Implementation of a Flexible Development Platform for Simultaneous Support of Software and Hardware Development Flow

Ki-Yong Ahn, Seonpil Kim, Jae-Moon Kim and Chong-Min Kyung
Department of EECS
Korea Advanced Institute of Science and Technology,
Guseong-dong, Yuseong-gu, Daejeon, 305-701, Korea
E-mail: {ahnky, spkim, jmkim, kyung}@vslab.kaist.ac.kr

Abstract—The biggest problem with SoC design is that there are two distinct heterogeneous development environments for hardware and software; Software engineers use software development tools such as compilers and debuggers to develop software codes for processor cores. Hardware engineers use traditional HDL development tools such as logic synthesizer and HDL simulators. Practically, there is no unified verification environment that encompasses both software and hardware domains. This paper describes design and implementation of a flexible development platform which supports software development flow and hardware development flow at the same time. The development platform is composed of prototyping hardware with processor core(s) and a simulation accelerator. The prototyping hardware provides a software development environment and the accelerator provides hardware development environment. They are interconnected with a bidirectional channel for synchronization.

I. INTRODUCTION

Present day, the design complexity of the system-on-chips (SoCs) is extremely increasing. Besides time-to-market pressure is also increasing. So, hardware/software co-verification platform is important to reduce SoC design time.

In a traditional design methodology, hardware and software design takes place in isolation with the hardware being integrated with the software after the hardware is fabricated. Bugs that cannot be fixed in software lead to costly re-fabrication and can adversely affect time-to-market [1]. But using co-verification platform, software development is accomplished at the same time with hardware development. So bugs can be fixed in early design stage.

There are few co-verification platforms. Among the few are co-simulation tools and hardware prototyping boards. The former models the operation of software and hardware blocks with Instruction Set Simulator (ISS) and HDL simulators, each. But, the simulation speed is painfully slow, which limits its practical use. The latter provides physical development environments for both software and hardware with actual processor cores and FPGAs. But, in reality it focuses on the development of the software and lacks debugging features of hardware.

In this paper, we propose a flexible development platform which supports software development flow and hardware development flow at the same time. Our platform is composed of two parts. Prototyping hardware provides a software development environment and a simulation accelerator provides a hardware development environment. They are interconnected with a bidirectional channel for synchronization.

The rest of this paper is organized as follows. In section II, we introduce some previous works. In section III, we present our proposed platform. Finally in section IV, an example of implementation for JPEG decoding system architecture is given.

II. PREVIOUS WORKS

A. Co-simulation

One of the basic co-verification platforms is co-simulation. In co-simulation, all part of the SoC is simulated by software tool. Generally, ISS which models processor and HDL simulator which models other IP blocks are used. Software part of SoC is simulated in the ISS and hardware part of the SoC is simulated in the HDL simulator. Data communication between ISS and HDL simulator is accomplished with co-simulation interface like inter process communication (IPC) method for synchronization [2]. Figure 1 shows this concept of co-simulation.

This co-simulation method is easy to use and convenience to debug because it is only software platform. But for the same...
reason, the simulation speed is painfully slow. HDL simulation time is very slow for complex design and communication overhead via IPC is very large. So, co-simulation method is adequate only a small system. Complex SoC system can’t use the co-simulation method.

Co-emulation method is appeared to improve the speed of co-simulation. In co-emulation, simulation accelerator is used for verification instead of HDL simulator.

B. Co-emulation

Co-emulation platform has both software platform and hardware platform. The hardware platform of the co-emulation is simulation accelerator. Simulation accelerator is an FPGA base emulation system. Calculation intensive part of the design is calculated in the FPGA of simulation accelerator for acceleration. In the software platform of the co-emulation system, it may be HDL simulator, ISS or C/C++ program. Generally higher abstraction level testbench is used for acceleration. Software part of the platform and hardware part of the platform are communicated with the co-emulation interface for synchronization. It is composed of API, device driver and physical channel like PCI bus [3]. Simulation accelerator software supports convenient debugging feature for hardware block in FPGA. And interface block between software and hardware is automatically generated. Figure 2 shows this concept of co-emulation.

In co-emulation, simulation speed is accelerated by higher abstraction in software part and hardware acceleration in hardware part. So simulation time of co-emulation is faster than co-simulation. But it is also slow to complex design because of software part. For more complex verification, the lower abstraction level of testbench is needed. Then, there is no more acceleration of software part. At this time, fully hardware platform is needed.

C. Hardware prototyping

Hardware prototyping platform is a fully hardware platform and simulation speed is fastest. It is composed of actual processor core and FPGA. Embedded software is simulated in the processor core and other hardware block is simulated in the FPGA. The software part of the design is easy to debug. In Circuit Emulator(ICE) supports source level debugging feature of embedded software. It can also control the processor core. But the hardware part of the design has lacks of debugging feature. Hardware debugging method is just FPGA debugging method. Logic analysis system can be used but it is hard to debug. So it can be said that hardware prototyping platform focuses on the development of software.

Now, we propose a flexible development platform which supports software development flow and hardware development flow at the same time. Development of software part is using an actual processor core that is the same as previous hardware prototyping platform. But development of hardware part is using simulation accelerator that is the same as co-emulation instead of FPGA in previous hardware prototyping platform. The processor core and the simulation accelerator are interconnected with bidirectional channel. The prototyping core with ICE provides a convenient debugging feature for software debugging and the simulation accelerator provides a convenient debugging feature for hardware debugging. Verifying hardware part of the SoC system with FPGA of simulation accelerator, Designer can easily debug his design such as HDL simulation debugging environment. Figure 3 shows the concept of previous hardware prototyping platform and our proposed platform.
Our proposed platform is composed of two parts. It is prototyping hardware with processor core and a simulation accelerator. We named ‘ProBase’ to our prototyping hardware. The prototyping core is plugged to the ProBase, and it is connected to the simulation accelerator with a bidirectional channel. In ProBase FPGA, System bus and bus components are implemented. And there are some peripherals like UART, external I/O port and memories are also implemented. We also implemented ‘bus splitter’ component for bidirectional channel. Physically there are two separate system buses in our platform. One is in the simulation accelerator and the other is in the ProBase. But logically there is only one system bus by bus splitter component. In other words, bus splitter component virtually makes two system buses to only one system bus. Detailed description about bus splitter will be mentioned in next subsection. IP under debugging, which is hardware part of the SoC is verified in the simulation accelerator. Figure 4 shows this structure of our proposed platform. ‘Optional IP Under debugging’ in the ProBase FPGA means that some part of the hardware of SoC can be verified in the ProBase FPGA. Of course, most part of the hardware is verified in the simulation accelerator.

B. Debugging environment

Using our platform, software part of SoC is executed in the prototyping core in the ProBase and hardware part of the SoC is executed in the FPGA of simulation accelerator. Because of using real core and real hardware, the simulation speed is faster than previous co-emulation. Debugging of software is using ICE via JTAG port [4]. In the debugging mode of prototyping core, ICE can control the operation of the processor and can trace state of the processor. Source level debugging is possible with ICE. Debugging of hardware is using feature of simulation accelerator. Conventional FPGA debugging method is using logic analysis system. But it is required an additional equipment. In simulation accelerator, it is internally implemented powerful debugging feature without additional equipment. Internal node and register value can be probed with this feature. Designer can debug SoC easily like using HDL simulator.

C. Implementing bidirectional channel

To connect ProBase to simulation accelerator we use a bidirectional channel. But this channel is an additional component to verify. Target SoC system doesn’t have this channel. So, we implement bus splitter to logically eliminate this channel. There are two system buses in the simulation accelerator and ProBase each. But, by bus splitter component, there is only single system bus in designer’s point of view.

Bus splitter is implemented by time sharing fashion. Whole signal count of the system bus is about 200 signals. But width of the bidirectional channel between simulation accelerator and ProBase is 54 signals. So, transferring of whole system bus signal takes 4 cycles. We made communication clock, that is 5 times faster than system clock, for bidirectional channel. Because an additional one cycle is needed for synchronization, splitter clock is 5 times faster. In ProBase, system clock and splitter clock are generates by oscillator. These two clocks have the same phase. But simulation accelerator doesn’t have clock generation unit. Simulation accelerator gets splitter clock from ProBase and makes system clock from this splitter clock and synchronization pulse. Synchronization pulse is generated in every five cycles of splitter clock.

To implement this function, we use simple counter. A simple 3-bit counter is used. This counter is operating with splitter clock and resetting synchronously with synchronization pulse. The counter is updated in every cycle. So, the counter has value of zero to four as shown in Figure 5. When the
synchronization pulse is high, the counter value is four and reset to zero in next cycle. Using this counter value, simulation accelerator makes system clock. The system clock is high when the counter value is four or zero. And the system clock is low when the counter value is the others, one, two and three. Also, Duty cycle of this system clock is not 50%. But the duty cycle does not matter to almost SoC systems. System bus signals are transferred by four cycles when the counter value is not three.

IV. EXAMPLE OF IMPLEMENTATION

We adapted our development platform to the JPEG decoding system [5]. JPEG decoding system is composed of three blocks - JFIF, VLD and IDCT. We partitioned these three blocks in hardware and software. JFIF block is executed by software code and VLD and IDCT blocks are executed by hardware IP. We use ARM 946E-S as a prototyping core and iPROVE as a simulation accelerator [6]. Figure 6 presents the JPEG decoding system. JPEG image is generated by PC camera in the host PC. And this image is transferred to the ProBase system by UART serial communication. In ProBase platform JFIF, VLD and IDCT is executed. After whole decoding process, decoded image is transferred to the host PC by UART serial communication. In the host PC, this image is compared to the result from reference decoding algorithm.

Figure 7 shows the experimental environment of JPEG decoding system. Simulation accelerator is plugged in the host PC by PCI bus and it is not shown in this figure. Host PC and ProBase is connected with bidirectional channel, ICE and UART. For using logic analysis system, micror type connector is also connected.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed a flexible development platform for simultaneous support of software and hardware development flow. Using previous method, most platforms focuses on only one development flow. If it supports software and hardware simultaneously, it is slow and hard to adapt complex SoC system design. Our platform composed of prototyping hardware with prototyping core and simulation accelerator. Software part of the SoC is simulated in the processor core and hardware part of the SoC is simulated in the simulation accelerator. Using ICE and simulation accelerator software, debugging is easy to use and simulation speed is fast.

We adapt our platform to JPEG decoding system. Some block of system is executed in the software code and the other block is executed in the hardware IP. It works well, and our platform is verified.

Future work will include as expansion of our system to multi-core environment. In this time, our platform support only ARM processor and AHB system bus. But we are working for supporting more various processor core and system buses for example, many DSP cores and AXI buses.

REFERENCES

Paper ID : paper_317

Paper Title :
Implementation of a Flexible Development Platform for Simultaneous Support of Software and Hardware Development Flow

Authors : Ki-Yong Ahn, Seonpil Kim, Jae-Moon Kim and Chong-Min Kyung

Key Words
SoC, SoC Verification, SoC Development Platform, Prototyping, Co-verification, Hardware/Software Co-design