Algorithm for Vectorizing Logic Simulation and Evaluation of "VELVET" Performance

Yoshiharu Kazama* Yoshiaki Kinoshita*
Motonobu Nagafuji** Hiroshi Murayama*

* Kanagawa Works, Hitachi Ltd., Hadano, Kanagawa 259-13, Japan
** Hitachi Computer Engineering Co., Ltd., Hadano, Kanagawa 259-13, Japan

Abstract

A very large-scale logic simulation engine "VELVET" has been developed.

VELVET is a vectorized event-driven simulator which can handle simultaneously both gate-level logic and Register-Transfer Level structure.

VELVET can process simulation jobs two orders of magnitude faster than a conventional gate-level simulator.

This paper describes how to realize such high performance, an algorithm for vectorizing the simulation and performance.

1. Introduction

VELVET has been developed with the following objectives :

(1) Speed up logic simulation* by two orders of magnitude over the existing gate level simulator.
(* : including preprocessing and postprocessing)

(2) Be compatible with the existing software simulator in terms of simulation test data, output format and other aspects

As with software simulators, the choice of simulation approch is crucial in achieving the performance goal with the simulation engine. Generally, two approaches to computer simulation are well known : the compilation approach and the event-driven approach. The former simulates the behavior of all the logic gates and flipflops(latches) at every clock cycle. Whereas it fits vectorization because of the simplicity and uniformity of control, it has a speed disadvantage because of the huge amount of calculation required.

On the other hand, the event-driven approach identifies, and simulates the behavior of only those logic gates and flipflops (latches) which can change their signal levels at each given clock cycle. It therefore requires more sophisticated execution control, while realizing higher performance, and is not readily vectorizable. When vectorized, it may further improve simulation performance by several times, which is not great enough to achieve the development objectives. The addition of a special hardware feature to facilitate the vectorization of simulation logic would provide potentials for a dramatic performance increase.

The event-driven approach would be preferable also from the compatibility point of view, as it is already employed by the current software simulator.

This paper describes how the development objectives of VELVET have been met from the software perspective.

2. Speed-up Methods

2.1 Addition of Special Instructions for Simulation

While it was decided to use Hitachi S-810 as the base hardware for the VELVET engine for performance reasons, the supercomputer is not necessarily fit for logic simulation as it is. First, the supercomputer in general is not designed to handle bit string data or variable length data efficiently, which is the basic data format in logic simulation. Second, it is not well equipped with such scalar features as would efficiently handle the complex data structures and the sophisticated control which often appears in an event-driven simulator.

In view of fully exploiting the performance potentials of the S-810 supercomputer, a special hardware feature has been devised to remove these shortcomings [1].

Chapter 3 discusses the special hardware feature which consist of six new instructions.

2.2 Utilization of Extended Storage

As the execution of simulation kernel is speeded up, the input/output time associated with magnetic disks tends to constitute major performance bottleneck in relative terms as observed by the user.

Fig. 2.1 shows the impact of input/output processing time on the total performance for a conventional general-purpose computer and for the S-810 supercomputer, taking an example of a gate-level simulator. In this example, the input/output time constitutes only ten percent of the total elapsed time on a general-purpose computer using magnetic disks as workareas to store the simulation results. As the CPU processing time diminishes through the use of the S-810 supercomputer, the proportion of i/o processing time increases drastically. The S-810 is equipped with the Extended Storage(ES) made of high speed semi-conductor storage which replaces disks as a workstorage to store the simulation results.

2.3 Clock Event Suppression

Reducing the number of events which are to be simulated or which trigger simulation is the key to improving the performance of a simulator. Hitachi developed an algorithm for suppressing unnecessary clock events in logic simulation [2]. VELVET employs the same clock event suppression algorithm to maximize its performance potential [3].

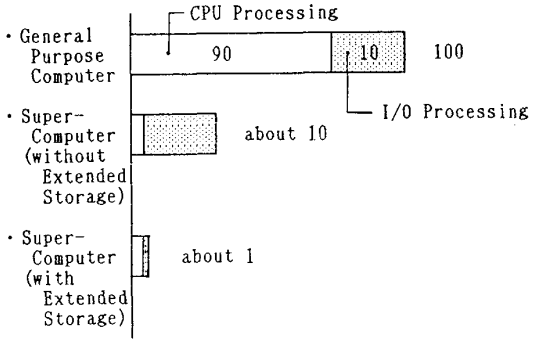


Fig. 2.1 Impact of I/O Processing on Total Performance

3. New Instructions for Simulation

3.1 Overview

Table 3.1 lists the six new vector instructions added to the supercomputer S-810 to speed up simulation processing. VSF and VCI are the vector instructions which directly executes the logical operation of the gate under simulation in one machine cycle. VITRL is useful for vectorizing variable-length data, and VBMG facilitates the handling of bit-string data. VLS and VLXS load data from storage into vector registers at high speeds.

Of these, the first four significantly contributed to the performance enhancement as they improve the vectorization ratio and the number of effective elements in one vector.

Table 3.1 Special-Purpose Instructions

No.	Instructions	Steps*
1	VSF: Vector Simulate Function	1437
2	VCI: Vector Compare Indirect	129
3	VITRL: Vector first order ITeRation Logical	51
4	VBMG: Vector Bitwise MerGe	52
5	VLS: Vector Load Separate	44
6	VLXS: Vector Load indeXing Separate	112

*:Number of steps of Instruction Interpreter developed for VELVET Software Debugging

3.2 Debug

The development of the VELVET software proceeded in parallel with the development of the additional hardware feature, i.e., the implementing of the six new instructions into hardware. To avoid serialization of the two developments, the authors developed, as a software debugging tool, an interpreter for each new instruction using conventional instructions.

This set of interpreters helped shorten the development period of the VELVET system considerably. The total number of program steps for these interpreters was about 2K(two thousands). Table 3.1 also shows its breakdown by instruction.

4. Outline of Simulation Method

4.1 Description of Logic Circuit under Simulation

Fig. 4.1 shows an example of the description of the logic circuit under simulation. This example, the logic circuit to be simulated consists of two parts: the processor which contains the ALU and the storage control, and the main-storage unit.

The entire processing unit is to be simulated at the instruction level. VELVET stores the test program designed for test and maintenance into the main-storage unit and then fetches and executes the test program to simulate both the processor and the main-storage unit. This macroscopic approach is effective to verify such a combination of relatively large-scale logic circuits [4], [5].

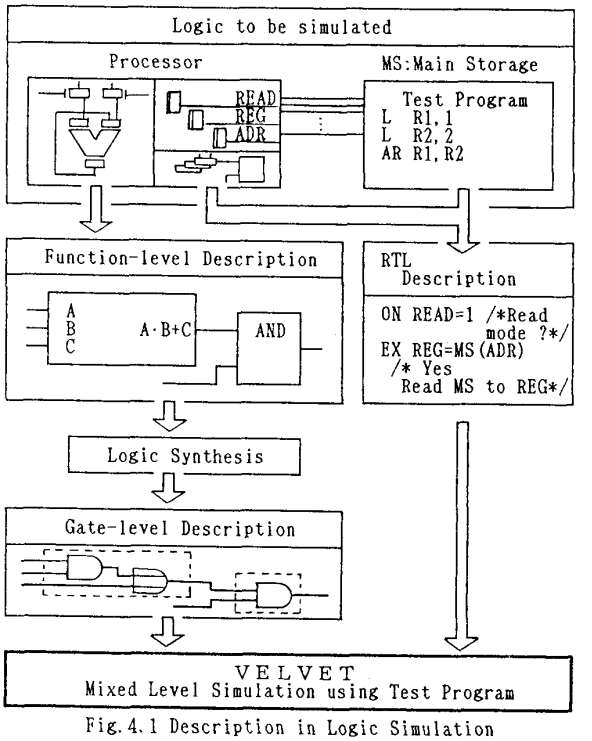


Fig. 4.1 Description in Logic Simulation

The processor part is first described in ALDL(A Logic Description Language), and then converted to a gate-level description by the logic synthesis program [6], [7].

On the other hand, the main-storage part is described in ON/EX, a kind of RTL(Register-Transfer Level) language, plus an ordinaly assembler language [8]. The internal storages in the processor, such as the control storage, are also described in this RTL language.

VELVET software performs "mixed level simulation" : it simulates the behavior of the logic circuit which is described in both RTL and gate-level language, using the test program as its input data.

As discussed in chapters 5 and 6, VELVET simulation time has been shortened at both the gate-level and RTL level.

4.2 Flow of Simulation Processing

Fig. 4.2 shows the general flowchart of mixed-level simulation involving gate-level and RTL descriptions. This simulation employs the " zero delay and clock synchronization" method : it repeats gate-level processing and RTL processing alternately for each clock cycle until there is no change in any signal. Logical operations at the gate level and at the register level are simulated, and if, as a result, there is a change in the output value of any gate or register, the simulator creates a new event.

5. Gate-level Processing

5.1 Data Structure for Gate-level Simulation

Fig. 5.1 shows an example of logic gates to be simulated. The VELVET logic compiler generates a table, shown in Fig. 5.2, to represent this combination of logic gates. This table can be considered a vector.

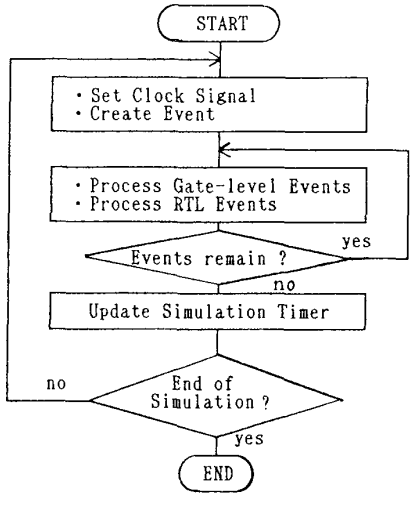


Fig. 4.2 Mixed Level Simulation Flow for Gate-level and RTL

In Fig. 5.2 FN means the function performed by the gate, such as AND, OR, and LATCH. The FO(fan-out) of each of gates G1 and G2 points to the record representing gate G3 ; the FI1(fan-in 1) and FI2(fan-in 2) of gate G3 point to gates G1 and G2, respectively.

5.2 Gate-level Processing

Fig. 5.3 shows the general process flow of gate-level simulation, corresponding to Figs. 5.1 and 5.2. Now, let us assume that the output level of gate G2 changes from '0' to '1'. As a result of this signal change, VELVET creates an event for gate G3 which is pointed to by FO1 of gate G2. This event is picked up and processed in the same clock cycle as that of the change in G2.

First, VELVET picks up Vs(Signal Values) of gates G1 and G2 pointed to by FI1 and FI2 of gate G3, and then places them in W1 (work register 1) and W2(work register 2), respectively.

Next, VELVET executes the logical operation as indicated by the FN part of gate G3. VELVET updates V of gate G3 with the result of this operation, and, if there is a change in V of gate G3, creates a new event to track the gates pointed to by the FO(s) of gate G3.

VELVET repeats these steps until no new event is created. In these operations, VELVET uses the six newly added vector instructions as well as the standard vector instructions of the S-810 supercomputer.

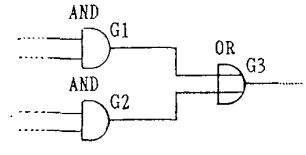
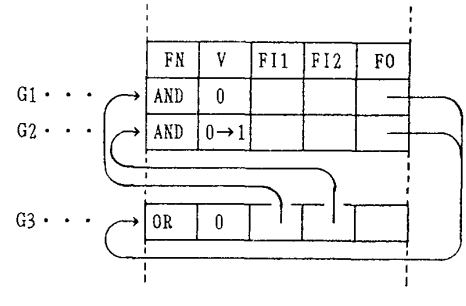


Fig. 5.1 Example of Simulation Model



FN:FuNction V:signal Value
FI1, FI2:Fan-In 1, Fan-In 2 FO:Fan-Out

Fig. 5.2 Data Structure for Gate-level Description

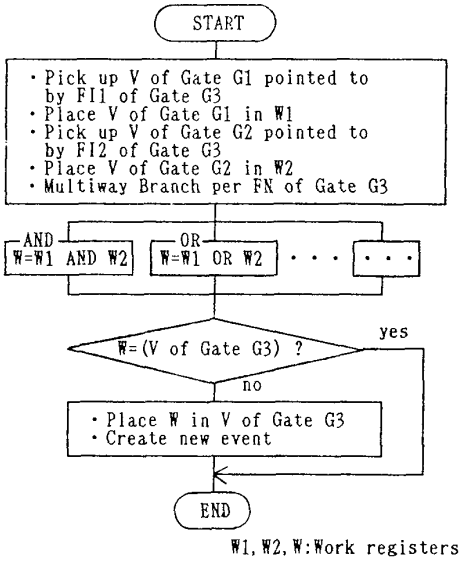


Fig. 5.3 Processing Flow of Gate-level Simulation

6. RTL Processing

6.1 Data Structure for RTL Simulation

An example of RTL description is given in Fig. 6.1. This example shows a data fetch operation from main storage (MS). Fig. 6.2 illustrates the data structures created by VELVET for the RTL descrption and for the gate level description corresponding to Fig. 6.1. Register variables such as T0, MODE, REG and ADR are placed in the gate-level structure such as t0, mode(0), mode(1), reg(0), reg(1), adr(0) and adr(1). In this gate-level structure, one signal (bit) of the actual logic occupies one word. Therefore, a conversion of signal value data format is required between the gate-level structure and the RTL data structure.

6.2 RTL Processing

Fig. 6.3 shows the general process of RTL simulation, corresponding to Figs. 6.1 and 6.2. In the "Update Registers in RTL" step, the contents of T0 or MODE are updated with the data taken from the gate-level structure. If there is a change in any of these values, VELVET checks on the "ON" condition : "T0=1 and (&) MODE=2" in this example. If the condition is true, VELVET executes the "EX" statement : "REG=MS(ADR)". Finally, the new contents of "REG" are converted and returned to the gate-level structure.

The majority of these procedures are vectorized using three newly-added vector instructions : VITRL for data format conversion, and VCI and VSF for the checking of the "ON" condition.

As a result, VELVET simulates an RTL description 40 times faster than the conventional gate-level simulator (in the case of the example shown in Fig. 6.1).

```
ON  T0=1 & MODE=2
EX  REG=MS(ADR)
```

Fig. 6.1 Example of RTL Description

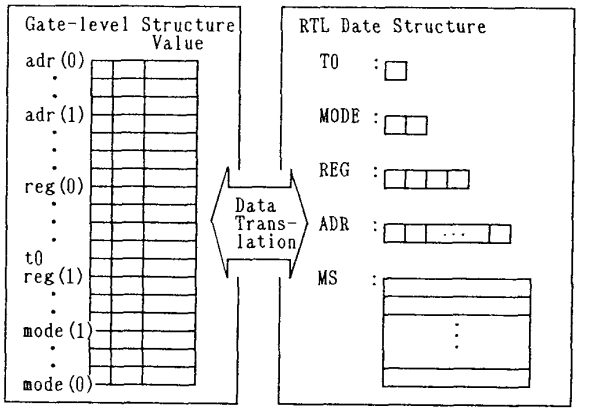


Fig. 6.2 Date Structure for Gate-level and RTL Description

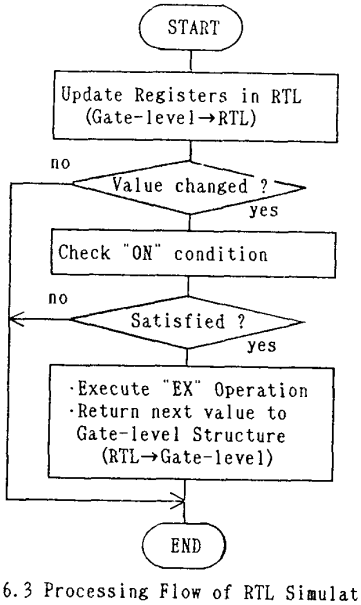


Fig. 6.3 Processing Flow of RTL Simulation

7. Evaluation of VELVET Performance

Fig. 7.1 illustrates the performance of VELVET versus the size of the logic simulated. The "VELVET processing time" is defined as the length of time between the moment SMF and SDF are input to the simulation and the moment the outputing of SRF completes (see Fig. 7.3). The VELVET processing time increases very slowly as size increases. This near-independence from the logic size owes much to the vectorization of simulation processing. Compared with software simulation in which simulation of one elementary logic operation takes a number of instructions, VELVET executes it in just one hardware machine cycle.

As a result, the increment in processing time per unit increase in the number of gates is much smaller than that in software simulation.

Thus, VELVET is especially advantageous in simulating large scale logic.

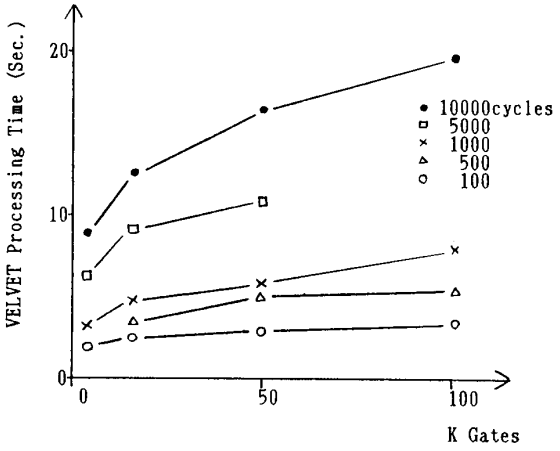


Fig. 7.1 Performance of VELVET (1)

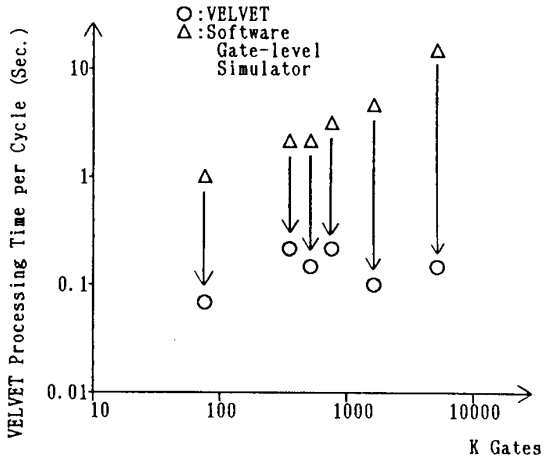


Fig. 7.2 Performance of VELVET (2)

Fig. 7.2 compares the performance of VELVET with that of a conventional gate-level software simulator. As demonstrated, VELVET has achieved more than 100 times higher performance.

The performance gain is not uniform due to the following factors :

(1) logic size
(2) number of cycles simulated
(3) number of effective events
(4) event density (This factor depends on the structure of the logic.)
(5) time split between gate-level processing and RTL processing

8. Result of using VELVET in the development of S-820

Fig. 8.1 shows the performance of VELVET in entire machine simulation for Hitachi's latest supercomputer S-820. These simulation use the test programs for design check and field maintenance as simulation test data. Whereas conventional software simulators can not efficiently simulate the behavior of the logic over such a large time span as running a test program, VELVET does it neatly and high speeds.

9. Conclusion

Hitachi's logic simulation engine VELVET has been developed, based on the S-810 supercomputer with six ad hoc vector instructions added. Simulation speed has been enhanced by two orders of magnitude over a conventional gate-level software simulator. For user-friendliness, VELVET retains compatibility with the conventional software simulator with respect to simulation data, result listings and other aspects; for completeness, it performs simulation at various levels. At the system level, it uses, as its input, test programs designed for actual design check and field maintenance.

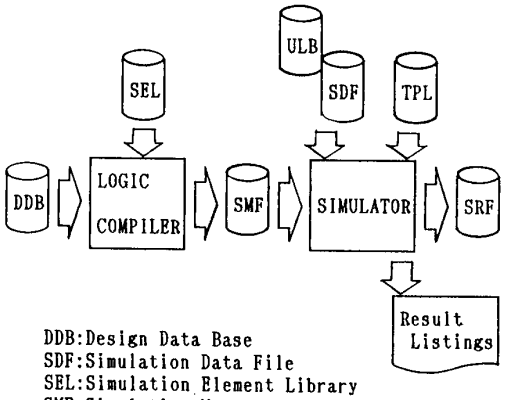


Fig. 7.3 VELVET System Configuration

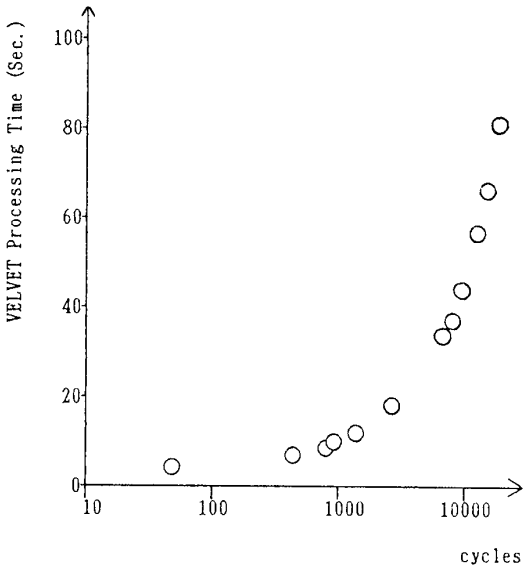


Fig. 8.1 Performance of VELVET in entire machine simulation for Hitachi's supercomputer S-820

10. Acknowledgment

The authors would like to thank Hisashi Horikoshi, Tokinori Kozawa, Kenichi Furumaya, Yasuhiro Ohno, and Toshihiko Odaka for giving them this challenging opportunity of developing VELVET, and Shigeo Nagashima, Shunsuke Miyamoto, Yoji Tsuchiya and Shun Kawabe for providing them with necessary support.

11. References

[1] Nagashima S. et al., "Hardware Implemetation of VELVET on the HITACHI S-810 Supercomputer", ICCAD, 1986, pp. 390-393.

[2] E. G. Ulrich, "A Design Verification Methodology based on Concurrent simulation and Clok Suppression", 20th DA Conf., 1983, pp. 709-712.

[3] Takamine Y. et al., "Clock Event suppression Algorithm and its Application to S-820 Development", 25th DA Conf., 1988

[4] Miyoshi M. et al., "An Extensive Logic Simulation Method of Very Large Scale Computer Design", 23rd DA Conf., 1986, pp. 360-365.

[5] Ohno Y. et al., "Design Verification of Large Scale LSI Computers", ICCC, 1982, pp. 443-446.

[6] Ohno Y. et al., "Princeples of Design Automation System for Very Large Scale Computer Design", 23rd DA Conf., 1986, pp. 354-359.

[7] Tsuchiya Y. et al., "Establishment of Higher Level Logic Design for Very Large Scale Computer", 23rd DA Conf., 1986, pp. 366-371.

[8] Ohno Y. et al., "Logic Verification System for Very Large Computers using LSI's", 16th DA Conf., 1979, pp. 367-374.