoon Bin and Kwangki Ryoo is with the Department of Information and Communication Engineering, Hanbat National University, Taejeon, 305-719, Korea (Tel: +82-42-821-1710, Email: {yhbin, kkryoo}@hanbat.ac.kr)

II. PLATFORM-BASED DESIGN

Platform-based design is an IP reuse strategy that facilitates the creation and verification of designs containing sophisticated IP from many different sources. Platform-based design provides rapid creation and verification of SoC designs by automating complex, tedious, and error-prone design creation, IP integration, and verification steps. By automating the non-differentiated design, design resources can devote their time to value-added design [4]. Platform-based SoC design allows an organization to develop a complete SoC that is central to its product line. Once the SoC platform is fully operational, derivative designs in which only a few virtual components are added or dripped or modified can be accomplished rapidly. Fig. 1 depicts a methodology of platform-based SoC design.

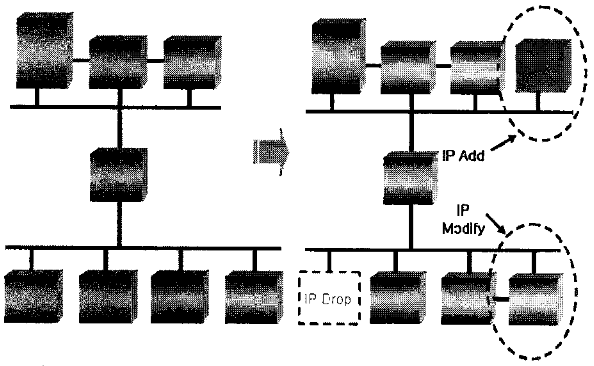


Fig. 1 Design methodology of platform based SoC

Generally, a platform-based SoC design goes through two stages [5]. The first stage creates the platform. A semiconductor house usually designs a platform for SoCs. The platform design process considers the needs of the application domain, including what is required by relevant standards and other descriptions of requirements. The second stage creates a product using the platform. Systems houses with a particular product in mind can then use the semiconductor house's platform as a starting point for their chip design. This stage must take into account the needs of the customer. Fig. 2 shows a typical platform-based design flows.

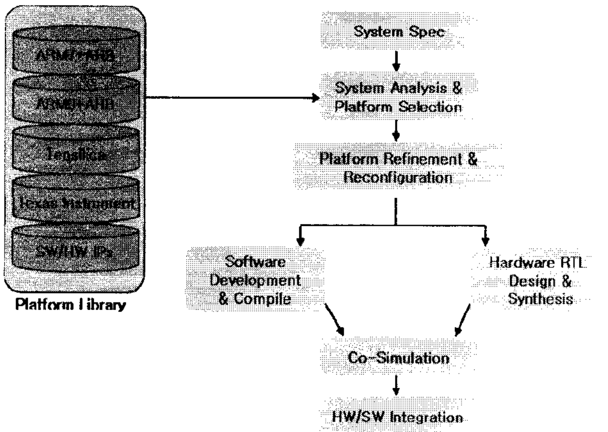

Tensilica
Texas Instrument
SW/HW IPs
Platform Library
System Spec
System Analysis & Platform Selection
Platform Refinement & Reconfiguration
Software Development & Compile
Hardware RTL Design & Synthesis
Co-Simulation
HW/SW Integration

Fig. 2 Platform-based SoC design flows

First, a system specification describes significant element and application requirements with C/C++ or SystemC. With considering the flexibility, power, performance, size and TTM, system analysis and platform selection phase is a key step that selects a possible platform. Refinement performs a high-level development process. In this step, application is partitioned into two groups, hardware and software, based on functionalities, performances and constraints. Finally, the design can be exported to a co-simulation tool and other tools to finish synthesis and execution code generation. In this design flow, design architecture and specification has less flexible because of fixed properties of platform selected and limitation of available platform.

III. PROPOSED SOC PLATFORM

A. Platform Architecture

We have developed an OpenRISC-based SoC platform that includes a 32bit RISC processor core and the minimum set of elements needed to provide basic functionality. These elements are WISHBONE interconnect, UART, debug interface, GPIO, VGA/LCD & DMA controller, and on-chip RAM. Fig. 3 shows the block diagram of the proposed SoC platform.

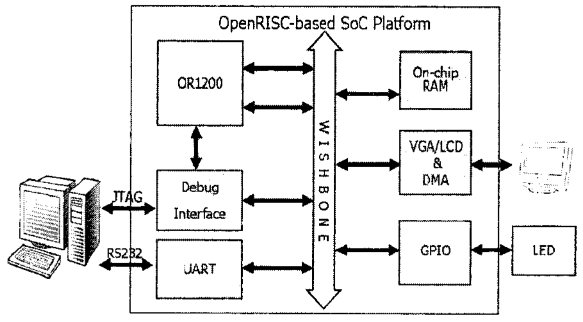


Fig. 3 Proposed platform architecture

The UART controller provides serial communication capabilities to the platform using an RS232 protocol. We have chosen a 16550 compatible UART because it is the most usual industry standard. Additionally, this UART provides a basic method to access the microprocessor and RTOS from host environment. This access is very important for debugging and loading of new software versions using serial communication.

The debug interface block performs various functions, such as memory initialization, processor and peripherals configuration, and system trace for debugging. Also, the platform includes on-chip SRAM consisted of four 1K byte block and VGA/LCD controller that support capability for CRT and LCD display. The VGA/LCD core supports a number of color modes, including 32bpp, 24bpp, 8bpp grayscale, and 8bpp pseudo color, and GPIO module performs a general transfer between the platform and external word[6][7][8]. The basic communication channel of the platform is an OpenCores WISHBONE on-chip bus. It has synchronous data and address buses with multiple masters and slaves. An arbiter decides in each moment which master takes control of the bus.

B. OpenRISC 1200 Microprocessor

We use the OR1200, a publicly available processor for our development of SoC platform. This soft-core is freely distributed under an LGPL license at OpenCores website, and fits for composition SoC in many ways. OpenRISC 1200, synthesizable core, is implemented with Verilog HDL, and has high flexibility because all configuration options for the processor are gathered together into a single file containing numerous define statements. A block diagram of the OR1200 architecture is depicted in Fig. 4.

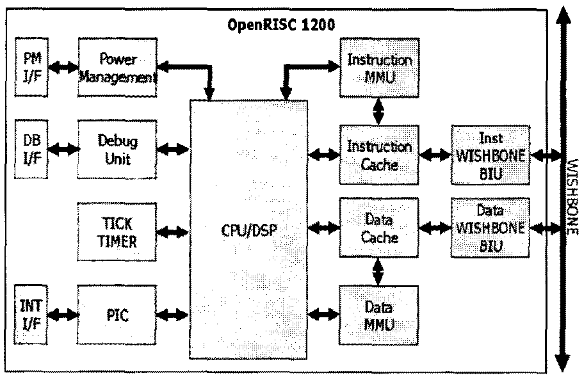


Fig. 4 Overall architecture of OpenRISC

The OR1200 is a 32-bit scalar RISC with Harvard micro architecture which has separated instruction/data bus, 5 stage integer pipeline, virtual memory support and basic DSP capabilities. OR1200 is intended for embedded, portable and networking applications.

Instruction and data caches are present in the microprocessor, and both default to 1-way direct-mapped 8KB caches. The memory management unit are implemented for data and instruction MMU as 64-entry, 1-way direct-mapped translation look-aside buffer. Other integrated functionality includes real-time debug, tick timer, programmable interrupt controller, and power management support [9][10]. The OR1200 communicates with the WISHBONE interconnect.

C. WISHBONE On-chip Bus

The OpenRISC 1200 interfaces to memory and peripherals via two WISHBONE compliant 32-bit bus interface. The WISHBONE System-on-Chip (SoC) interconnect architecture for portable IP cores is a portable interface for use with semiconductor IP cores. The WISHBONE interface supports point-to-point, shared bus, cross-bar switch and data flow interconnection scheme. The multi-master and multi-slave bus also supports both single data transfers and burst transfers [11]. Block diagram of WISHBONE interconnection is depicted in Fig. 5.

The WISHBONE architecture is consisted of 8-master and slave side interfaces, round robin arbitration logic and glue logic including multiplexer and address decoder. In this architecture, multiple bus masters can use a common shared bus when specific bus master is granted, and then can transfer data to a selected slave.

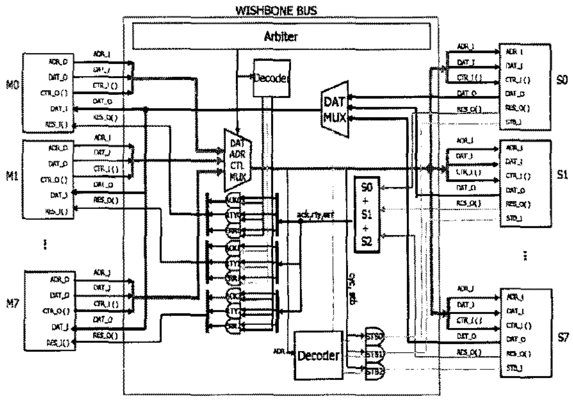


Fig. 5 Overall structure of WISHBONE interconnection

D. Development Environment

The distribution of the Open Source OpenRISC core includes a set of development tools, comprising binary utilities, C/C++ compiler, debugger, all of them ported from GNU tools, and an instruction set simulator developed by the OpenRISC project team [12]. Many operating systems such as eCos, uClinux, Linux, RTEMS, microC/OS-II have been ported to OpenRISC architecture. SoC designers can use this integrated development toolkit to develop and verify when software program development, ISS-based software emulation, executable program configuration and debugging are performed for specific application. For platform software development environment, GNU tool chain is built in Linux 7.3 platform, and uClinux OS is ported to OpenRISC architecture for developing device driver and target software.

E. Design Flows

Fig. 6 shows a revised design flow for platform design and application design using the platform. First, platform architecture is defined, and power, size, operation condition, functional analysis and estimation of performance are performed for anticipated applications. In the platform integration, existed and customizable IP are merged and integrated to SoC platform with a number of design methodologies, and verified through test environment.

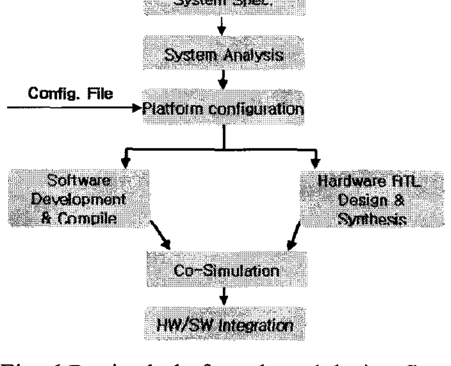


Fig. 6 Revised platform-based design flow

For application SoC, that is instantiated design of SoC platform, platform reconfiguration can performed with simple methodology using a file for configuration. For a tiny application, non-used hardware block can removed, and can add a number of IPs or modify the platform architecture according to application complexity and design require-ments. Application specific block is partitioned to hardware and software part. Using hardware/software co-simulation, system functionality is verified and then application hardware and software are integrated through a hardware/software development environment built by us. Hardware is implemented in FPGA, and software is loaded and executed in the SoC inside FPGA.

IV. VERIFICATION

A. Module Simulation with HDL Simulator

At this level, functional testbench described with Verilog HDL is used to verify design. Module level verification tests major functionality of the block and signaling between each module. For easy monitoring, assertion and monitor code in Verilog HDL is used. In this environment, we have checked activity and special corner case using various test vectors, verified by comparing the result of logic simulation with commercial simulator. Most of the testbenchs are described at the behavioral level, and various task and function are used for test vector generation and debugging of modules.

In our works, the Modelsim, RTL simulator, is used for functional verification of platform. For generation of test patterns, first cross-compile some test program developed, and then executive file is created. After that we can use the objcopy utility to generate the binary file which is then transformed into hex file by bin2hex program. Then the hex file can be read into On-chip SRAM model in testbench, and simulated by simulator.

Fig. 7 shows result of simulation for OpenRISC processor. The result indicates that instruction fetch cycles of processor are performed during the three clock periods in case cache memory was not implemented. Also the instruction and data interface signals are activated in accordance with WISHBONE specification.

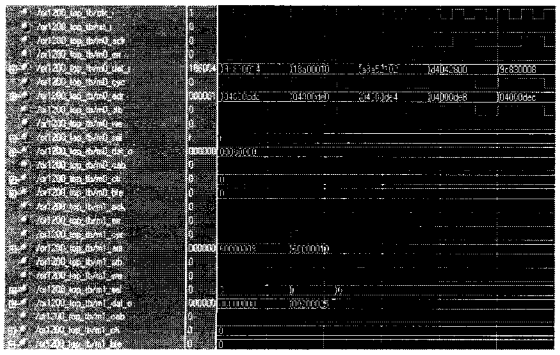


Fig. 7 Result of simulation for OpenRISC processor.

Serial communication program to test design is compiled into OpenRISC execution code using the OpenRISC GNU tool-chain. The binary image is loaded into SRAM memory model in testbench, as needed by the Verilog simulator. The task of testbench reads the hexadecimal code and initializes the memory model. Consequently, processor can execute instructions in memory model by location and test the processor and other hardware Fig. 8 shows simulation result for serial communication function of SoC platform.

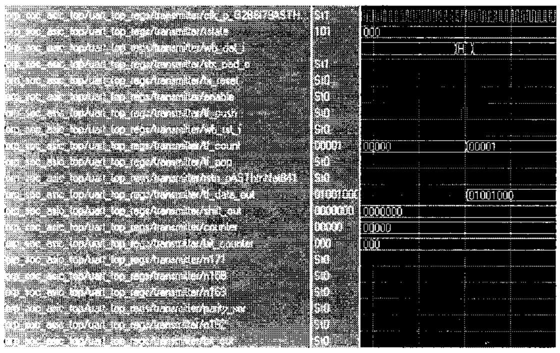


Fig. 8 Result of simulation for SoC platform serial communication

The waveform of Fig. 8 shows that transmission data is transferred from processor to the UART port through on-chip bus. In the waveform, one character 'H', represented in ASCII value, is stored to FIFO (First In First Out) buffer in the UART module, and is transferred to txd pin in external output port of UART module.

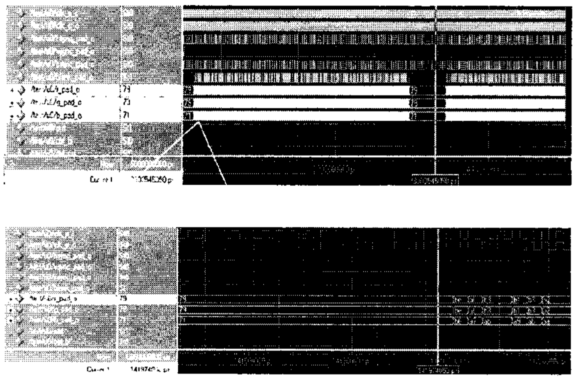


Fig. 9 Result of simulation for SoC platform VGA driver

There are other simulation examples for SoC platform peripherals, vga driver, testing. Fig. 9 shows a result of simulation for vga driver functional operation and timing generation. The simulation framework of vga driver is similar to previous process. First cross-compile the vga driver software with C image file converted from raw file primary proceeds by raw2c program. And then an executable file is generated with memory layout using linker scripts file. After hex file created though bin2hex utility, the hex file is likely loaded by Verilog user defined tasks.

B. ISS-GDB Software Emulation

ISS-based simulation is available before hardware implementation is completed or there is no physical device. Designer can confirm architecture and instruction level functionality of processor or test the user program at firmware or middleware with the cycle-accuracy architectural simulator. Using ISS developed by OpenCores, it is possible to test software and hardware IP through configuration of specified hardware. Fig. 10 shows the environment of ISS-based software simulation.

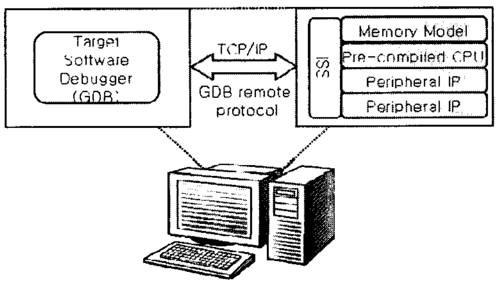


Fig. 10 Test environment of architecture/instruction level verification

All terminals are accomplished in a Linux 7.3 machine, various peripheral IP and memory models is configured to a specific system with processor model in ISS. Or1ksim, OpenCores architectural simulator, supports OpenCores peripherals, such as VGA/LCD, UART, DMA, Ethernet, general purpose IO, PS/2 interface and so on. Or1ksim also provides remote debugging through a network socket with GNU debugger. GDB, Target software debugger, loads image to the memory model of ISS, and then run the program and debugs the operation. Under the cross compiling environment, we first compile C/C++ program or assembler into executive file. Then we can use or1ksim to execute every instruction and observe the change in memory and resisters. Figure 11 shows the architectural /instructional simulation result using GDB-ISS connection.

For verification of the platform, abstract level of configured hardware IP and software are integrated to GDB-ISS simulation environment and the test program is virtually executed in the SoC model. GDB remotely control the SoC model through TCP/IP protocol.

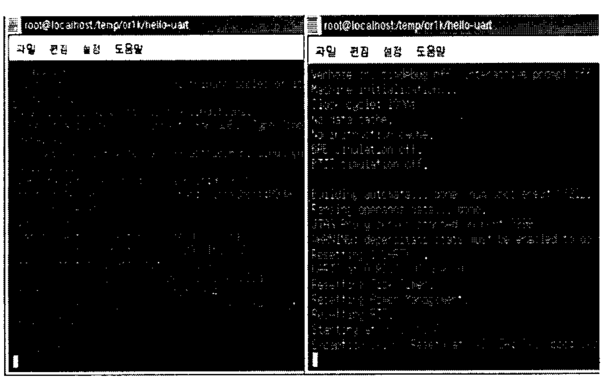

root@localhost:/temp/or1k/hello-uart
파일 편집 설정 도움말
root@localhost:/temp/or1k/hello-uart
파일 편집 설정 도움말

Fig. 11 Architectural/instructional level simulation using GDB-ISS connection

The right side terminal is OpenRISC ISS and left is GNU GDB. Both terminals are connected with TCP/IP protocol which uses the same port definition.

The or1ksim ISS is a C/C++ application that runs on the host computer, to mimic the behavior of a program running on the OpenRISC processor, and it is a generic architecture simulator capable of emulating OpenRISC based computer systems. There is a simulation example using ISS which is configured with our SoC model. In the simulation, the VGA peripheral can output images like the following Fig. 12, showing uClinux booting on a VGA console.

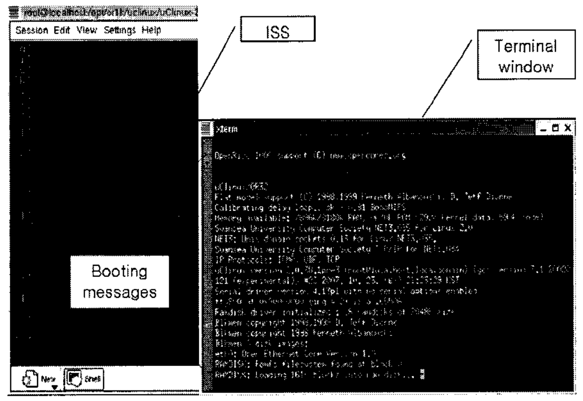


Fig. 12 uClinux booting on ISS and a VGA console

C. HW/SW Co-verification with FPGA

At this level, platform hardware/software co-verified using FPGA prototyping. Hardware of platform is programmed into FPGA, some software programs for running on the platform is loaded to on-chip ram in SoC platform. In this verification environment, some test programs (instruction level test) have been developed to validate particular operations of the processor and peripheral IPs. Fig. 13 shows FPGA prototyping environment for system level verification.

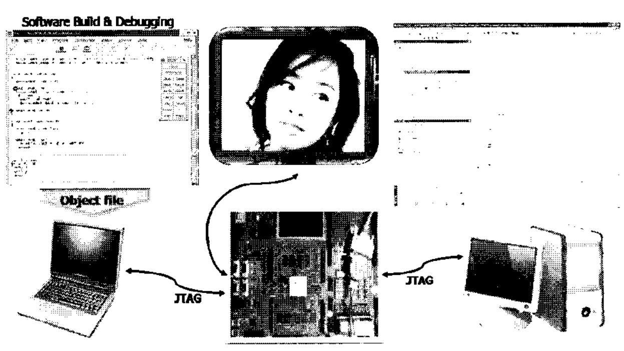
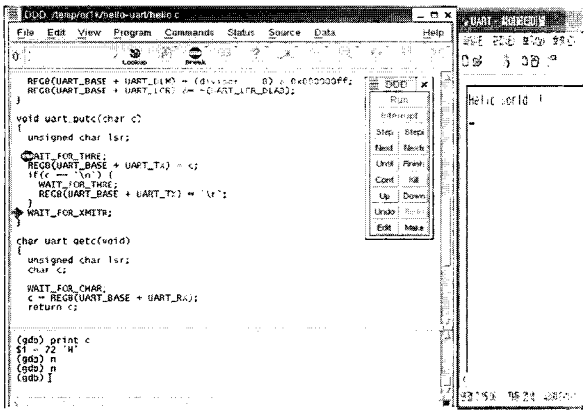


Fig. 13 System level verification using FPGA prototyping

Completed SoC hardware platform is configured using JTAG connector on FPGA development board and then test program image cross-compiled with or1200 compiler is downloaded for SoC HW/SW verification. GDB/DDD debugger is used for program configuration and debugging. Fig. 14 shows the serial test program and executed result of the program. The program is running on the OpenRISC 1200 based SoC inside FPGA.



```
  REG8(UART_BASE + UART_DLM) = (divisor >> 8) & 0x000000ff;
  REG8(UART_BASE + UART_LCR) &= ~(UART_LCR_DLAB);
}

void uart_putc(char c)
{
  unsigned char lsr;

  WAIT_FOR_THRE;
  REG8(UART_BASE + UART_TX) = c;
  if(c == '\n') {
    WAIT_FOR_THRE;
    REG8(UART_BASE + UART_TX) = '\r';
  }
  WAIT_FOR_XMITR;
}

char uart_getc(void)
{
  unsigned char lsr;
  char c;

  WAIT_FOR_CHAR;
  c = REG8(UART_BASE + UART_RX);
  return c;

(gdb) print c
$1 = 72 'H'
(gdb) n
(gdb) n
(gdb) 
```

```
Hello world !
```

Fig. 14 Executed result of test program

Serial test program initializes the UART device, and includes uart_getc() and uart_putc() functions. If the uart_putc() function is called, UART of SoC platform transmits a character transferred from processor. Running the program, it shows the result of execution in the hyper terminal of host PC.

V. SYSTEM IMPLEMENTATION

A. FPGA prototyping

XC3S1000 device in Xilinx Spartan 3 series has used to implement SoC platform for FPGA prototyping. The device is capable of using one million gate logic and includes dedicated storage elements, Xilinx block RAM [14]. In this work, FPGA development board is used including Spartan 3 FPGA, external interface, and peripheral devices, such as UART, PS2, VGA, etc. Table 1 shows the summary of implementation using Xilinx Spartan 3 FPGA. The results show about 40% of logic utilization and maximum performance of 32 MHz frequency.

Table 1. Summary of SoC platform implementation in FPGA

Device	XC3S1000
Slices	3347 (43%)
LUTs	6034 (39%)
Minimum period	31.524 ns
Min setup time	7.757 ns
Min hold time	10.605 ns
Max frequency	31.722 MHz

B. ASIC implementation

RTL of proposed platform was synthesized with Synopsys Design Compiler using the Magnachip 0.18 ㎛cell library under the worst-case condition and most conservative wire-load model [15]. The system clock frequency is approximately 91 MHz. The synthesized netlist was passed through Synopsys Astro placement and routing tool. Parasitic R, C values were extracted using the Synopsys Star-RCXT from final design implementation, and post-layout simulation was performed using the Modelsim simulator.

Fig. 15 is the layout created from result of placement and routing using Astro tool and Table 2 shows the summary of final ASIC design. Memory macro blocks are surrounded with hard blockage for on chip cache, MMU, and on chip SRAM are placed on the side of core. Marked part by A, B, C, D, E, and F indicates name in which it use. I/O pad of core is eleven, and that is system clock and reset, UART, JTAG ports respectively. Rest pads are power pad for supplying of core and IO power.

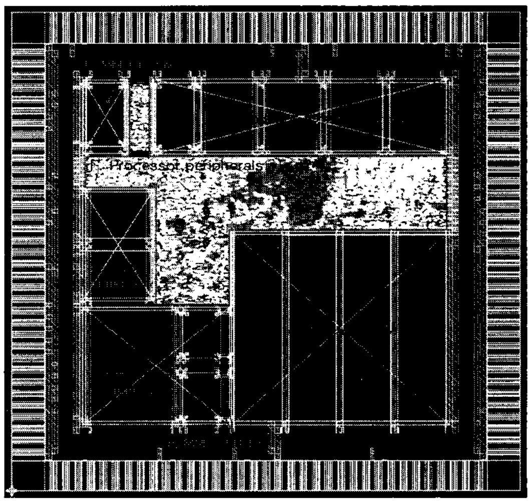


Fig. 15 P&R result of SoC platform

Table 2 indicates the specification of SoC platform layout. Completing the placement and routing, operation frequency is 91 MHz that is met to 8 % lower constraint than the synthesis level.

Table 2. Summary of SoC platform implementation in ASIC

Technology	Magnachip 0.18㎛ CMOS
Cell Library	M18GM180S
Synthesis frequency	96 MHz
Routed frequency	91 MHz
Area	2.6mm^2 core-region out of 4mm^2
Supply voltage	1.8V Core, 3.3 V I/O
Memories	8K byte I/D cache, I/D MMU, 8K on-chip SRAM

VI. SUMMARY AND CONCLUSIONS

In this paper a synthesizable SoC platform based on Open-Source cores has been presented. The platform is very flexible due to easy configuration through a system configuration file and is reliable because all designed

SoC and IPs are verified in the various test environments. The platform has been implemented on FPGA development board with Spartan 3 XC3S1000 FPGA. OpenCores IPs are analyzed and verified at 3 level verification environments and then integrated into our platform. We have used our tool-chain to test our design and develop software on the processor, and we have obtained encouraging results. The system clock frequency of this platform is about 32MHz in the FPGA implementation and 91 MHz in ASIC.

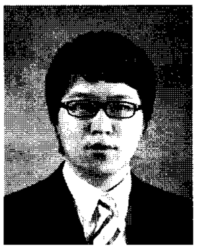
As future work we plan to revise our platform in order to develop more configurable, stable, and variable applications. For accomplishment of this purpose, various system IPs will be designed and integrated to our SoC platform, such as AC97, PS2, USB 2.0, Ethernet controller, H.264 codec. In addition to automation of system reconfiguration process is also required.

ACKNOWLEDGMENT

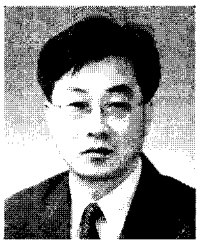
This paper was partly sponsored by ETRI SoC Industry Promotion Center, Human Resource Development Project for IT SoC Architect. Also, CAD tools which used in this work are supported by IDEC.

REFERENCES

[1] Zhihui Xiong, Sikun Li, Jihua Chen and Dawei Wang, "A Platform-based SoC Hardware/Software Co-Design Environment", ICCSC, 2003
[2] Han Qi, Liang Yu, Wei Tong li, "IP-Based SoC Design Methodology", ICDA 2000, August 13, 2000
[3] Diana Wu, Platfrom-based SoC design comes of age, Faraday Technology Corp, April 20, 2003
[4] Mentor Graphics, http://www.mentor.com
[5] Y. Xie, J.Xu, and W. Wayne, "Augmenting platform-based design with synthesis tools", Circuits, Systems, and Computers, 2003, 12, pp.125-142
[6] Jacob Gorban, UART IP Core Specification, Rev. 0.6 August 11, 2002
[7] Igor Mohor, SoC Debug Interface, Rev. 3.0 April 14, 2004
[8] Richard Herveille, VGA/LCD Core Specification, Rev. 2.0 March 20, 2003
[9] Damjan Lampret, OpenRISC1000 Architecture Manual, January 28, 2003
[10] Damjan Lampret, OpenRISC1200 IP Core Specification Rev. 0.7, September 6, 2001
[11] Richard Herveille, WISHBONE SoC Architecture Specification, Revision B.3, September 7, 2002
[12] Richard Stallman, Debugging with GDB, Rev. 9, June 2002
[13] M. Bolado, "Platform based on Open-Source Cores for Industrial Applications", IEEE Computer Society, 2004
[14] Xilinx, Spartan-3 FPGA Family Data Sheet, http://direct.xilinx.com/bvdocs/publications/ds099.pdf
[15] Magnachip Semiconductor, LTD. 0.18-Micron 1.8V Standard Cell Library Datasheet, June, 2005



Younghoon Bin was born in Taejeon, South Korea in 1981. He received a B.S. degree in Computer Engineering in 2006 and M.S. degree in Information and Communication Engineering from Hanbat National University in 2008, respectively. He is currently Ph.D. candidate in Information and Communication Engineering at Hanbat National University. He has joined Silicon Works Co., Ltd. In 2008 and has been engaged in the research and development of timing controller for LCD display device. His current interests include image processing and display SoC design.



Kwangki Ryoo is an Associate Professor of the Department of Information and Communication Engineering at Hanbat National University. He was born in Konju, South Korea in 1964. He studied at Hanyang University, Seoul, Korea where he received his B.S., M.S. and PhD in Electronic Engineering in 1986, 1988, and 2000, respectively. From 1991 to 1994, he was the full-time lecturer in the department of Electronic Engineering at Korea Military Academy (KMA), Seoul, Korea. From 2000 to 2002, he worked at System IC Design Team at Electronics and Telecommunications Research Institute (ETRI), Taejeon, Korea. His current research interests include system chip design, SoC platform design and verification, hardware/software co-design and co-verification, and multimedia codec design.