논리회로 설계 자동화를 위한 시뮬레이션 시스템*

A Simulation System for the Automation of Logic Circuit Design

한 창 호**
Chang Ho Han

Abstract

This paper describes an integrated environment for logic circuit simultion which is an important step of logic circuit design. The system consists of a logic simulator kernel, an expandible element routine library, a functional level element routine generator, several HDL input parsers, and a postprocessor. The system can simulate the same system in several levels of hierarchy. The experimental result shows that the system is very efficient and useful for design of logic circuits.

1. Introduction

Just about twenty years ago, commercial logic simulators like TEGAS began to be used [1]. Eventually almost all designers use logic simulators now. System level design can be easier if high level abstract simulation is performed in the early stage of the design [2]. Uncertainties can be minimized by performing simulation step by step. Some designers uses simulation technique for lower level desgin only. It is good for small size circuits [3]. Several different descriptions of one component or user defined components are allowed in some simulators, while others support only fixed, pre-defined components.

Many designers simulates circuits in several different levels of hierarchy [4]. Years ago, many logic simulators supported only structural domain descriptions and it means that circuits must be designed with the connection of other components. Logic simulators of todays support behavioal domain also. Occasionally, mixed domain design can be supported by some HDLs like VHDL [5,6,7,8]. Relatively small number of circuits need that kind of descriptions. Direct evaluation of the model is possible if the circuit is described fully in behavioral domain and indirect evaluation is needed if the circuit is described in structural or mixed domain.

Logic simulation is an important procedure in degin of digital systems including computers. When a new digital system is to be designed, there must be some

* 본 연구는 1993년도 인하대학교 연구비 지원에 의하여 수행된 것입니다.

** 인하대학교 전자계산공학과

way to verify the designed system, before it is actually implemented. Very simple circuits could be designed and verified using pencil and paper, and then test circuits can be built with IC parts, wires and solder. It is usually called prototyping. However, if the system is large, it is very expensive and time consuming to build the prototype. Hence, logic simulation systems are essential tools for digital cicuit designers. Using logic simulation systems, it is easy to describe, to verify, and to modify the design.

Circuit designers usually use hardware description languages (HDLs) [7,8,9] or graphics interface tools to provide the circuit design. However, usually systems have dedicated input methods for the particular systems, and the input methods are not compatible with each other. It is the reason why there are tremendous efforts to enforce VHDL (VHSIC hardware description language, VHSIC stands for very high speed integrated circuit.) as the standard HDL. However, VHDL is rather difficult to learn for hardware designers. It seems to be a mixture of all kinds of languages which are ever developed. Hence, people tend to use subsets of the language which are appropriate for the particular purpose.

While logic simulation is used to verify the design, fault simulation is used to evaluate test patterns [10]. When integrated circuit chips are fabricated, the chips need to be tested for manufacturing defects. Finding test input patterns which can cover most kinds of defects is not an easy problem. Hence, after sets of test patterns are found with some techniques, they must be evaluated. The technique to evaluate the test patterns with simulation is usually called fault simulation. Fault simulation is considered to be more complicated than logic simulation.

To perform simulation easily, the simulator must have a large set of library. The library contains many subroutines which act like electronic components. Usually the library is provided by the developer of the particular simulator or library vendors. Some commercial systems provide library development tools which are difficult to use. There have been a few published research results on the development of library [2,11,12].

Although there have been a lot of research efforts to improve the performance of logic simulation, the main focus was to speed up the simulation. Therefore, in many cases, users with several seperate good tools may not have good working environment.

This paper present a system which is an integrated logic simulation environment. The system is designed to make the design and simulation procedure easier by providing several important tools and convenient interface among the tools in an integrated environment. Next section present the overall structure of the system. Preprocessor, including HDL parsers, are explained in section 3. In section 4 the structure of the library and the library generator is explained. Section 5 shows the internal of the logic simulator kernel and the postprocessor. Finally the experimental result and the conclusion appear in section 6 and 7, respectively.

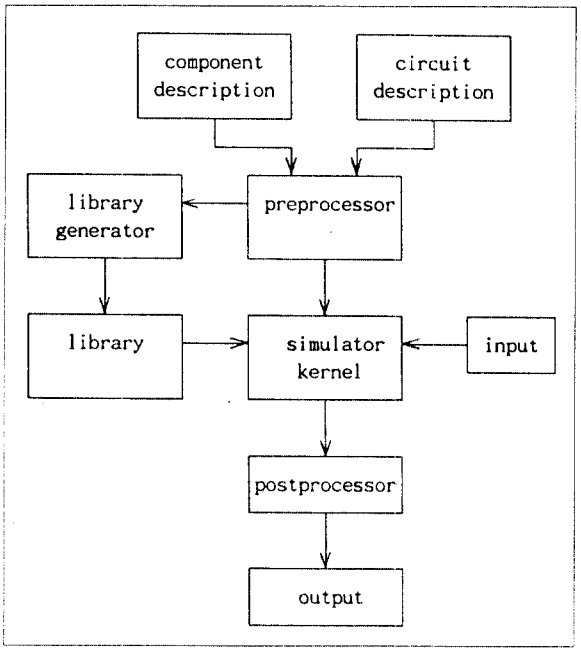
2. Overall Structure

The system consists of a logic simulator kernel, an expansible element routine library, a library generator, a preprocessor containing several HDL input parsers, and a postprocessor. 〈Figure 1〉 shows the structure of the system. As is shown in 〈Figure 1〉, the library generator is used to provide element routines which are eventually called by the simulator kernel when the circuits are being simulated. Each element routine in the library works like an electronic component like an AND gate, a flip-flop, or like an integrated circuit chips like 7440, 7475, etc.

The preprocessor consists of several parsers. It reads the circuit desctiption written in HDLs and generates internal representation of the circuits which can be used by the simulator kernel. Since the HDLs allow the circuit descriptions to be written in hierarchical

manner, the system can simulate a digital circuit in several levels of hierarchy. For example, a microcomputer system can be described with several high level blocks. Then abstract behavioral description of each block can be given for high level simulation. If the designer satisfies with the result, then detailed description of the blocks can be written.

The heart of the system is the simulator kernel. The simulator kernel performs simulation using the internal representation of the circuit description and the input patterns which should be applied to the circuit. Main function of the simulator kernel is to keep track of the internal states of the circuits, to calculate the output of each component of the circuit by calling appropriate subroutines from the library, and eventually to generate the output values for each set of input values.



〈**Figure 1**〉 **Overall structure of the system**

There are many possible methods to deliver the output values of the simulator kernel to the user. For example, the output voltages can just be displayed to the character terminal or printed on a piece of paper. More sophisticated graphical user interface could also be used to enhance user friendliness. The result may be compared to the desired output to check the design. Output can be reformatted to feed other design automation tools. In this research, to simplify the system and to concentrate more important blocks, postprocessor is currently implemented in relatively simple way. Usually commercial systems have excellent postprocessors including graphics display tools.

3. Preprocessor

The preprocessor mainly consists of several input parsers. The same preprocessor is used to parse circuit description for simulation or to parse component description for library generation. The preprocessor includes VHDL [1,2] parser, TDL (Tegas description language) [3] parser, and a table parser. In fact the parsers are not fully implemented. Full implementation of the parsers will be done in future research. The table parser is used only for library generation while other parsers are used for simulation also.

The table parser translates the component described in function table format to internal data structure. While it is relatively simple, it allocates a large amount of memory if the table size grows. Hence, the table description can be used for only relatively simple components.

The HDL parsers generates two differnt types of internal data structures defending upon the types of given circuit or component descriptions. If the description is in behavioral domain, the final target is to generate a C language subroutine representing the behavior of the circuit or the component. Hence, the parsed result is a tree containing arithmetic equations, Boolean equations, control flows, and others which can be converted to C language statements easily.

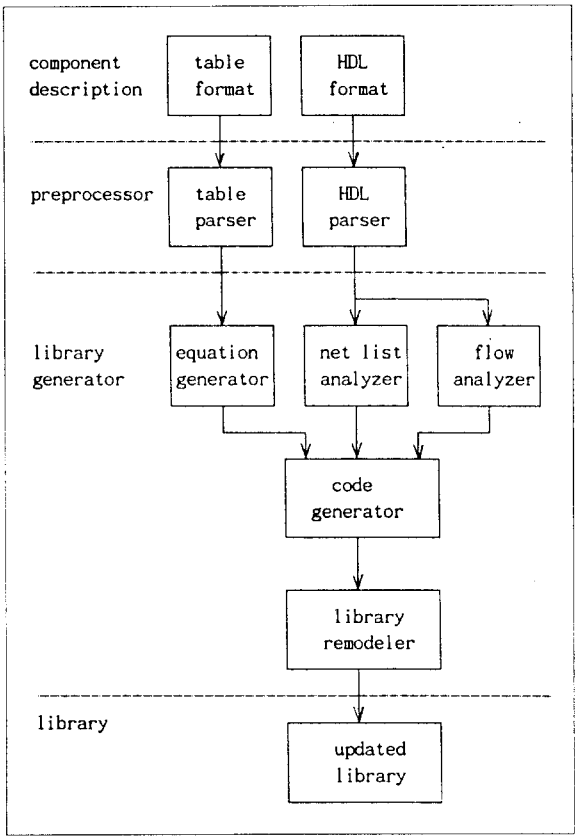
On the other hand, structural type descriptions contain lower level components. Actually, they are

language representations of circuit diagrams. A structural domain HDL description maps a logic circuit diagram. Usually, small circuits can be described with circuit diagrams easily. However it is not easy to describe a large system using circuit diagrams. Even though hierarchical diagram is used, it goes to deep levels to keep each diagram small enough to handle conveniently. Therefore, HDLs are preferred for large circuits. The parsed result of a structural description is a connectivity graph representing the net list of the circuit.

4. Library and Library Generator

The library contains the subroutines which are being called for component evaluation. Since the library should have a lot of subroutines for effective simulation, it is usually very difficult to build the library when a logic simulation system is developed. In this research, an expansible library is used which can be expanded easily using the library generator. The subroutine in the library are called element routines since they represent the electronic component which are to be evaluated. The element routines in the library reside as an archive file of object code. Each element routine can be coded and compiled manually. However, manual coding is difficult and time consuming. Therefore the library generator is developed to provide functional level element routines.

〈Figure 2〉 is a simplified diagram showing how the library generator works. When simulating a circuit, sometimes it is enough to describe the circuit in behavioral domain. In that case the cicuit description is compiled and simulated right a way. However, very often, cicuit is described as a network of lower level components. It is said to be a structural domain description. To simulate a circuit described in structural domain, subroutines for components in the circuit are needed. If all of the components used in the circuit are already defined, the description can be simulated



〈Figure 2〉 Block diagram showing library generation.

using the parser and the simulator kernel. However, if the element routine of any component is missing, it must be added to the library before simulation begins.

If one chooses to use the library generator rather than to code by himself, he can describe the behavior of the component in one of the two ways. The description can be given in a table format similar to the function table of a TTL circuit. The table is processed by the preprocessor to be parsed. The parsed result is passed to the equation generator. The equation generator generates a set of Boolean equations from the parsed table. The Boolean equations are sent to the code generator which generates a C language subroutine representing the characteristics of the component described by the function table. The code

generator uses signal model when generating program. The signal model is the way representing the electrical signal in the simulation process. Hence, if signal models are different, the generated programs are different even though the same description is used.

If the behavior of the component is described in an HDL, it is parsed first. If the component is described in hierarchical manner, ie., in structural domain, the output of the parser is a form of net list representing the connection of lower level components. The data structure of the net list is a directed graph. The directed graph representing the connectivity is then analyzed by the net list analyzer and sent to the code generator where it is converted into a C language subroutine.

If the component is described in the behavioral domain, the output of the parser is a tree representing mainly the control flow and sets of Boolean equations. The tree is passed to the flow analyzer to generate sets of control flows and Boolean equations which can be easily converted to high level language statements. The result is then sent to the code generator which generate a C language subroutine for the HDL description.

Once the C subroutine is generated, the subroutine should be added to the library. This can be done by the library remodeler. It is implemented using shell script. It updates the element list showing the list of component that can be used for simulation. It calls the C compiler to generate the object code and adds the object code to the library archive file. To perform simulation, subroutines in the library archive file are linked to the simulator object code.

5. Logic Simulator Kernel and Postprocessor

Generally, simulation is the process of deciding the expected output values for given input values to the model of the system. A logic simulator does the same. It generates the expected voltages of output terminals of a logic circuit for each set of voltages of the input terminals using the circuit model given in various ways. The main function of the simulator kernel is to schedule the components of the circuit and to run subroutines associated with the scheduled components to get the system response correctly.

To keep the cicuit connectivity, a directed graph is used. A node is used for each component of the circuit. An edge represent the connection between two components. The direction of an edge represent the direction of the electric signal. The record associated with each node carrys static data and dynamic data. Static data is given by the preprocessor when the circuit description is translated. Static data includes the name of the component, number of input terminals and output terminals, pointers to the connected components, a pointer to the subroutine for the component, delay of the component, etc.

While static data is not changed throughout simulation, dynamic data is updated when simulation proceeds. Dynamic data includes the current voltage of the signal connected to the output terminals of the component, an evaluation counter for detecting oscilation, state pointer for keeping internal state of sequential components, etc.

The simulator kernel is an event driven simulator. It means that the simulator evaluates only if an event occurs. Event is defined as change of signl values. When inputs are given for simulation, time stamp is associated to each given input. If a new input is different from the previous input value, an event occurs. The events are not transferred to the components directly. Instead, the events are scheduled for future evaluation. When the scheduled time reaches, the event is transferred to the specific component for evaluation. Evaluation is done by calling the appropriate element routine in the library. After evaluation, the output of each terminal is compared to the previous value of the signal. If any output value is changed, it

is scheduled for future evaluation. The scheduled time is the current time plus delay of the component.

To schedule the events, an improved time wheel method is used. The method uses a time wheel which contains several time slots. Simulation time is represented as integer values internally. The time difference between the two slots is the smallest time quantum used in the simulation process. When scheduling an event, scheduled time is divided by the number of slots in the time wheel. The remainder of the division is the time slot number where the scheduled event should be put it. Simulation time advances one slot when current evaluation is finished. After the simulation time is increased, any events in the time slot is processed. If there is no event, time is increased again.

The postprocessor reformats the output of the simulator for better understanding of the result. Output can be displayed on the CRT terminal or saved in a file. It also calculate some statistics which are useful to understand the characteristics of the circuit. The statistics include number of inputs, number of outputs, number of components, number of processed events per second, preprocessing time, simulation time, etc.

6. Experimental Results

The system is implemented on Sun Sparc 10 workstation using mainly C programming language. Parsers are written with LEX and YACC. 〈Table 1〉 shows the library generation results. Current implementation support table and TDL description for component library generation. For simulation, TDL and VHDL can be used. As is shown in 〈Table 1〉, it took about 0.033 seconds to generate the subroutine for a simple 6 gate component named C17 and about 147.7 seconds for a 3513 gate component named C7552. The descriptions were given in TDL. A 3 input 8 output decoder is described in a table form. It was translated to a C subroutine in 1 second.

〈Table 1〉 Library generation results.

name	C17	C432	C7552	decoder
description	TDL	TDL	TDL	table
size (gates)	6	160	3513	3in 8out
gen. time (s)	0.033	0.517	147.7	1.00

〈Table 2〉 shows the simulation results for several circuits with different sizes. C95 is simulated using TDL and VHDL. Simulation took similar amount of time for two different types of circuit descriptions. 〈Table 2〉 shows excelent results of executing several thousand events per second.

〈Table 2〉 Simulation results.

name	C17	C95	C95	C432
description	VHDL	TDL	VHDL	VHDL
size (gate)	6	27	27	160
input patterns	20	20	20	100
sim. time (s)	0.016	0.050	0.067	0.850
event/sec	10800	11979	8984	3315

The results in 〈Table 1 and 2〉 only show that the individual tools of the system are reasonably fast. Because the tools are implemented based on the current state of the art technology with some heuristics, the performance of each tool should be similar to those tools. The real value of this system is the tight integration of individual tools. Usually saving computer time with fast tools does not necessarily mean saving designer's time. For example, let us assume that you need to design a circuit board. If you use a new component in the design, it is usually very difficult to simulate the circuit because the element routine for the new component is not provided. In this case you may buy the element routine if you use particular commercial simulators. But usually the element routines can be provided several months later. Otherwise the element routines must be coded if you do not have an element routine generator, and usually it is the case. Even though you have an element routine generator

as a separate tool, it is not easy to integrate the new element routine to the simulator.

One of the advantages of this system is that it supports automatic library routine generation and automatic library remodeling. As a result, users can easily simulate circuits which contain new components. While the previous research results emphasise performance of the individual tools [11,12,13], this research mainly focuses on the integration of tools. The result is that users can save a lot of "human time" even if the "computer times" are similar to other individual tools.

7. Conclusion

In this research, an integrated logic circuit simulation environment was designed and implemented. The system is capable of generating simulation library and performing simulation. The component description can be given as a table or an HDL description. The circuits can be written in HDLs for simulation. The experimental results show that the system can perform logic circuit simulation very efficiently. The results also show that it is very easy to add a new component to the library if it is not already in the simulation library.

References

[1] S.A.Szygenda, "TEGAS2 - Anatomy of a General Purpose Test Generation and Simulation System," Proc. of Ninth Design Automation Conference, June 1972, pp. 117-127.

[2] M.Bloom, "Behavioral Models Take the Pain out of System Simulation," Computer Design, February 15, 1987, pp. 38-46.

[3] S.A.Szygenda, "The Evolution of Functional Simulation from Gate Level Simulation," Proc. of the IEEE Conference of System and Circuits, 1971.

[4] G.Zymmermann, "Top-Down Design of Digital Systems, Logic Design and Simulation," E. Hoerbst, Ed., Elsevier Science Publishers, 1986.

[5] R.Waxman, "The VHSIC Hardware Description Language - A Glimpse of the Feature," IEEE Design & Test, April 1986.

[6] R.Lipsett, E.Marschner and M.Shahdad, "VHDL - The Language," IEEE Design & Test, April 1986.

[7] J.H.Aylor, R.Waxmann and C.Scarratt, "VHDL Feature Description and Analysis," IEEE Design & Test, April 1986.

[8] VHDL Language Reference Manual, Intermetrics, Inc., 1987.

[9] TEGAS Language Reference Manual, TEGAS Systems Inc., 1973

[10] M.A.Breuer and A.D.Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, 1976.

[11] C.H.Han, S.Kang and S.A.Szygenda, "AFMG: Automatic Functional Model Generation System for Digital Simulation," Proc. of Int'l Asic Conference, 1991.

[12] C.Chuang, "The Automatic Element Routine Generator: An Automatic Programming Tool for Functional Simulator Design," The University of Texas at Austin, Master's Thesis, 1988.

[13] P.M.Maurer, "Two New Techniques for Unit-Delay Compiled Simulation," IEEE Tras. on Computer-Aided Design, Vol.11, No.9, September 1992.

● 저자소개 ●

한창호

1980 성균관대학교 전자공학과, 공학사
1982 서울대학교 전자공학과, 공학석사
1991 The University of Texas at Austin
Electrical Engineering, 공학박사
1983-1986 한국전자통신연구소 연구원
1991-1992 Cadence Design Systems, Inc., SMTS
1992-현재 인하대학교 조교수
관심분야: 논리회로 시뮬레이션