Co-development of Media-processor and Source-level Debugger using Hardware Emulation-based Validation

Yeon-Ho Im, Sang-Joon Nam, Byoung-Woon Kim, Kyong-Gu Kang, Dae-hyun Lee,
Jin-Hyuk Yang, Young-Su Kwon, Jun-Hee Lee, and Chong-Min Kyung

VLSI Systems Lab., Dept. of Electrical Engineering,
Korea Advanced Institute of Science and Technology, Korea

VLSI Systems Lab., ChiPS Bld., Dep. Of EE, KAIST, 373-1 Kusong-dong,
Yusong-gu, Taejon 305-701 Korea

E-Mail : mini@duo.kaist.ac.kr
Phone : +82-42-869-4402 Fax : +82-42-869-4410

Abstract

To exploit the development system with a media-processor as soon as possible, the co-development and co-validation of them is very important. We describe the co-development of a media processor and its source-level debugger, which is necessary to develop multimedia systems. Even though a real chip is available, it takes a long time to validate the functionality of the source-level debugger. Here in this paper, we have adopted hardware emulation for a fast development and validation before a processor is fabricated.

1 Introduction

As multimedia systems become more and more complicated, their functions are implemented as software rather than hardware now a day. Because the software approach makes it easier to increase the flexibility of the whole system and to deliver the product to market as soon as possible with fast development, easy debugging, and easy upgrading. To meet these demands, media-processors and microprocessors support high-level programming languages such as C and the source-level debugging system, which allows a program developer to follow any execution path of the complex programs in source level and access inside states of processor such as register value.

Furthermore, many real-time programs are required to operate on target hardware with complete validation [1]. Two approaches have been used for debugging system, hardware-based debugging system and software-based debugging system. A hardware-based debugging system is capable of operating in the actual hardware environment considering interrupts, PCI (Peripheral Component Interconnect), and several interfaces, while software-based debugging system is poor at handling such a system-level signals. Also, hardware-based debugging system makes it possible for a programmer to debug target programs at full speed in real environment.

There are two implementations, JTAG emulator and ICE (In-Circuit Emulator), for a hardware-based source-level debugging system. JTAG emulator is based on the boundary scan techniques as defined by IEEE 1149.1[2]. ICE uses a special bond-out processor to observe internal states of the target processor precisely. While ICE adopts an expensive bond-out hardware to observe and control the processor, JTAG emulator requires an additional small control logic and five testing pins defined by IEEE 1149.1.

Until now, the source-level debugging systems have been implemented and verified after the fabrication of processor[3]. In terms of time-to-market strategy, it is a great disadvantage. Because hardware emulation can be considered as "the first silicon" as soon as RT-level design is available, the whole system can be designed and verified concurrently at the earlier stage.

In this paper, we propose a methodology for co-development of a media processor and its source-level debugging system based on hardware emulation. A proposed method needs co-validation of processor and debugging system. And two methods have been used for co-validation, simulation-based and emulation-based validation[4]. The comparison between a previous method and co-validation is shown in Table 1. The choice between them depends on the complexity of the target system being designed. If the design complexity is rather low and there is enough time to check the functionality, simulation-based approach would be adequate due to its high observability. However, to design a complex system, it is necessary to run a lot of testing programs on a processor model, which can be either software or hardware model. In such case, emulation-based approach is proper because it takes too long to run necessary programs on the software model [3,5,6,7]. Because the design of source-level debugging system is as complex as that of the processor, we adopted emulation-based method as validation for co-development of them.

methodology	Previous approach	Co-validation approach	
		Simulation-based	Emulation-based
Validation engine	A fabricated target chip	A microprocessor	FPGAs
Sped	10 ~ 300 MHz	5 ~ 100 Hz	100 ~ 1000 KHz
Validation period	After fabrication	Before fabrication	Before fabrication
Interrupt handling	Good	Poor	Good
Time-to-market	Long	Short	Short

Table 1. Comparison of validation methodologies

0-7803-5727-2/99/$10.00 © 1999 IEEE

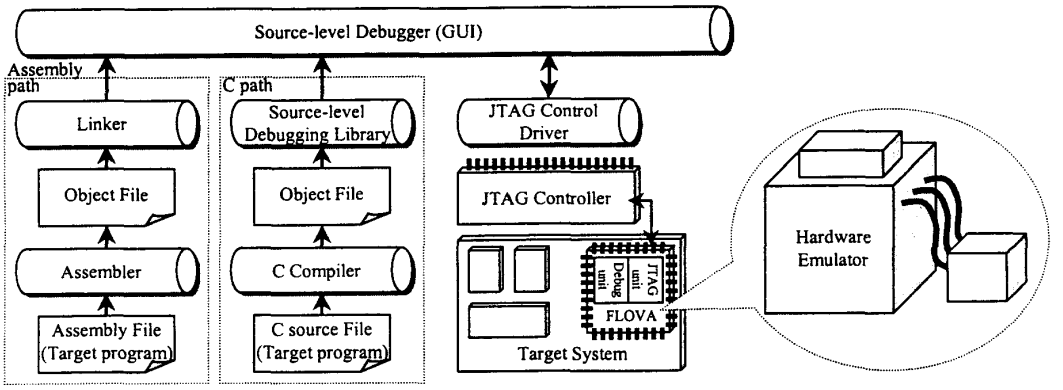

Fig. 1 Source-level debugging system of FLOVA consisting of a software and a hardware part. A software part is composed of GUI, Source-level debugging library which support information on C source File and assembled code, JTAG Control driver, on-chip JTAG unit and debug unit. C source files and assembly files are converted into object files via assembler and C compiler respectively.

We have developed the processor, FLOVA (FLOating-point VLIW Architecture), and its source-level debugger targeted for image processing and 3D graphics [8]. (Also, we have developed assembler, linker, scheduler, instruction-set simulator, and C compiler for FLOVA.)

The organization of this paper is as follows. In section 2, overview of FLOVA is described. Source-level debugger is described in section 3. And then, hardware emulation environment for co-development is explained. In section 5 and 6, results and conclusions of the proposed co-development methodology are presented.

2 Overview of FLOVA

FLOVA has an architecture of VLIW (Very Long Instruction Word). It can issue four instructions per cycle and executes them with 13 functional units. It consists of a branch/program control unit, three integer ALU's, an integer multiplier, a shift unit, two load/store units, three floating-point units which support IEEE 754, and a media ALU and multiplier which executes sub-word instructions just like MMX instructions of Intel processors. And it has an instruction cache of 8KB and two data memories of 16KB and several peripherals like DMA, PCI, TIMER, and so on. Its speed is 100 MHz.

The register file with 64 entries has eight read-ports and four write-ports and can be accessed as 32-bit or 64-bit register. When sub-word instructions of a media ALU and multipliers are executed, register file is access as 64-bit, while integer operation is executed, it is accessed as 32-bit. The floating-point multiplier has fast lighting operation for vertex lighting and the twin and cross mode operations for effective 4×4 matrix multiplication in geometry stage of 3D graphics. A lighting operation is to calculate the intensities of 3 points of a polygon given the position and normal vector of light and polygon in 3D graphics. Using a fast lighting operation, FLOVA can accelerate the lighting calculation significantly. Using twin and cross mode operations of floating-point multiplier, FLOVA is capable of enhanced geometry transformation such as translation or rotation more efficiently compared to traditional DSP's and VLIW's.

FLOVA support simple predicated execution. A branch/program control unit compares register values and set the predicated bit in it according to specified condition in the instruction. In FLOVA instruction, there are 2-bit predicated bits. The predicated bit is set to be true if the condition is satisfied and false otherwise. If the predicated execution bit is true, only the instructions that have predicated bits of 2'b01, are executed. In the case of the predicated bit of false, only the instructions with the predicated bit of 2'b00 are executed. Instructions supposed to always executed are with 2'b11. A branch/control unit can treat assembly-level FOR instruction. (up to four nested FOR instruction). In addition, it treat interrupts to response to a stimulus from outside. A programmer can set the interrupts to be edge/level and also to be proper interrupt-level between 0 and 2.

3 Source-level Debugger of FLOVA

Fig. 1 shows source-level debugging system of FLOVA. It is composed of software and hardware parts. The software part consists of JTAG control driver, source-level debugging library and a source-level debugging program that is implemented in GUI on WINDOWS 95. The hardware part of source-level debugging system consists of on-chip debug and branch units.

Using the GUI of the source-level debugging system enables a programmer to download a target program into a target system, set the breakpoints, run and stop FLOVA to observe the states of the target system.

3.1 Source-level Debugger

The source-level debugger is an advanced interface that helps programmers to develop, test, and modify a target program easily. It downloads the target program, sets breakpoints in the target program and observes the internal states of FLOVA and a target system. The internal states of a target system can be accessed by

FLOVA instructions. Also, it can stop, restart, and step the target program.

A source-level debugging library and a JTAG control driver support these debugging operations (stopping, restarting, stepping, and setting breakpoints). A source-level debugging library offers the debugger an information on the register index or memory address of a variable used in a C source program. And all operations are supported by the JTAG control driver, through which the debugger can access the debug registers in the on-chip debug unit.

3.2 Source-level Debugging Library

Because an information about register index or variable is lost when C source program is compiled, to set the breakpoint at arbitrary point in C source program, a relation between C source program and object code must be known. When a programmer sets the breakpoints in C source program using GUI, source-level debugger asks source-level debugging library the position in object code corresponding C source program.

When an assembly program is used as a target program, the source-level debugging library does nothing.

3.3 JTAG control driver

JTAG control driver supplies these debugging operations to the on-chip debug unit through JTAG ports. Also, it can write an instruction into the debug instruction register (DIR) in the on-chip debug unit and read the content of the debug data register (DDR) in on-chip debug unit.

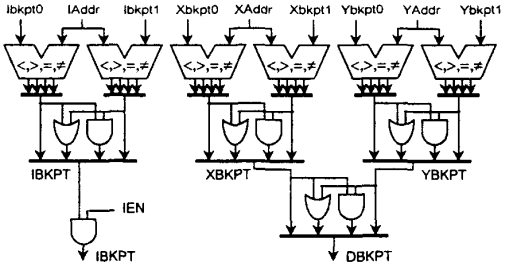

Fig. 2 A break-point logic for a program address and two data addresses. Several conditions are decided by the values which are fed into top-most resisters through JTAG.

3.4 On-chip Debug Unit

The on-chip debug unit is controlled through JTAG ports standardized as IEEE 1449.1. Its functions are to stop/resume/run single-step/set breakpoints/access DIR and DDR. The source-level debugger can also set breakpoints on two program addresses and four data addresses, where several conditions (=,≠,<,>) can be checked for each breakpoints. Several conditions are checked by the breakpoint logic as shown in Fig. 2 .

3.5 Branch Unit

If target program executes within the address area specified by several conditions in the breakpoint logic, a HDE (Hardware-Debug-Event) interrupt occurs to stop the normal execution of a program and to service the corresponding interrupt service routine that is a special kind of routine whenever the interrupt occurs.

In the debugging mode, FLOVA executes NOP (no-operation) instructions waiting for an instruction from the debug unit. And then, a source-level debugger writes a FLOVA instruction into DIR in the on-chip debug unit through JTAG ports. The FLOVA instruction filled in DIR can be executed at proper time. The internal states of FLOVA can be written by such an instruction as "*add r10,r10,1*", and read by such an instruction as "*mvtos DDR, r10*". Fig.3 shows a logic for a single instruction execution.

By these operations, FLOVA supports debugging functions

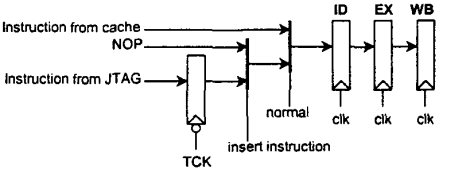

Fig. 3 A logic for normal and debug mode. In normal mode, instruction from instruction cache is executed. While in debug mode, instruction from JTAG is latched in DIR executing NOP and execute the instruction in DIR.

4 Hardware Emulation Environment of Co-development

Generally, it is hard to detect the timing errors through hardware emulation. But, because its speed is fast and only 100 times slower than a real chip, hardware emulation is used to verify the functional correctness mostly and it becomes one of standard techniques for verification.

We used a QUICKTURN of M3000 class as a FLOVA model which allows a maximum capacity of 3 million gates and a speed of 1000 FLOVA instructions per second [9]. The software of it includes partitioning and mapping on the different FPGAs. Also, there is an integrated instrument available to capture and to progress real-time data for debugging. M3000 model especially supports flexible memory emulation. Small memories can be compiled into the internal RAM of the FPGAs. In addition, a special equipment, TIM (Target Interface Module), connects the QUICTURN with the target system. It enables complex ASIC functions such as micro-controllers, and peripheral controllers.

5 Results

We first verified the function of FLOVA using the emulator and developed a simple target system. And then, we also checked the operation of whole system with a lot of test vectors and programs. Fig. 4 shows co-development environment of an emulator and a simple target system. With the speed of emulator, we could validate the functionality of the source-level debugging system easily. If a general RTL simulator, which could

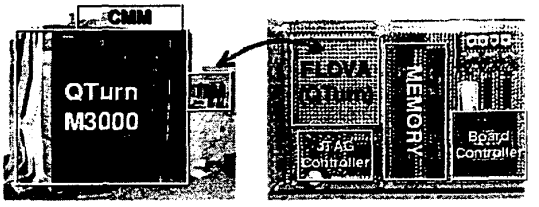

Fig. 4 Co-development environment of FLOVA source-level debugging system.

execute less than 10 FLOVA instructions per second, were used as a FLOVA model, it would have taken a long time to validate the source-level debugging system.

Fig.5 shows an example of a source-level debugger GUI. It represents information such as source program, breakpoints, register values, and memory values.

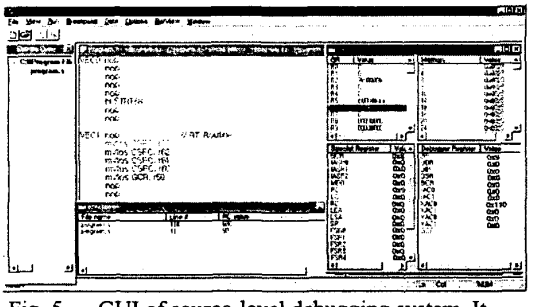

Fig. 5 GUI of source-level debugging system. It consists of a source-program window, a breakpoint window, a register window, and a memory window.

6 Conclusions

In this paper, we described the co-development methodology of media-processor and source-level debugging system that is essential to develop an application system using emulation-based validation technology. And we adopted hardware emulation for a fast development, because it takes a long time to validate the operation of the processor as well as the source-level debugging system. Using this methodology, we have validated the whole system successfully even before a fabrication of FLOVA. It is necessary to use a co-development methodology based on hardware emulation to fulfill both a short product time and a reduced time-to-market.

ACKNOWLEDGEMENTS

This work was performed as a part of ASIC Development Project supported by the Ministry of Trade, Industry & Energy, the Ministry of Information and Communication, and the Ministry of Science and Technology.

References

1. R.A.Gott, "Debugging embedded software," Computer Design. pp.71-78, Feb. 1998.
2. IEEE, "IEEE Standard Test Port and Boundary-Scan Architecture" IEEE, 1990.
3. J.Kumar, N.Strader, J.Freeman, and M.Miller, "Emulation Verification of the Motorola 68060," Internal Conference on Computer Design, pp.150-158, Oct.1995.
4. G.J.Bunza, "The Impact of Hardware/Software Co-development on Desing Process Methodology: Big Changes and Bigger Success," Proc. Design, Automation and Test in Europe, pp.37-41, Feb.1998.
5. J.Gateley et.al, "UltraSPARC-I Emulation." Proc. Design Automation Conference, pp.13-18, June 1995.
6. J.Monaco, D.Holloway, and R.Raina, "Functional Verification Methodology for the PowerPC 604 Microprocessor," Proc. Design Automation Conference,1996.
7. G.Ganapathy, R.Narayan, G.Jorden, and D.Fernandez, "Hardware Emulation for Functional Verification of K5," Proc. Design Automation Conference, pp.315-318, June 1996.
8. S.J.Nam et.al, "VLIW Gerometry Processor for 3D Graphics Acceleration," International Symposium on Low-Power and High-Speed Chips (COOL Chips), pp.107-120, Apr, 1999.
9. Quickturn Design Systems, Inc., "System Realizer User's Guide Version 5.0."