Efficient Prototyping System Based on Incremental Design and Module-by-Module Verification

Yongjoo Kim, Youngsoo Shin, Kyuseok Kim, Jaehee Won, and Kiyoung Choi

ISRC, Seoul National University
Seoul, Korea 151-742
Phone: +81-2-880-6768
Fax: +81-2-887-6575
E-mail: kchoi@azalea.snu.ac.kr

ABSTRACT

This paper presents an efficient hardware prototyping methodologies of digital systems. It is based on a low-cost and flexible prototyping system which consists of a general-purpose CPU and a FPGA-based custom board. Using our prototyping methodologies such as incremental system design and module-by-module verification, we can map partial system specification into hardware prototype, which is implemented by programming FPGAs on custom board. This allows flexible and efficient system verification as well as reduction in cost and time of prototype building.

I. INTRODUCTION

In developing digital systems such as ASICs and custom chips, it is necessary to simulate the systems completely before committing ASICs and custom chips to silicon. Simulation, however, can be a bottleneck in overall system development cycle with the rapid increase in the complexity of digital systems. Prototyping can be another efficient system verification method although prototyping itself is a design goal in the initial stage of system development. With current trend of ever-increasing system size and ever-decreasing system development time, rapid prototyping is becoming an important design issue in system development.

When prototyping a system, it is necessary to use a prototyping medium which is inexpensive, flexible, easy to build, and with comparable performance to target fabrication technology. Field-programmable gate array (FPGA) is an ideal prototyping medium because of its field-programmability. Using FPGA as prototyping medium, a class of methods is emerging called computer-aided prototyping(CAP). is emerging. CAP combines synthesis and verification softwares with FPGA technology to automatically produce hardware prototypes of chip designs. The CAP environment can be made much more powerful by adding the capability of concurrent verification of all system aspects - including hardware, software, and external interfaces - leading to higher quality products and shorter time to market [1].

There are several commercial or research prototyping systems such as RPM emulation system [1], Diodes system [2], and DSP system with multiprocessors [3]. They are very expensive or limited in application and flexibility. Although simulation compiler system [4] is similar to our prototyping system, it is not oriented to rapid prototyping of system but oriented to simulation acceleration.

In this paper, we propose a flexible and cost-effective CAP system and some system prototyping methodologies using the system. The CAP system consists of a workstation as a software platform and FPGA-based custom board as target hardware prototype. The methodologies we propose as new approach to system prototyping are as follows:

(1) Incremental system design
Functions which are already defined and verified are implemented with FPGA(s). Functions to be newly verified will be software-simulated concurrently with the FPGA implementation. This process continues until all the functions are implemented.

(2) Module-by-module verification
One or several modules of a system can be implemented with FPGA(s) and the functionality can be checked by the software. Then the modules are replaced with other modules to be checked. This process continues until all the modules to be verified in the form of FPGA implementation are checked.

The rest of this paper is organized as follows. In Section II, we describe our prototyping system which is used as platform for the application of our methodology. In Sections III and IV, our system prototyping methodologies are described in detail. Then after describing some

0-7803-2570-2/95 $4.00

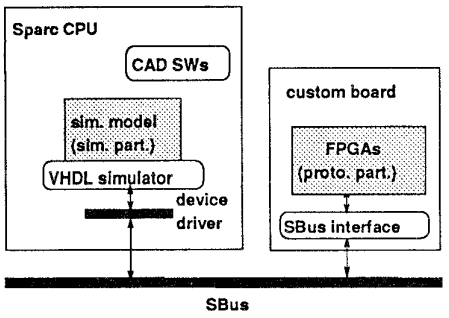

Figure 1: Prototyping system hardware environment

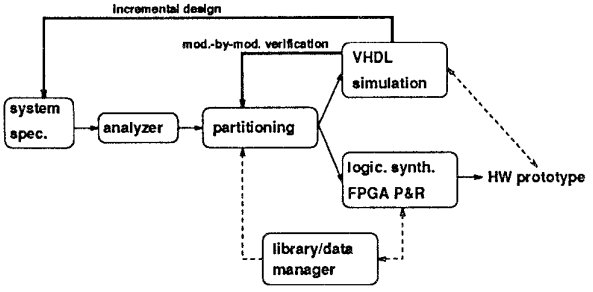

Figure 2: Major steps in system prototyping or softwares in prototyping system

experimental results in Section V, we conclude and give some remarks on future work in Section VI.

II. PROTOTYPING SYSTEM

As shown in Figure 1, the hardware environment of our prototyping system consists of a general purpose CPU(SUN Sparc processor in Sparc Classic workstation) and a custom board. The CPU is in charge of running various softwares such as simulators and tools for synthesis of hardware prototype. The Custom board consists of FPGAs(Xilinx 3090 [11]) and bus interface. The communication between CPU and custom board is done through SBus. To interface between SBus and hardware prototype, we used a SBus DMA Controller chip(LSI Logic L64853A [8]) and some control logic. They are provided as a part of SBus-based development board (Dawn VME Products DPS-1 [7]), which we used for preliminary experiments. The CPU is always the bus master of SBus transactions in our system. To send(receive) data to(from) hardware-prototype, softwares can write(read) data to(from) the device driver program which provides and interface on the software side.

Major steps in system prototyping using our system are as follows(Figure 2): (i)system specification (ii)system partitioning (iii)hardware prototype synthesis (iv)hardware prototype building (v)system verification.

The system prototyping begins from the specification of system to be designed. It is described in VHDL, a standard hardware description language. The input system description is then compiled and checked using VHDL analysis tool. The error-free system description is partitioned into two parts. One part is to be software-simulated and the other part is to be hardware-prototyped with FPGAs.

At present, partitioning is done manually. There are several factors to be considered during partitioning such as the number of usable gates/pins of FPGAs, expected speedup of simulation with resultant partition, communication overhead, etc. The partitioning strategy and feedback path for the next design iteration (indicated as thick lines in Figure 2) depends on the selected prototyping methodology. The feedback path and library/data manager will be described in more detail in later sections.

After partitioning, the hardware prototype for the partition to be hardware-prototyped is synthesized using FPGA synthesis tools(Synopsys Design Compiler [10] for logic synthesis and Xilinx XACT [12] for FPGA placement and routing), and the configuration data to program FPGAs is generated.

Hardware prototype can be built by down-loading the configuration date into FPGAs. The down-loading also occurs through SBus and SBus interface.

System verification is done using an event-driven VHDL simulator specially designed for an interface to physical hardware. Besides normal simulation jobs such as event handling, the simulator is supposed to do following jobs:
- sending and receiving events on interface signals between two partitions.
- translating the events so that they conform to the available data width and protocol of bus transaction and to the format of internal data structure of the simulator. This is essential in maintaining the semantics of events across the heterogeneous partitions.

Input vectors are applied to the software-simulated part. The simulator communicates with the hardware prototype on the custom board. Because part of the design is handled by the hardware prototype, the VHDL simulation is accelerated.

III. INCREMENTAL SYSTEM DESIGN

Our prototyping system provides a powerful incremental system design environment. As functions are added and/or refined incrementally, the whole system design becomes bigger and bigger requiring longer and longer verification cycles. This problem can be solved through incremental prototyping. At any time during the development of a system, functions which are already verified through simulation are synthesized and implemented with FPGAs,

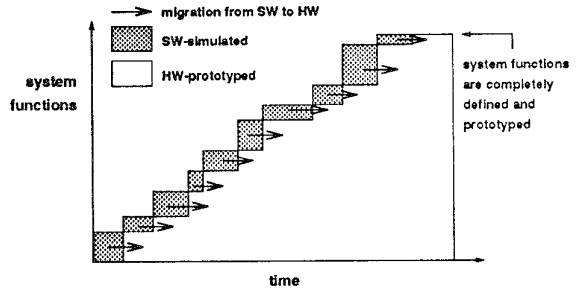

Figure 3: Incremental system design

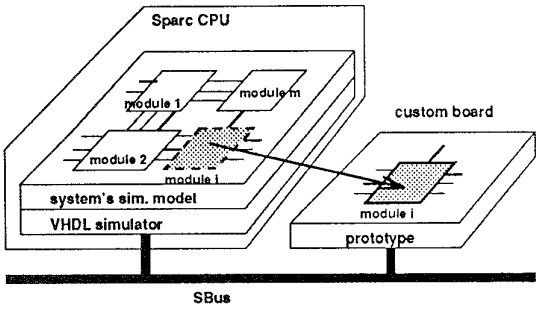

Figure 4: Module-by-module verification(*i=1, 2, ..., m; m = no. of modules*)

thus become a part of hardware prototype since then. Functions to be newly defined, coded in VHDL, and/or synthesized are software-simulated. Because the part put into FPGAs are still there during the simulation, and we need to software-simulate only the incremental part, we can minimize the time spent for simulation. This process continues until all the functions are fully prototyped. Figure 3 illustrates the process. When the development of whole system is finished, full-scale system prototype has also been produced.

In Figure 2, partitioning identifies the incremental part of system functions and regards the part as software-simulated part. Component reuse is assisted by library/data manager which serves as library and manager of data of hardware-prototyped parts of system functions. Hardware-prototyped parts in the library can be instantiated in the next iteration which is indicated by outer feedback path in the figure. Component reuse enables fast incremental prototyping.

IV. MODULE-BY-MODULE VERIFICATION

When the function of a system is already coded in VHDL, some modules in the system can be verified in the form of FPGA implementation with the remaining modules in the form of software-simulated model. This process of module-by-module verification continues until all modules of interest are individually verified. Figure 4 depicts the idea. Assume there are m modules in a system and we want to verify module i in the form of FPGA implementation. The module i is then implemented using FPGAs on the custom board. Remaining modules are simulated as VHDL models using the VHDL simulator. During simulation, module i and the remaining modules communicate through SBus. This concept resembles a hardware modeler which are often used to develop microprocessor-based system but with much higher flexibility in obtaining hardware models. This approach allows us to easily detect and locate prototyping-related problems by the narrowed modulewise design space.

In Figure 2, partitioning is applied to overall system description and selects the module which is not yet verified. Although component reuse is not practiced during the verification iterations as in the incremental system design methodology, verified modules are saved in the library in each iteration. When the verification is finished, all the modules are in library/data manager and ready for full-scale system prototyping just by instantiating them. The verification iteration is represented by inner feedback path in the figure.

V. EXPERIMENTAL RESULTS

To confirm the effectiveness of our prototyping system and methodologies, we performed some experiments. As our example, we chose DCT(discrete cosine transform) core for image processing [9]. It consists of four functional modules: a 16-bit shift and adder(M1), a ROM as a look-up table(M2), a 16 x 14-bit parallel multiplier(M3), and a 22-bit adder(M4).

Important assumptions to simplify the preliminary experiments are as follows:
- Sparc CPU, thus software simulator is the only master of bus transaction.
- The systems to be designed are synchronous circuits.
- Clock signal is applied to the software simulator as an input vector and fed to the hardware prototype via SBus.
- Hardware prototype is fast enough that before the end of the current software-simulation cycle, computation by the hardware prototype for that simulation cycle is finished.

To confirm the effectiveness of our system when applied to incremental system design, we partitioned the description of DCT core into two parts: P1 and P2. Assuming that the partition P2 was designed before P1, P2 was prototyped into hardware and P1 was software-simulated. Table 1 presents the results. We tried two different partitions. In both cases, we achieved considerable improvement in simulation speed over all-software simulation. The speedup heavily depends on the size, abstraction level and characteristics of each partition. In

Table 1: Experimental results: simulation results, hardware size, and VHDL code size (all SW: all modules are simulated, mixed: one partition is simulated, the other partition is prototyped)

	partitioning 1		*partitioning 2*	
	all SW	mixed	all SW	mixed
sim. time(sec)	191	19	196	19
no. signals	5747	1690	6372	1254
no. events	239185	29593	251611	30325
no. gates		3725		5080
no. flip-flops		79		201
no. I/O pads		19		18
no. VHDL lines (SW simulated)	2921	1463	2921	1281

our experimental cases, the hardware partition contains the multiplier which generates many simulation events. That is why we obtained such drastic improvement.

To validate the module-by-module verification as another flexible methodology of system prototyping and verification, we performed another experiments with the same circuit. We verified conveniently and successfully each module in the form of FPGA implementation with the remaining modules in the form of software-simulated model.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a flexible and cost-effective computer-aided prototyping system and efficient and flexible system prototyping methodologies using the system. The efficiency and flexibility of our system and methodologies is due to the field-programmability of FPGAs and the partial module-oriented prototyping of a system instead of full-scale prototyping.

Comparing with other systems and methodologies, the contributing points of our system and methodologies are as follows:

- While existing systems convert a completed system design into a hardware prototype, our methodologies allows partial and incremental development of hardware prototype through incremental system design.
- Our methodology allows system verification through the arbitrary mixture of software-simulated modules and hardware-prototyped modules, whereas existing systems allow only limited mixtures.

Future work will be as follows:

- Extension of our system and methodologies to hardware/software co-design [5] of mixed or embedded systems.
- Development of automatic partitioning softwares.
- Expansion of the prototyping system to cover the systems with larger size. Because this requires multiple FPGA chips on custom board, we need the capability of multi-chip partitioning of system functions.

REFERENCES

[1] S. Walters, "Computer-aided prototyping for ASIC-based systems," *IEEE Design and Test of Computers*, pp. 4-10, June 1991.

[2] R. Hartley, K. Welles II, and M. Hartman, *et al.*, "A rapid prototyping environment for digital-signal processors," *IEEE Design and Test of Computers*, pp. 11-26, June 1991.

[3] M. Engels, R. Lauwereins, and J. A. Pepertstaete, "Rapid prototyping for DSP systems with multiprocessors," *IEEE Design and Test of Computers*, pp. 52-62, June 1991.

[4] K. A. Olukotun, R. Helaihel, J. Levitt, and R. Ramirez, "A software-hardware cosynthesis approach to digital system simulation," *IEEE Micro*, pp. 48-58, August 1994.

[5] J. A. Rowson, "Hardware/software co-simulation," in *Proc. 31th ACM/IEEE Design Automation Conference*, pp. 439-440, June 1994.

[6] *Standard for a Chip and Module Interconnect Bus: SBus (P1496/Draft 2.3)*, IEEE Standard Department, 1993.

[7] *User Guide for DAWN VME Products DPS-1: Development Platform SBus Version 1.0 Revision B*, DAWN VME Products, April 1991.

[8] *L64853A SBus DMA Controller Technical Manual*, LSI Logic, 1991.

[9] J. S. Lim, *Two-Dimensional Signal and Image Processing*, Prentice-Hall, 1990.

[10] *Design Compiler Reference Manual*, Synopsys, Oct. 1991.

[11] *The Programmable Logic Data Book*, Xilinx, 1993.

[12] *User Guide and Tutorials*, Xilinx, 1991.