An Integrated Hardware-Software Cosimulation Environment for Heterogeneous Systems Prototyping

Yongjoo Kim*, Kyuseok Kim*, Youngsoo Shin*, Taekyoon Ahn*, Wonyong Sung+, Kiyoung Choi*, Soonhoi Ha+

* Dept. of Electronics Engineering
Seoul National University
Seoul, Korea 151-742
kchoi@azalea.snu.ac.kr

+ Dept. of Computer Engineering
Seoul National University
Seoul, Korea 151-742
sha@ce2.snu.ac.kr

Abstract — In this paper, we present a hardware-software cosimulation environment for heterogeneous systems. To be an efficient system verification environment for the rapid prototyping of heterogeneous systems, the environment provides interface transparency, simulation acceleration, smooth transition to cosynthesis, and integrated user interface and internal representation. As an experimental example, a heterogeneous system is cosimulated and prototyped successfully, which shows that our environment can be a useful heterogeneous system specification/verification environment for rapid prototyping.

I. INTRODUCTION

Cosimulation refers to the simulation of heterogeneous systems whose hardware and software components are interacting. Traditionally, the task has been performed only after the prototype hardware became available and with the help of in-circuit emulators and/or other techniques [1]. With hardware-software codesign, it is essential to verify correct functionality even before hardware is built. In contrast to the conventional(or homogeneous) simulation of digital hardware, cosimulation should care for the interaction among hardware and software components.

The available techniques for hardware-software cosimulation trade off among a number of factors such as performance, timing accuracy, model availability. In [1], Rowson classified cosimulation techniques into several classes according to the factors. The processor model availability dominates the choice of techniques. Becker, Singh, and Tell [2] performed cosimulation of a network interface unit on a distributed network using Cadence Verilog-XL simulator and Unix socket. They used C++ and Verilog in describing the software and the hardware components, respectively. Their cosimulation is a combination of synchronized handshake and cycle accurate processor model. Thomas, Adams, and Schmit's cosimulation scheme [3] is similar to [2], but their technique is based on synchronized handshake with no processor model. Cosimulation techniques of Poseidon [4] and Ptolemy [5] need pin-level model of processors. Their approaches are most accurate but take much more simulation time.

In this paper, we present a hardware-software cosimulation environment for heterogeneous systems. To be an efficient system verification environment for the rapid prototyping of a heterogeneous system which consists of both hardware and software components, the environment provides interface transparency, simulation acceleration, smooth transition to system prototype synthesis from simulation, and integrated user interface and internal representation. The resultant benefits of those features are as follows: the modularity of cosimulation components, no need of processor models, target architecture independence, and the conceptual simplicity and easiness in establishing and expanding the environment.

The rest of this paper is organized as follows: Section II presents the overview of our cosimulation environment. Sections III, IV, V, and VI describe the details of the environment - interface transparency, simulation acceleration, smooth transition to system prototype synthesis, and integrateduser interface and internal representation, respectively. After describing the application of our cosimulation environment to real system prototyping as an experimental example in Section VII, we conclude with some remarks on future work in Section VIII.

II. HARDWARE-SOFTWARE COSIMULATION ENVIRONMENT

As shown in Fig. 1, the environment has three elements for the execution of cosimulation: a software process running C program, a simulation process executing partial hardware model in VHDL, and a custom board emulating remaining hardware model. Inter-process communication (IPC) routines connect the two processes through socket IPC on a single Sparc CPU. Ptolemy, which is a framework for simulation and prototyping of heterogeneous systems, is extended to provide a user interface and internal representation for system specification and verification.

A. Simulation Levels

The environment supports cosimulation at any abstraction levels. Initially, the specification of heterogeneous system is given in VHDL and C for core hardware and software components. Then simulation models for interface are generated automatically and added to the cores, thereby allowing cosimulation at very abstract level. Fig. 2 represents the abstract level cosimulation. As shown in the figure, the interface simulation models are mainly IPC routine calls with appropriate parameters.

After a target architecture is determined and an interface is synthesized, more detailed simulation models for the interface are generated and inserted, thereby allowing detailed level cosimulation.

B. Software Process

Software process is a process executing a C program which is the software component. Since we use "synchronized handshake" simulation technique [1], there is no need for processor models. The communication between hardware and software is done through a synchronizing handshake implemented using Unix socket [6]. Using this technique, the software can run at the workstation speed even though overall speed will be dominated by the hardware simulator performance.

C. Hardware Simulation Process

Hardware simulation process is a process running a VHDL simulator, which executes a hardware model in VHDL. Our simulator, IVSIM(SNU ISRC VHDL SIMulator), is a VHDL simulator based on an event-driven compiled code simulation algorithm with a concept called modified 'gateways', to improve simulation performance substantially [7]. The simulator is implemented in about 10,000 lines of C++ language (excluding VHDL parser and procedural interface routines of IVDT, SNU ISRC VHDL Development Toolkit [8], which are invoked while data structures are built) and generates C routines for every concurrent statements in VHDL. The generated C routines are linked with C code for the simulation core and then executed for a given test vector.

Another important feature of IVSIM is that it can recognize and support 'foreign' attribute defined in VHDL-93 [9]. The attribute enables a system to be described by not only VHDL but also non-VHDL procedures such as IPC routines in C-language. Using the attribute, if we declare IPC routines as foreign procedures and invoke them at appropriate places in architecture bodies of a VHDL description, they can also be linked with C codes generated for hardware description and simulation core as mentioned above. The resultant executable code simulates the whole hardware component communicating with the software component.

D. Interface

The interface model for simulation is based on Unix IPC,

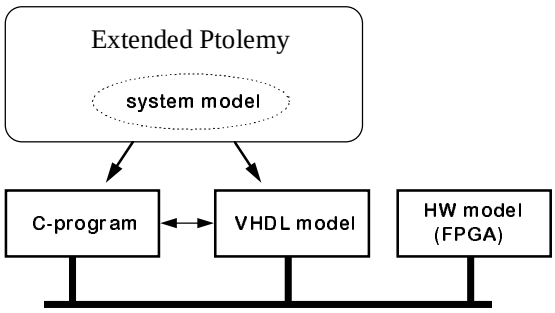


Fig. 1. Cosimulation environment

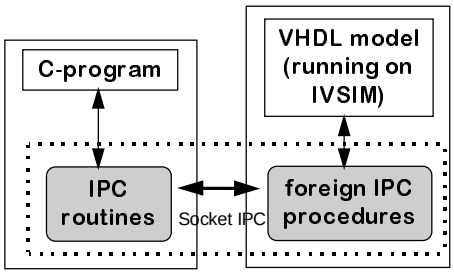


Fig. 2. Cosimulation at the abstract level

regardless of the abstraction levels of cosimulation. The IPC routines are implemented using Unix socket. They are: socket initialization procedure (*init_socket*), socket closing procedure (*close_socket*), socket read procedure (*read_socket*), and socket write procedure (*write_socket*).

III. INTERFACE TRANSPARENCY

Our cosimulation environment provides users transparency about communication interface between software core and hardware core regardless of target architectures and communication protocols. This is especially important for detailed level cosimulation. Once the user selects the target architecture and communication protocol, he or she can concentrate on the functionality simulation of the hardware and software components without having to concern about the details of the interface or communication. To provide the interface transparency in a single processor cosimulation environment, we implemented the following points:

(i) Process modularity - We regard hardware and software components as separate processes(VHDL simulation process and C program process, respectively) running on an identical processor and communicating with each other only through IPC channels during the simulation.

(ii) Automatic interface model generation - Appropriate simulation models(in the form of IPC routines and VHDL models) for the interface between the two components are generated by invoking automatic interface model generator with parameters according to the chosen communication protocol and the target architecture as shown in Fig. 2 and 3.

(iii) Automatic interface model call/instantiation - To simulate hardware-software interface communication correctly, interface simulation models should be inserted at appropriate places in the C program and VHDL model. For complete transparency, IPC routine call insertion and interface VHDL model instantiation in system components is automated through the combination of automatic generation technique and the extension of Ptolemy(described in Section VI). For VHDL description of hardware part, a top-level entity is newly defined to accommodate synchronized handshake using IPC routines. In the top-level entity, IPC routines are declared as foreign procedures and then calls for them are placed within a concurrent process statement which cares for actual synchronized handshake. The core hardware component to be simulated and relevant interface models are also instantiated as component instances within the entity.

Fig. 3 shows the relevant interface elements for hardware part and how they are created(or selected) and combined to provide the overall simulation model of interface, which enables detailed level cosimulation, according to target architecture and communication protocol. If a user gives only the hardware and software cores, the interface elements are automatically created by generators or selected from libraries.

Function or role of each interface element is as follows:
(i) IPC handler takes care of reading/writing data from/to the software process using foreign IPC routines. It handles IPC jobs by handshaking. It also interfaces and translates between IPC routines and channel unit. It is created by a generator.
(ii) Channel unit represents the abstract simulation model of a physical channel device such as DMA controller. It is selected from interface library.
(iii) Decoder/signal register is inserted between hardware core and channel unit to overcome the difference in the width of data transfer and the limitation in the number of pin of hardware prototype device such as FPGA. It is created by a generator.
(iv) Top-level entity acts as a top-level container which gathers all other interface elements. Interface elements are interconnected using component instantiation and signals declared in the entity.

Among those elements, hardware core and decoder/signal register will be mapped into real hardware prototype using FPGA. When standard bus architecture is used, a standard channel device such as SBus DMA controller [10] is used as a physical device for channel unit. If user-defined bus or channel is used, channel unit also should be a part of hardware prototype.

IV. COSIMULATION ACCELERATION

Our cosimulation environment provides a facility to accelerate cosimulation. In cosimulation using synchronized

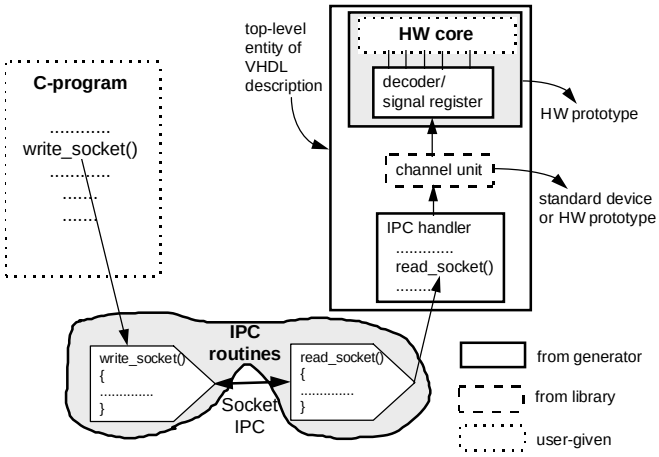


Fig. 3. Interface generation or selection from library

handshake technique, hardware simulation time dominates overall cosimulation time [1]. As hardware subcomponents are added and/or refined incrementally, the whole cosimulation time gets longer. This problem can be alleviated by simulation acceleration through incremental prototyping [11]. At any time during the cosimulation of a heterogeneous system, any hardware subcomponent whose function is already verified through simulation is synthesized and prototyped with FPGAs, thus become a part of hardware prototype afterward. Newly added(or refined) hardware subcomponents(incremental part) are described in VHDL and simulated. Because the part which is already prototyped with FPGAs remains as hardware prototype during the rest of cosimulation of the system and we need to simulate only the incremental part, we can reduce the time spent in VHDL simulation considerably and consequently overall cosimulation time. This process which consists of incremental part definition/simulation and incremental prototype synthesis/addition will continue until the whole function of the hardware component is fully verified. When the cosimulation of whole system is finished, full-scale prototype of the hardware component is already obtained. Fig. 4 depicts the concept of incremental prototyping process.

Fig. 5 shows the execution environment of the accelerated cosimulation. It consists of a general purpose CPU(Sparc processor in a Sparc Classic workstation) and a custom board. The CPU is in charge of running VHDL simulation process for incremental hardware subcomponents, C program process for software component, and CAD tools for the synthesis of hardware prototypes. The custom board consists of a FPGA(Xilinx 4010 [12]) and bus interface. The FPGA is used to implement the hardware prototype. The communication between the CPU and the custom board is done through SBus [13]. To interface between SBus and hardware prototype, we used a SBus DMA Controller chip(LSI Logic L64853A [10]) and some control logic. They are provided as a part of the SBus-based prototype development board(Dawn VME DPS-1[14]) which

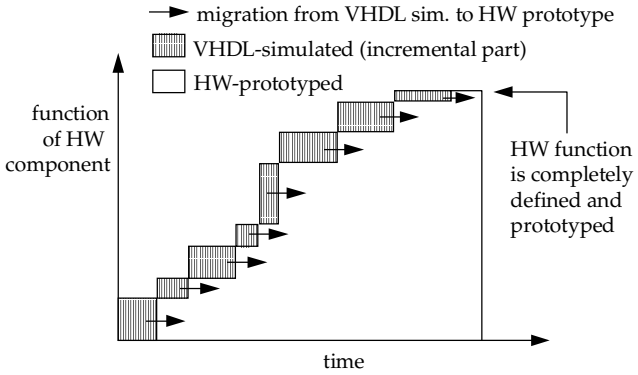


Fig. 4. Incremental prototyping process

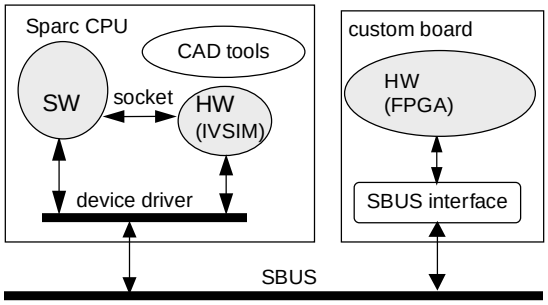


Fig. 5. Execution environment of accelerated cosimulation

we use for preliminary experiments. CPU is always the bus master of SBus transactions presently. To send(receive) data to(from) the hardware prototype, software process and VHDL simulation process should write(read) data to(from) the device driver program. The software and VHDL simulation processes communicate each other through socket IPC as mentioned above.

Current implementation of cosimulation acceleration facility is based on the following assumptions:

(i) Processes in Sparc CPU, VHDL simulation and C program processes, are the only master of the bus transactions through SBus.
(ii) The hardware component is a synchronous circuit.
(iii) Clock signal is applied to the VHDL simulator as an input vector and then fed to the hardware prototype by the simulator via SBus.
(iv) Hardware prototype is fast enough that before the end of the current VHDL simulation cycle which consists of event handling and updating values, etc., computation by the hardware prototype for that cycle is finished.

V. Transition To Cosynthesis

After cosimulation, the system components must be synthesized as the physical components on the selected target architecture. For the cosynthesis, the invokes to the interface simulation models are replaced with the corresponding device driver calls or I/O function calls for the software component. For the hardware component, top-level entity with foreign interface procedure declaration is stripped off and the corresponding interface hardware model is inserted. Since this modification of each component specification for the cosynthesis is very simple and limited to a minimum degree, it is possible to provide a smooth and fast transition from cosimulation to the cosyntheis of system prototype in our environment. Fig. 6 depicts the transition from cosimulation to cosynthesis.

VI. Integrated User Interface And Internal Representation

To provide an integrated user interface and internal representation, we extended Ptolemy. Ptolemy [15] is being developed at University of California, Berkeley as a block-diagram oriented environment for simulation and prototyping of heterogeneous systems. Instead of trying to capture all possible models of computations into one all-encompassing model, the Ptolemy kernel implements an object-oriented open architecture that enables any extensible model to be defined and added seemlessly. Thus, heterogeneous systems can be specified using different levels of abstraction and semantics for the various subcomponents.

For hardware-software codesign, we are developing Hetero Domain where heterogeneous models may coexist in the same representation. Fig. 7 shows an conceptual design flow under the extended Ptolemy environment. Initially, the user or system designer represents the abstract system model(or "universe" in the Ptolemy terminology) which consists of hierarchical blocks("galaxies") or atomic blocks("stars") with only data I/O ports. The internal representation of each atomic block may be either C or VHDL model, which does not imply implementation at this level of abstraction. According to the user's selection of partitioning option, it is then partitioned into CGC Domain (C Code Generation Domain) and VHDLF Domain (Functional VHDL Code Generation Domain) automatically or manually. If we perform manual partitioning, the initial

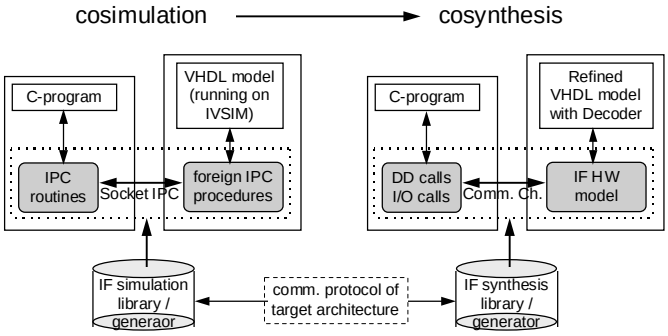


Fig. 6. Transition to cosynthesis from cosimulation

representation may imply the manually partitioned graph as shown in the second template of Fig. 7. The next action that the user has to do is just to "run"(or cosimulate) the system after selecting some architecture options to be explained below.

The Ptolemy partitions the universe into two separate model-specific universes after inserting the appropriate communication blocks(send and receive stars) at the boundary of these universes. The communication blocks are selected according to the user-specified architecture options such as the level of abstraction or the communication protocol. First, assume that we choose to cosimulate at the abstract level. In our example, two universes to generate C code(for software components) and VHDL code(for hardware components) are created respectively and the communication stars are selected to use the UNIX socket for cosimulation. Note that the send and receive stars do not imply that the communication protocol is a message-passing type. Instead, they do imply where communication between two different models arises. In Ptolemy, the kernel object to generate the code is called "target". While we use the default target for C code generation, we have developed the CosimTarget which generates a VHDL code for abstract level cosimulation. CosimTarget replaces the communication stars into a Socket Interface Star and adds protocol-related signal so that the modified model can be cosimulated. On the other hand, if we select the option for the detailed level cosimulation, not only communication stars but also appropriate communication models for emulating communication channel are inserted from the interface library.

In Fig. 7, only the first graph is visible to the user. Other graphs are internally generated by the Ptolemy, thus hidden from the user. We show an example of system specification using Ptolemy user interface in Fig. 8.

VII. Experimental Example

As an experimental example, a lossless data compression

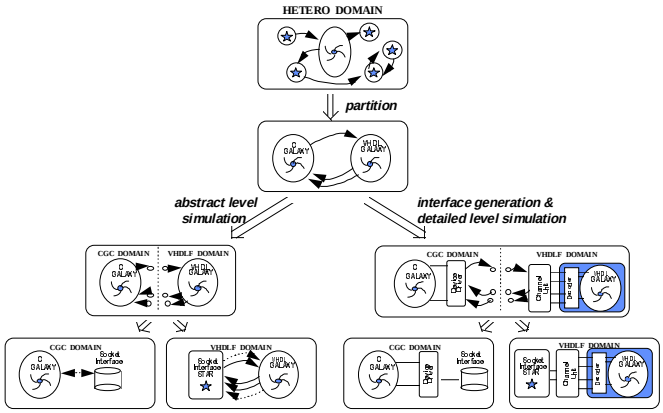


Fig. 7. Conceptual design flow under extended Ptolemy

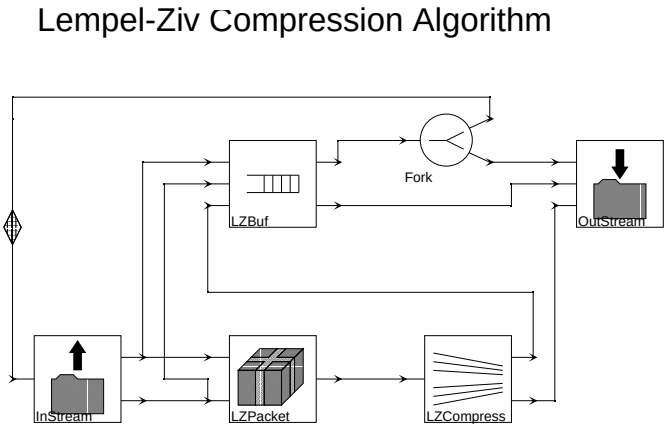


Fig. 8. An example of system specification in Ptolemy graphical environment

system was cosimulated and prototyped using our approach. Initially, the system contained only a C program implementing Lempel-Ziv lossless data compression algorithm(called LZ77 or LZ1 algorithm) [16]. Fig. 9 represents the skeleton of the program.

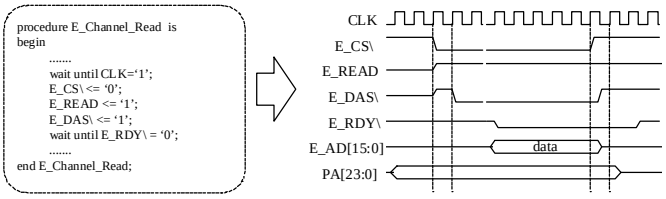
Then the system was manually partitioned into software and hardware components resulting in a mixture of a hardware component implementing parsing step and a software component implementing the remaining steps - initialization, coding, buffer updating, and file I/O. After inserting IPC routine calls in the components, we performed abstract level cosimulation. After the target architecture(Sparc + SBus + custom FPGA board) and communication protocol(SBus) were determined, hardware interface elements(described in Sec. III) were generated or selected from library and added to the hardware component for detailed level cosimulation. Fig. 10 represents the simulation models of IPC handler and channel unit(E-channel of SBus DMA controller). Due to the limit of space, description of the other simulation models such as decoder/signal register, hardware core, and top-level entity are omitted. After combining all simulation models, the detailed level cosimulation was done successfully and the result was the same as the result of the abstract level cosimulation.

```
lz77_compression( )
{
    /* fields of code word - maxlength, pointer, last symbol */
    int  maxlen, ptr;
    char  lsym;

    initialize( );                          /* initialization */
    for ( ; ; )      {
            shift_and_feed( );              /* file in & buffer update */
            parse(&maxlen, &ptr);           /* parsing */
            lsym = buf[bhalf + maxlen];
            put_code(ptr, maxlen, lsym);    /* coding & file out */
    }
}
```

Fig. 9. The skeleton of the C program of LZ77 algorithm

```
entity IPC_handler is
    port (maxlen : in bit_vector(3 downto 0);
          pointer : in bit_vector(3 downto 0);
          data : out bit_vector(6 downto 0);
          ..... );
    end IPC_handler;

architecture behave_IPC_handler of IPC_handler is
    procedure init_socket;      -- foreign procedure declaration
    procedure read_socket;      -- foreign procedure declaration
    procedure write_socket;     -- foreign procedure declaration
    procedure close_socket;     -- foreign procedure declaration
    signal  signal_id, input_data, output_data : integer;
begin
    init_socket;
    process
        .....
        read_socket(signal_id);
        read_socket(input_data);
        ....
        read_socket(signal_id);
        ....
        write_socket(output_data);
        ....
    end process;
end behave_IPC_handler;
```

(a)



(b)

Fig. 10. Simulation models of (a) IPC handler and (b) channel unit(E-channel read cycle protocol of SBus DMA controller).

Then the hardware component of the system was prototyped with an FPGA. The resultant hardware used 645 CLBs of Xilinx's 4010 FPGA [12]. With the FPGA clock of 6.25 Mhz, we obtained speedup of 1.7 over the implementation using only software component. In this experiment, some of the system prototyping tasks were performed manually because the environment had not yet been completely established.

VIII. Conclusion

In this paper, we present a hardware-software cosimulation environment for heterogeneous systems prototyping. To be an efficient system verification environment for the rapid prototyping of heterogeneous systems which consist of hardware and software componets, the environment supports special features: interface transparency, cosimulation acceleration, smooth transition to system prototype synthesis, and integrated user interface and internal representations. The resultant benefits of those features are the modularity of cosimulation components, no need of processor models, target architecture independence, and the conceptual simplicity and easiness in establishing and expanding the environment.

On-going and future works are as follows:

(i) Complete the implementation of the environment.

(ii) Extend interface model generator and library.

(iii) Generalize the environment to various target architectures including general purpose microprocessors or microcontrollers, DSPs, and ASICs. Currently, the system works only for Sparc + SBus + ASIC architecture.

(iv) Apply our approach to various system prototyping examples.

References

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

[2] D. Becker, R. K. Singh, and S. G. Tell, "An engineering environment for hardware/software co-simulation," *Proceedings of 29th ACM/IEEE Design Automation Conference*, pp. 129-134, June 1992.

[3] D. E. Thomas, J. K. Adams, and H. Schmit, "A model and methodology for hardware-software codesign," *IEEE Design and Test of Computers*, pp. 6-15, September 1993.

[4] R. K. Gupta, C. Coelho, and G. De Micheli, "Synthesis and simulation of digital systems containing interacting hardware and software components," *Proceedings of 29th ACM/IEEE Design Automation Conference*, pp. 129-134, June 1992.

[5] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Design and Test of Computers*, pp. 16-28, September 1993.

[6] W. R. Stevens, *UNIX Network Programming*, Prentice-Hall, 1991.

[7] Y. S. Lee and P. M. Maurer, "Two new techniques for compiled multi-delay logic simulation," *Proceedings of 29th ACM/IEEE Design Automation Conference*, pp. 420-423, June 1992.

[8] D. H. Ko and K. Choi, "IVDT: A VHDL developer's toolkit," *Korea Institute of Telematics and Electronics Journal of Electronics Engineering*, Vol. 5, no. 2, pp. 56-63, December 1994.

[9] The Institute of Electrical and Electronics Engineers, Inc., New York, New York. *IEEE Standard VHDL Language Reference Manual, ANSI/IEEE Std 1076-1993*, 1994.

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

[11] Y. Kim, Y. Shin, K. Kim, J. Won, and K. Choi, "Efficient prototyping system based on incremental design and module-by-module verification," *Proceedings of IEEE ISCAS 95*, pp. 924-927, May 1995.

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

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

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

[15] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems, "International Journal of Computer Simulation, Vol. 4, pp. 155-182, April 1994.

[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, Vol. IT-23, no. 3, pp. 337-343, 1977.