LOGIC SIMULATION SYSTEM USING SIMULATION PROCESSOR (SP)

Minoru Saitoh*, Kenji Iwata*, Akiko Nakamura*, Makoto Kakegawa*,
Junichi Masuda*, Hirofumi Hamamura*, Fumiyasu Hirose**, Nobuaki Kawato**

* Fujitsu Limited ** Fujitsu Laboratories Limited
Nakaharaku Kawasaki, 211, Japan

ABSTRACT

This paper describes a (special-purpose) logic simulation processor (SP) and a software system for the SP for use in verifying the design of computers and other logic devices. Our system can evaluate a logic circuit containing 4 million logic primitives and 32M bytes of memory at a maximum speed of 800 million active primitive evaluations per second.

This paper outlines the hardware architecture, then discusses a software system that optimizes hardware performance. It presents the results of system use and evaluates the system.

1. INTRODUCTION

As computers grow in scale and complexity, rapid advances in VLSI are enabling circuits to be integrated more densely. These advances, however, have led to a problem of how to implement engineering changes after VLSI component manufacture without lengthening the development time. The only practical answer is to minimize design errors before actual manufacturing starts.

This requires a logic simulator that can handle entire large logic circuits quickly. To do this, we developed a simulation processor (SP), which simulates logic circuits at high speed, together with logic simulation software that maximizes hardware features.

We started by setting up specifications that would enable system-level logic simulation of millions of gates simultaneously. We wanted to develop a "total" logic simulation system providing as many of the functions required by logic designers as possible, using optimum-performance hardware and software.

SP hardware implements up to 64 gate processors (GPs) in a high-speed, parallel-processing pipeline architecture. Its hierarchical network enables high-speed communication of events between GPs.

We minimized host-SP communication and improved overall system performance by implementing the following into hardware: (1) a dedicated function to hold external input signals for driving the simulated circuit; (2) monitoring of up to eight points in the simulation model and the stopping of simulation when a set condition is met; and (3) a dedicated function to hold simulation results.

The performance of a simulation system does not depend only on the hardware operation speed. When the simulation speed is extremely high, it is important to minimize software overhead and raise overall performance. In developing software, we implemented four functions to maximize hardware features and improve overall simulation:

(1) Mixed-level simulation

Mixed-level simulation is achieved by automatically generating a gate-level circuit from a function-level circuit described in digital system description language (DDL) [1], which describes hardware at a register transfer level,and combines the generated gate-level circuit with a separately defined gate-level circuit.

(2) Minimized overhead in host-SP communication

The simulated circuit model is combined with a logic circuit model synthesized with control conditions specified in simulation control language (SCL). Overhead is minimized by monitoring communication conditions using the SP hardware monitoring feature during simulation.

(3) Optimized circuit partitioning

The work load is distributed evenly to GPs to raise the parallel processing efficiency.

(4) High-speed simulation model modification

A circuit model is modified by interactively changing the simulation model on the SP for all functions without recompiling and reloading.

2. HARDWARE ARCHITECTURE

Hardware configuration

Figure 1 gives the SP configuration, and Table 1 lists SP specifications. The SP consists of gate processors (GPs) for gate and memory operations, event transmission (ET) for communicating events between GPs, an input processor (IP) for supplying external input signals to the simulation model, an output

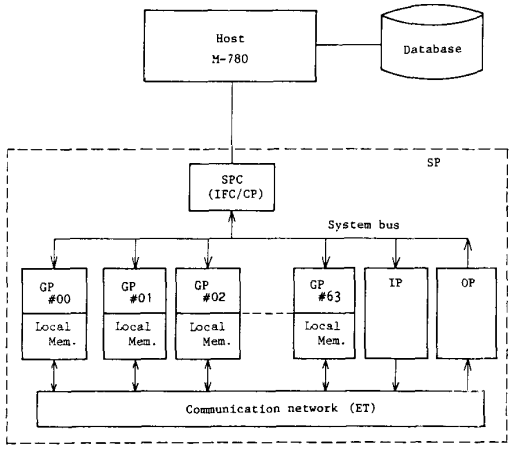

Figure 1 SP hardware configuration

processor (OP) for holding values of signals from the simulation model and monitoring signals from eight points in the simulation model, and a control processor (CP) for controlling SP processors. The host that generates simulation models and controls the SP is connected to the SP by an interface controller (IFC).

Events are communicated from the IP to GPs, between GPs, or from GPs to the OP at one unit time by the ET in parallel with GP gate operations. Overhead in event communication is masked by GP gate operations.

Processor functions are outlined below.

Gate processor (GP)

The GP simulates gates and memory primitives (SP primitives). Each GP can process 64K gates and 512K bytes of memory.

Figure 2 gives models of gate and memory primitives simulated by a GP. The operation of the 4-input, 1-output random logic gate is defined by a truth table; 1024 different gates can be defined. Each gate is evaluated based on unit delays; signals can assume four values--0, 1, X, and Z. A memory primitive is represented by a model having 16 address buffers (treated as a gate) and one memory cell. Data memory is 1 bit wide; each memory primitive has an address capacity of 64 to 64K that is represented as 2^n. Memory contents are binary--0 or 1--and stored in 4M bits of RAM.

Input processor (IP)

The IP supplies external input signals, including a simulation clock, to the simulated circuit.

Periodic events, such as clocks, and events that vary randomly are defined independently and paired with time data indicating signal changes. Periodic events are then stored in Periodic event memory and random events in random event memory. Periodic event memory retains up to 2K items of time and event data, and is used cyclically.

Table 1 SP hardware specifications

Speed	800 million eps (peak)
Capacity	Logic: 64k primitives/processor Memory: 0.5M byte/processor
Simulation accuracy	Delay: unit delay values: Maximum of 16
Algorithm	Event-driven
Host interface	Byte multiplexer channel

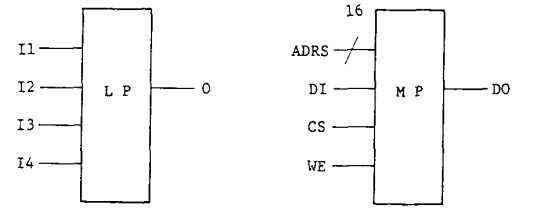

Figure 2 Logic and memory primitive models

Random event memory retains up to 256K items of time and event data.

Output processor (OP)

The OP has two functions.

First, it holds up to 4M signal value changes (events) about nets designated as trace points. Scrolling enables a full buffer to be overwritten.

Second, it continuously monitors a specific net, checks whether the net output matches the monitoring value, and notifies the CP to stop the SP when a match is made.

Scrolling and monitoring are combined so that event data occurring at a trace point need not be transferred between the host and the SP. The OP can also stop the SP. Data can be read from the host each time the buffer becomes full.

Event transmission (ET)

In communication between processors, the IP, OP, and GPs transmit 32-bit events via ET. ET is a network of nodes called ET units. Each event has a destination processor number, which ET units check to determine the transmission route. Each ET unit has buffer memory and can receive events from GPs at any timing.

Control processor (CP)

The CP supplies a system clock to, and synchronizes, processors. When a condition for stopping the SP is met during gate monitoring, the CP sends a signal to stop related processors,

and sends an interrupt the host to report that the SP has stopped.

Interface controller (IFC)

The IFC controls data communication between the host and the SP. SP registers and memory are connected to the host by a system bus that carries data transferred from the host to the SP. The IFC controls the system bus to write data from the host to SP registers and memory and to read data from SP registers and memory for sending to the host.

Host computer

The host, usually an Fujitsu M-series model such as the M-780, generates simulation models and performs other software tasks, such as SP control.

A detailed description of the SP hardware is presented in [2].

3. SOFTWARE SYSTEM

Figure 3 gives the software system concept. The software system we developed is outlined as follows:

Outline

Figure 4 gives the software configuration.

In addition to the SP simulator, the system includes software simulator run on a general-purpose computer. Except for simulation execution programs, including those for simulation model generation and analysis of simulation results, programs can be used for both the software simulator and the SP simulator. The user can switch as needed from one to the other. In LSI-level simulation, more than one designer usually performs different simulations at the same time, and interactive quick turnaround time tasks are required to some extent. The software simulator is most efficient in LSI-level simulation. The SP is powerful in simultaneous large-scale, system-level simulation, and is most efficient in simulating a system that combines LSIs.

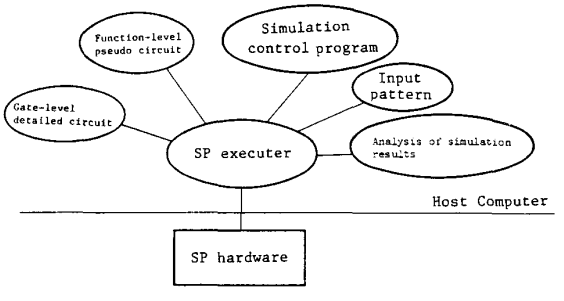

Figure 3 Software system concept

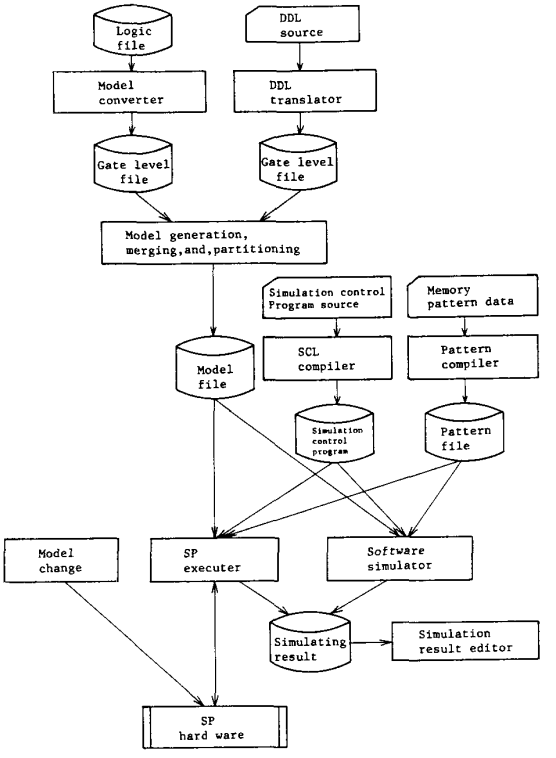

Figure 4 Software system configuration

Software functions are as follows:

Logic circuit model conversion program

The logic circuit model conversion program retrieves circuit information needed for simulation from the logic circuit database, which is segmented into blocks, each corresponding to an LSI or hierarchical level. It then generates a gate-level file for retaining information on gates and memory primitives on which the SP performs operations. The gate-level file also retains information on connection between primitives. Gate-level files are stored as a library; when part of a circuit (circuit block) is changed, only the gate-level file for the circuit block needs to be updated.

DDL translator

The DDL translator synthesizes a gate-level simulation model based on DDL-coded functions. The results of synthesis are output to the gate-level file. The model builder combines the description in DDL and the gate-level description from the logic circuit diagram; this combination is then loaded into the SP and simulated at high speed. This makes it possible to describe, in DDL, the pseudo circuit to drive and control the designed circuit to be simulated and to load the

```
<SYSTEM> EXAMPLE:
  <UNIT> U1:
    <EXTERNAL> CLOCKIN.
    <TRIGGER>  CLK=CLOCKIN.
    <TERMINAL> A(3),B(3),C(4).
    <REGISTER> D(3).
    <AUTOMATON> A1:CLK:
      <LOGIC>
        |* C:=B'1010' *|
          D <- A + B;
          D <- B'000'.
    <END> A1.
  <END> U1.
<END> EXAMPLE.
```

Figure 5a An example of DDL description

SP with these two circuits for high-speed simulation. This is much faster than simulation with the pseudo circuit on the host computer. Figure 5 gives a simple example of a DDL description and the synthesized gate-level circuit.

The DDL translator synthesizes logic in two stages:

- First, it generates a register-transfer-level simulation model from the DDL-coded source.
- Second, it expands components, e.g., registers, memory, and adders, for the model into gate-level primitives by using an algorithmic method.

The DDL translator must meet two requirements:

First, the number of gate stages between registers must be minimized. Too many gate stages in DDL makes the number of levels per clock pulse large in the overall simulation process, lowering efficiency.

Second, the number of gates in the gate-level circuit must be minimized. The SP has only a limited capacity for a simulation model, and DDL code involved in SP processing must be minimized to increase logic design capacity. Schemes adopted to solve these problems are discussed below.

Adders of up to 16 bits are represented by models using a carry look-ahead (CLA); adders having more bits are represented by combinations of CLA adders using a ripple-carry.

The numbers of gates and gate stages are reduced using the SP feature that any logic primitive that can be represented with a truth table can be defined if it has four inputs and one output.

See the circuit example in Figure 5b. The AND/OR gate used as a data selector is treated as an SP primitive. The flip-flop model is represented by using two SP primitives connected by a feedback loop. An EOR gate with up to four inputs can be defined by one SP primitive; the actual logic circuit consists of several gates. The EOR gate is used to represent the sum of the adders, reducing the number of gates.

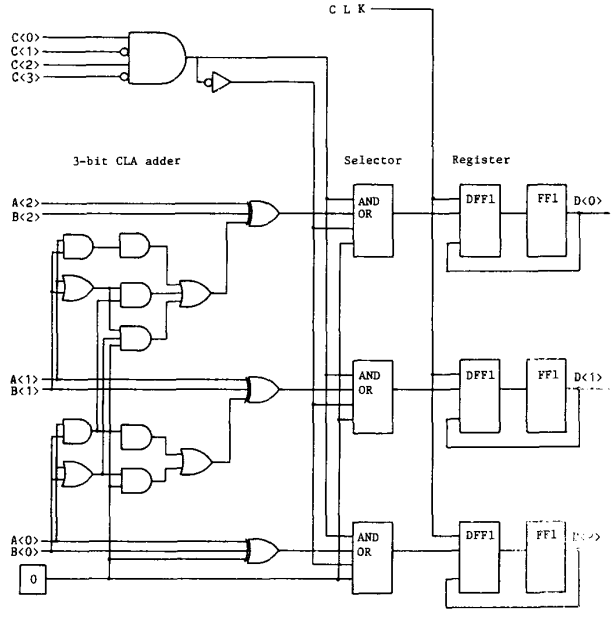

Figure 5b Gate-level circuit generated from the DDL example of Figure 5a

Model builder

The model builder generates a model file by combining gate-level files, each of which has been generated for an LSI (or a hierarchical level) or generated through translation from DDL. It also divides the circuit as instructed manually and assigns circuit segments to GPs. Two requirements must be met for circuit partitioning:

First, interprocessor communication must be reduced. Second, the work load must be evenly distributed among processors.

The first requirement is met by assuming that gate-level files consist of function units, then dividing the function units into several groups and assigning them to individual processors.

The second requirement is hard to meet because the circuit locations where many events occur vary with the test data. Currently, the circuit is partitioned as directed manually by the designer.

SCL compiler

The simulation control language (SCL) compiler receives source code written in SCL and generates SCL object code in the form of statements for controlling simulation.

SCL enables a condition for stopping circuit simulation to be specified using Boolean expressions. The SCL compiler synthesizes the gate model from termination condition to combine it with the simulated circuit model. This combination is then loaded into the SP and

```
SIMLA:  PROC;
                /* clock definition */
CLOCK #CLKA   T(50) PHASE(25) WIDTH(3)
              POLARITY(+);

BOOLEAN;        /* monitor condition setup */
  MONITOR #M1(/UNIT-A/REG1<0:1>);
  MONITOR #M2(/UNIT-B/REG2<0:1>);
  MONITOR #E1(/UNIT-A/ER-REGA);
  MONITOR #E2(/UNIT-B/ER-REGB);
  MONITOR #STP1(1),#ERR(1),#STOP(1);
  LOGIC   #STP1=#M1==#M2;
  LOGIC   #ERR=#E1|#E2;
  LOGIC   #STOP=#STP1|#ERR;
END;

ON GATE(#STOP); /* processing performed when monitor condition
                   is satisfied */
  CALL PRINT;   /* subroutine call */
  STOP;
END;

SIML 10000;     /* 10,000 clock simulation */

STOP;
```

Figure 6 SCL example of monitor conditions

executed. The gate status value corresponding to the output of the Boolean expression is monitored by the OP monitoring feature during simulation; when the condition is met, simulation stops. This enables designers to specify detailed conditions for monitoring the circuit and to perform simulation efficiently without having to always read signal values from the SP and check whether the condition is met on the host. Figure 6 gives an SCL example of monitoring conditions. Figure 7 gives a gate model synthesized from Boolean expressions.

Pattern compiler

The pattern compiler generates data to be loaded into memory in the simulated circuit. It compiles microprograms and test programs coded in Assembler to drive the simulated circuit.

SP simulation executer

The SP simulation executer loads the simulation model into the SP and controls simulation based on the SCL-coded procedure. It loads additional monitoring gates into the SP, specifies conditions for stopping the SP, reads trace memory contents of simulation results when the SP stops, and outputs a simulation results file.

Software overhead for executer processing includes the following:

(1) Time taken in loading the circuit model into the SP

(2) Time taken in loading memory data to the SP

(3) Host-SP communication overhead involved in SP control

(4) Time reading simulation results memory

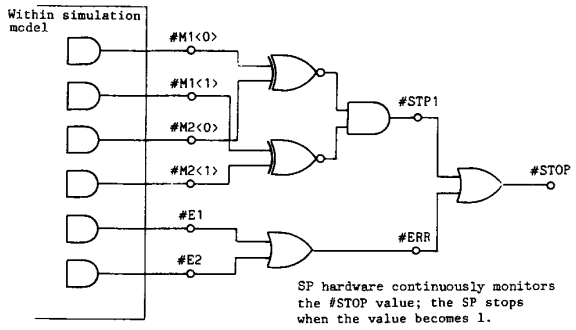

Figure 7 Circuit synthesized from Boolean expressions in SCL

The overhead in (1) and (2) cannot be avoided. Once circuit data is loaded, it can be directly modified using the logic modification program described later. Loading is then not needed for second and subsequent simulations, if any. Overhead in (3) can be minimized by SCL coding so that the SP is accessed less frequently. Overhead in (4) is minimized by designating only required locations as trace points and using scrolling.

Interactive simulation model modification program

The interactive simulation model modification program has two functions:

① Modifying connections of the simulation model processed in the SP

② Modifying connections of the simulation model in the model file to generate a new model file

Connection errors in the circuit found as a result of simulation can be modified by the designer as follows: The simulation model processed in the SP is temporarily modified using function ①, then simulation performed; this can be repeated as needed. The history of modifications is stored in the command file. A model file reflecting the included changes is generated by receiving the command file and calling ②.

Part of a simulation model can be modified by (1) restarting from logic entry, (2) modifying the model file, or (3) directly modifying the simulation model loaded into the SP. The time needed for processing varies greatly, especially for models consisting of several million gates. Their ratio is 1000:100:1 for (1), (2), and (3).

Simulation results editor

The simulation results editor reads simulation results file contents and displays simulation results in the screen format requested by the designer. Signal nets, time, and other factors can be specified interactively.

Table 2 Simulation results

Number of processors	Relative performance*	Load distribution ratio
12	166	0.495
24	252	0.410
32	361	0.439
48	408	0.344
64	500	0.310

* Performance of software simulator on M380 =1
Model size logic : 468k primitives
memory: 211k bytes
Model generation time 250 sec (M780 CPU time)

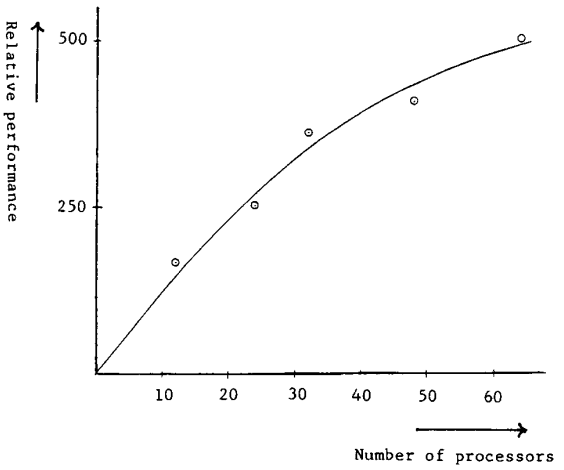

Figure 8 Relative performance

Software simulator

The software simulator has an internal table format, processing, and other characteristics similar to SP hardware. The simulator shares input and output files with the SP simulator. The results of simulation by the software simulator and the SP agree for each time unit.

The software simulator has two advantages:

(1) When a relatively small circuit is to be simulated, the software simulator is more efficient because it enables multiprocessing by users and interactive, close simulation.

(2) SP hardware can be tested by comparing the results of SP simulation with the software simulator's results.

4. RESULTS

Table 2 gives the results of performance measured by SP simulation. The load distribution ratio in Table 2 indicates the equalization level of the load distribution to individual processors and how many processors are operating efficiently. Here, only 20 of the 64 processors are operating efficiently.

Figure 8 shows how the performance increase rate falls when the number of processors is increased.Performance varies greatly with circuit characteristics and partitioning. This test data is not for optimum circuit partitioning.

Performance is improved by optimizing circuit partitioning.

5. CONCLUSION

We have developed a logic simulation processor (SP) and a related simulation system.

SP hardware operates at 800 million active primitive evaluations per second to evaluate a logic circuit containing up to 4 million logic primitives and 32M bytes of memory. The following software architecture was developed to maximize the above SP hardware performance:

(1) Logic synthesis from a function description language (DDL)

(2) SP control to minimize overhead from host-SP communication

(3) Circuit partitioning tool

(4) Circuit modification model tool

Logic simulation is vital to VLSI component development, and our simulation system using the SP is the key to shortening computer development time.

References

[1] Duley, J. R. and Dietmeyer, D. L. D., "A Digital System Design Language (DDL)," IEEE Transactions and Computers, vol. c-17, No. 9, pp. 850-861, September 1968.

[2] Hirose, F., et al "Simulation Processor SP", Proceedings of IEEE International Conference on Computer Aided Design (ICCAD), pp.484-487, Nov. 1987.

[3] Pfister, Gregory F., "The Yorktown Simulation Engine: Introduction," Proceedings of the 19th Design Automation Conference, pp. 51-54, June 1982.

[4] Sasaki,T.,et al "HAL:A Block Level Hardware Logic Simulator" ,Proceedings of 20th Design Automation Conference, pp. 150-156, June 1983.