A Large Scale Parallel Processing Logic Simulation System

KENSUKE TAKASHIMA*, MASAO KATO*, SHINJI TAKAMURA*, KATUHIKO ARAI*,
AYATOMO KANNO** AND HIDEO IMADE**

1. *Introduction*

This paper describes a large scale parallel processing logic simulation system capable of handling 30,000 logic elements with the effective speed of 0.5 second per clock.

The system was developed and extensively used for the logic design and the maintenance design of a real time data processor employing 20,000 logic elements for use in an electronic switching system.

It is the utmost necessity for the processor to have accurate and efficient maintenance capability to keep it alive at any single fault, which required a complicated design both in hardware and programming.

It is desired to carry out various and voluminous program exercises and get exact information about what is happening in the whole processor logic by simulation so as to facilitate thorough logic debugging and proving fault isolating and diagnosing algorithm and programs.

The idea of gate level logic simulation,[1] which evaluates the logic function of each element, is not new. However, the disadvantages of the gate level logic simulators made so far by us[2] were a small number of elements to be accomodated and slow simulation speed that made whole processor simulation impractical. Due to these limitations, logic designer has to partition the whole processor logic into several blocks, which in turn gives him extra work and less accuracy of logic debugging.

Although recent efforts have been made toward register level simulators[3)4)5)6)] which provide the designer with register to register operation to describe his machine structure for simulation, the automatic translation of the machine structure described in the register level statements into the complete gate level specifications is still not an easy problem. Besides, the register level simulation which naturally does not handle the logic function of each element has difficulties in analyzing the behavior of the machine in various fault conditions.

From these reasons, a bigger, faster and convenient logic simulation system

This paper first appeared in Japanese in Joho Shori (the Journal of the Information Processing Society of Japan), Vol. 7, No. 5 (1966), pp. 263-272.

* Electrical Communication Laboratory NTT.
** Hitachi Ltd.

is desired. It is the purpose of the system to provide the designer a more immediate design help by which he can exercise ample test programs on the whole processor logic and get detailed logic information by ssimulation as though he is given an actual machine.

Two major ideas employed in the system to accomplish those objectives are the parallel processing logic simulation and the integrated use of the register register level and the gate level simulation.

The parallel processing technique employed in the system can carry out the gate level simulation for fifteen different test programs simultaneously which effectivcly speeds up the simulation.

Flexibility and generality is other important factors for the efficient use of the system. For ease of the gate level simulation, several kinds of statements are provided in the system to set input conditions, control a course of the simulation and display the results. The designer can freely define and change a simulation procedure with these statements to suit his need.

There is no reason, however, to exclude register level simulation so that it is another feature of the system to accept the register level simulation within the gate level simulation. The designer can easily define some other logic blocks than he has designed with the register level statements, or exclude the detailed gate level simulation of a part of the whole processor logic which he has already tested or has less concern, by replacing it with register level statements.

The following sections describe programming techniques to implement these ideas.

2. *Parallel Processing Logic Simulation*

The principle of the parallel processing logic simulation is simple.

Let S_{ij}^{k} be the state of element i for the j-th set of input variables at clock k and f_i be the logic function of element i, the state of the element i for j-th set at clock $k+1$ is represented by

$$S_{ij}^{k+1}=f_i(S_{1j}^{k},S_{2j}^{k},\ldots\ldots S_{nj}^{k}),$$

where n is the number of elements.

A 32 bit word is assigned to each element and fifteen bits each are used to store $(S_{i,1}^{k}\ldots\ldots S_{i,15}^{k})$ and $(S_{i,1}^{k+1}\ldots\ldots S_{i,15}^{k+1})$. The functions of all conbinatorial logic are compiled using bit parallel logic instructions so as to evaluate the functions simultaneously for the fifteen different sets of input variables.

The function of storage element or flipflop is interpreted in the following way bcforc compilation

$$F'=\overline{R}(F+S)=\overline{R}\cdot\overline{\overline{S}\cdot\overline{F}} \qquad (1)$$

where F and F' represent the previous and the next clock states of a flipflop, and R and S are reset and set gate conditions.

The gate condition such that

$$R \cdot S = 1 \qquad (2)$$

should be avoided in the design. Equation (1) and (2) are compiled as a subroutine and are used in common to all flipflops for the considerable saving of memory space.

The problems to use and control these simultaneous simulations are described in the following section.

3. *The Simulaiion Language*

Flexible means of the simulation are as important as the speed of simulation for the efficient use of the system. Table 1 shows typical simulation statements. Any one ($b=1\sim15$) or all ($b=0$) of the fifteen simultaneous simulations can be specified by the parameter b in the statements.

Table 1. Typical Simulation Statements.

Initial Procedure	
RDTB, b	Read Table
CLST, b	Clear States
Input Conditions	
SBTB, b, m	Set by table
SET, b, x	Set
RESET, b, x	Reset
Simulation	
SIML, p,q	Simulate
MOVE, b	Move
COMP, b,****	Compare
Simulation Control	
STCL, b, α	Set clock counter to α
RTCL, b, β	Raise clock counter by β
TEST, b, x,****	Test element x

b: case number, m: table number, x: element name,
p: clock phase, q: logic level,****: branch address.

Input conditions are specified by tables listing element names, states and clocks, which are read into the system by the statement RDTB. Setting of the specified element states according to the table is done by the SBTB.

The statement SIML calls for the compiled logic equations to be executed. The MOVE statement is used to replace the previous clock states ($S_{i,1}{}^{k}$.....$S_{i,15}{}^{k}$) by the next clock states ($S_{i,1}{}^{k+1}$.....$S_{i,15}{}^{k+1}$). Hazard condition can be detected by repeating the simulation with the same clocking condition and compare the results using the COMP statement.

The STCL and RTCL statements control clock counters provided one for each fifteen simulations, which in turn control required timing of other statements.

The TEST statement can be used to change the simulation procedure during the course of simulation.

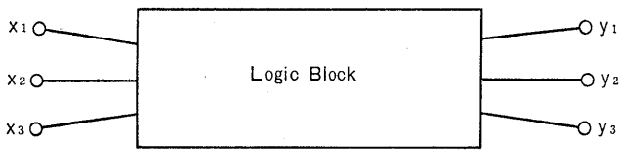
In order to integrate gate level and register level simulation, a statement

DEF, <register name><element names>

is provided to define a group of elements to be called by a common neme, Actually the group is not necessarily a register, but it can be any set of elements in a combinatorial network. A statement,

CONVERT, <register name><direction>

gives the way to transform the register contents into the element states or vice versa. The direction field defines direction of the transform. Figure 1 and Figure 2 show an example of integrated simulation procedure.



$r_1=x_1, x_2, x_3,$ $r_2=y_1, y_2, y_3$

DEF $r_1, x_1, x_2, x_3,$

DEF r_2, y_1, y_2, y_3

Fig. 1. Register Name Definition.

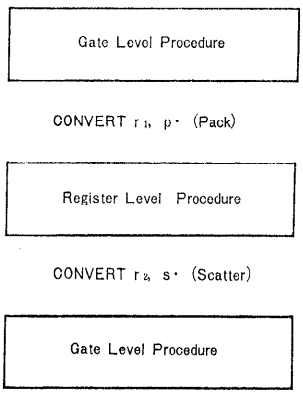


Fig. 2. Integrated Simulation Procedure.

Besides such usual register level statements as to add, count and transfer the contents of registers, another useful simulation statement is

TRANSPM, <adddress reg><data reg><timing>.

This statement transfers data between specified data register and pseudo memory provided in the system for the purpose of program execution by simulation.

A simplified example of the parallel processing simulation procedure is shown in Figure 3. In this example, fifteen test programs for logic debugging are first loaded into the pseudo memory and the simulation is started. Whenever termination condition is detected in any one of fifteen cases another test program is loaded so as to fill the fifteen cases all the time. The termination condition is detected by the state of the flipflop m indicating the execution of a halt instruction or some kind of instruction which is placed at the end and at every wrong way of the test programs. It is also possible to terminate the simulation whenever the time to execute an instruction exceeds some predetermined value to prevent unnecessary simulation from being continued.

As the system is often shared by several logic designers or programmers simultaneously, the actual parallel processing procedure is given the following capability to accept each users requests.

A. Setting of initial program address

B. Monitoring register and flipflop contents during the course of simulation

C. Specify "dump all states" period.

Here "dump all states" means recording all element states into magnetic tapes. The period is specified by clock times.

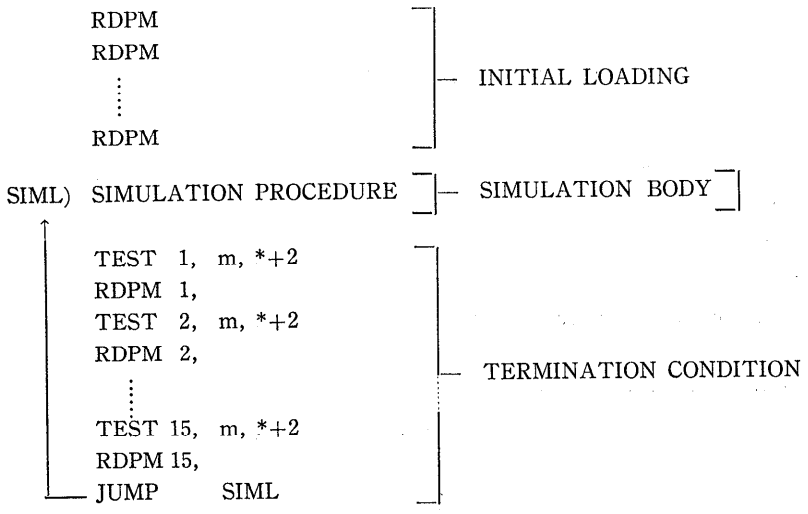


Fig. 3. Parallel Processing Logic Simulation Procedure.

4. *The Simulation System*

In order to handle a number of logic elements as many as 30,000, it is natural and appropriate to automate the handling. Actually the system was made

following four jobs.

Job 1. Data arrangement
Job 2. Translation
Job 3. Simulation
Job 4. Data conversion.

The data flow in the system is shown in Figure 4.

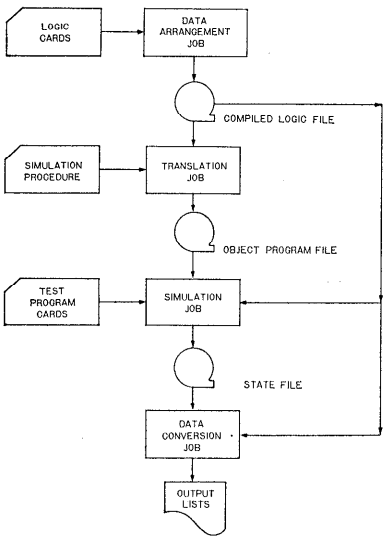


Fig. 4. The System Data Flow.

The data arrangement job takes care of logic data maintenance, checking, sorting and compilation. The second and the third functions trace down logic trees and check out excessive fan-ins and fan-outs and rearrange the data in the sequence of their logic levels. Here the logic level of a gate is defined to be the largest number of gates required for a signal to travel from a flipflop to reach the gate.

Undefined input terminals and the elements with excessive levels and/or with feedback connection are checked out in this job before simulation. The compiled logic file, the output of this job, includes compiled logic equations as well as an address—element name conversion list for later translation and data conversion jobs.

Simulation procedure is first translated into the assembler language and the control is transferred to commercial assembler and loader routines. And then the simulation is started.

During the course of simulation the contents of specified registers and/or the states of all elements are dumped into magnetic tapes for later display at the data

conversion job.

Three kinds of display form are:

A. Register form

B. Time chart form

C. Element name form.

The element name form which shows the element names in lexicographical order only when they changed their states, provides the most condensed and convenient information for the final logic error shooting.

5. *Application*

This system was introduced in the logic design and the maintenance design of the real time data processor. The design was fairly complex due to several instruction look ahead control together with micro level control lockout function for fault locating purposes.

The whole processor logic was tested under nearly 10^4 steps of test programs to complete the logic debugging by simulation. Programmed checking, as shown in Figure 5, was successfully used in the final debugging period to speed up the voluminous testing.

The system has been used in the following three areas of the maintenance design.

A. Simulation of duplicated processors

B. Fault isolation program testing

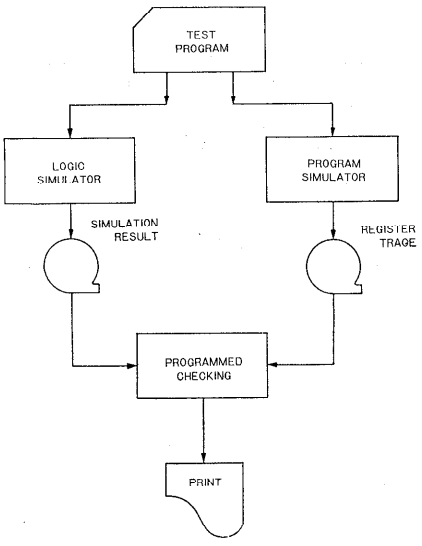


Fig. 5. Programmed Logic Error Checking.

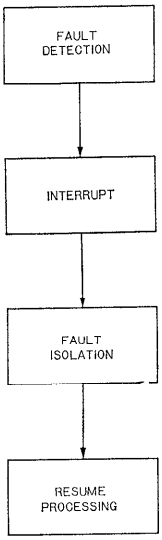


Fig. 6. Fault Isolation

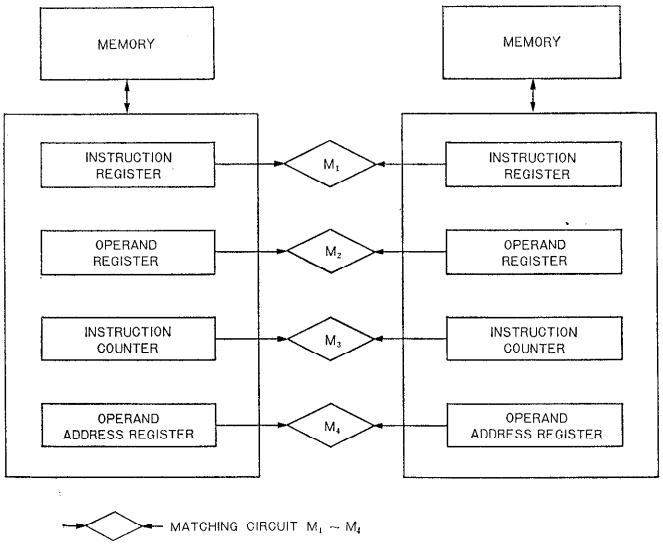


Fig. 7. Fault Detecting Circuits in the Duplicated Processors.

C. Diagnostic program testing

From the purpose of the real time processor, the course of fault isolation, shown in Figure 6, must be quick and accurate, which required the circuits shown in Figure 7 for immediate fault detection.

Duplicated processors together with these fault detecting circuits were conveniently

accommodated in the system using the parallel processing technique by which both the processors can be simulated simultaneously.

Communications between the processors were done by CONVERT statements in the simulation, while various fault conditions were injected using the element state tables and the statement SBTB.

6. *Conclusion*

The system has been extensively used in the development of the duplicated processor for an electronic switching system.

The design was tested for nearly 10^4 steps of test programs to carry out logic debugging in which five per cent logic elements are renewed through 20 major logic file corrections. Further use of the system, for duplicated processor simulation on fault condition and for the fault isolating atd fault diagnosing program testing is now being done.

With this system it is possible to give various and voluminous program exercises and get exact information about what is happening in the whole processor logic. The process of logic data processing, from the logic data arrangement, through the logic simulation to the final data conversion, is automated and no human intervension is required. This made it possible to return the results to the designer within an hour after he issued his logic changes, for most of the test programs. Moreover fifteen exercises or fifteen people can join the debcgging or testing simultaneously.

It is faster, more accurate and convenient than to handle the actual machine. The ideas employed in the system have proven to be successful for the efficiency and flexibility of the system.

References

[1] Stockwell, G.N., Computer Logic Testing by Simulation. *IRE Trans. on Military Electronics, MIL-6*, (1962) 275-282,

[2] Takashima, K., and H. Tsuda, Logic Simulation Programs. *The Journal of the Information Processing Society of Japan, 4*, 2,(1963).

[3] Gorman, G. F., and J. P. Anderson, A Logic Design Translator. *Proc. FJCC* (1962) 251-261.

[4] Proctor, R. M., A Logic Design Experiment Demonstrating Relationships of Language to Systems and Logic Design. *IEEE Trans Trans. on Electronic Computers, EC-13*, August, (1964) 422-430.

[5] Zuckfr, M. S., Locs, An EDP Machine Logic and Control Simulator. *IEEE International Convention at New York City*, March 23 (1965).

[6] Griffin, J. F., and M. J. Haims, An Experiment with the Simulation of Machine Logic and Control. *IEEE International Convention at New Bork City*, March 23 (1965).