VLSI DESIGN
1996, Vol. 4, No. 2, pp. 119–133
Reprints available directly from the publisher
Photocopying permitted by license only

A Hardware Accelerator for Fault Simulation Utilizing a Reconfigurable Array Architecture

SUNGHO KANG[a,*], YOUNGMIN HUR[b,†] and STEPHEN A. SZYGENDA[b,‡]

[a]*Yonsei University, Seoul, Korea;* [b]*Dept. of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712.*

In order to reduce cost and to achieve high speed a new hardware accelerator for fault simulation has been designed. The architecture of the new accelerator is based on a reconfigurable mesh type processing element (PE) array. Circuit elements at the same topological level are simulated concurrently, as in a pipelined process. A new parallel simulation algorithm expands all of the gates to two input gates in order to limit the number of faults to two at each gate, so that the faults can be distributed uniformly throughout the PE array. The PE array reconfiguration operation provides a simulation speed advantage by maximizing the use of each PE cell.

This new approach provides for a high performance, cost effective, gain over software simulation. Simulation results show that the hardware accelerator is orders of magnitude faster than the software simulation program.

Keywords: Fault simulation, Hardware accelerator, Parallel computer architecture.

INTRODUCTION

Simulation is the process of exercising a realistic model of a digital system with sets of input stimuli. Fault simulation plays an important role in testing and diagnosis by determining the fault coverage of a test set. However, the computation time for this process is growing rapidly due to increases in the size and complexity of the systems to be simulated.

In order to reduce simulation time, a number of techniques have been introduced in recent years. One of these techniques is hardware simulation. It is known that hardware simulation can be more than 1000 times faster than software simulation [1]. These hardware approaches include two speedup factors, namely; architectural concurrency and algorithmic concurrency. Efficient architectural concurrency can be achieved by allowing parallel processing elements to effectively deal with the circuit. Algorithmic concurrency is characterized by a simulation algorithm which can take advantage of the architectural concurrency. The major disadvantage of the hardware simulation approach is that it is expensive and inflexible as compared to software simulation approaches.

Several simulation accelerators have been devel-

*Corresponding author. Present address: Electrical Engineering Dept. Yonsei University, Seoul, Korea. Phone: + 82-2-361-2775, Fax: + 82-2-312-7735. E-mail: shkang@bubble.yonsei.ac.kr

†Email: yhur@cerc.utexas.edu

‡Email: szygenda@ece.utexas.edu

oped, including; the TEGAS Accelerator [2], the Yorktown Simulation Engine [3], the Logic Evaluator [4] Realfast [5], LSM [6], MARS [7], etc. These were primarily designed for logic simulation purposes, since the fault simulation process is far more complex and costly. Recently, it was reported by Fehr [8] that a dramatic speed-up potential is achievable in logic simulation by using a parallel hardware configuration which directly maps the design simulation topology onto the accelerator. This architecture focuses on large scale concurrency through; hardware parallelism, high speed input/output, optimized interconnect (fanout) circuits, and pipelining of the design execution cycle. This architecture was mainly intended for functional verification of gate-level, unit-delay, binary-valued, logic designs.

There have been several other attempts to develop a hardware accelerator for fault simulation. Levendel [9], introduced a special purpose architecture for simulation, using a parallel fault simulation algorithm. Also, there is an approach based on a concurrent fault simulation algorithm [10], which consists of parallel processing units for each task of concurrent fault simulation. Agrawal [11], presented a pipelined multiprocessor system using a concurrent fault simulation algorithm. Later, the accelerator was upgraded by adopting a new pipelined algorithm for message passing multicomputers [12]. The major drawback with existing approaches to fault simulation has been their cost.

The object of this paper is to describe a new architecture for fault simulation. Since the main disadvantage of the hardware approach is the cost, a major concern was to reduce this cost. The architecture introduced in this paper is an array type architecture, which is very cost effective as compared to other architectures. Also to achieve high performance, a new fault simulation algorithm was developed for the new architecture. This algorithm takes advantage of the new architecture through the use of parallelism, where possible.

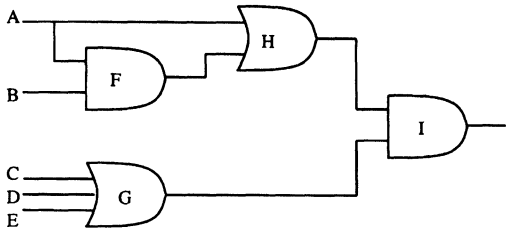
ARCHITECTURE

The underlying architecture of earlier hardware accelerators is that of a multiprocessor system, where the processor can be configured in a distributed fashion for architectural concurrency, or in a pipelined fashion for algorithmic concurrency. These architectures have the advantage of high simulation speed compared to software simulation. However, this higher cost limited their usage to special applications.

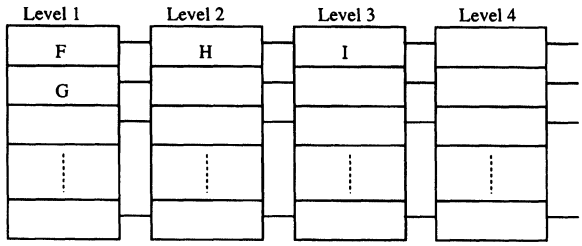
Generally, the required capabilities of an accelerator to perform efficient fault simulation are as follows. Firstly, boolean function evaluation and processing must be fast for all functional types. Secondly, the platform needs a large capability to contain the entire design partition during a simulation cycle. Thirdly, the communication structures for signals such as; fanins, fanouts, and feedback loops among the individual processors must be simple and fast. Lastly, fault insertion and propagation must be executed efficiently and the required memory space must be as small as possible.

In this approach, fault simulation is executed on a massively parallel processor array. This massively parallel hardware accelerator is composed of a tightly connected array of simple boolean evaluation processing elements, where the accelerator provides high speed fault simulation at a reasonable cost. In this architecture, we apply a direct mapping strategy as described by Fehr [8] for logic simulation, which maps the fault simulation topology onto the accelerator PE array. The basic mapping concept of the new architecture is a one-netlist-node-per-array-element representation of the circuit and it uses pass-gate logic for array interconnections. Therefore, the netlist topology can be mapped as an overlay onto the PE array, and all gates which are assigned to the same level in a netlist can be evaluated concurrently. An example of the mapping of a circuit onto the hardware accelerator is as follows. Figure 1 (a) shows the original circuit to be simulated. This circuit is levelized in the netlist and mapped onto the PE array (Figure 1 (b)) in the hardware accelerator. However, the new algorithm must execute an expansion of the number of gates and adjustment of the number of fan-ins. Figures 2 (a) and (b) show the expanded circuit of Figure 1 (a) and its mapping onto the PE array, respectively.

If the number of gates at one level in a circuit and the number of PEs in one column of the PE array are



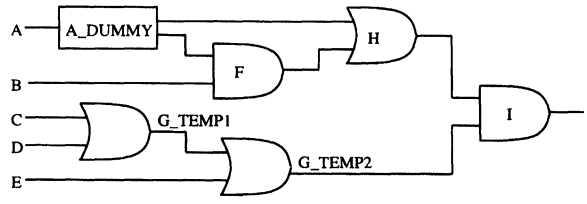
(a) Example Circuit



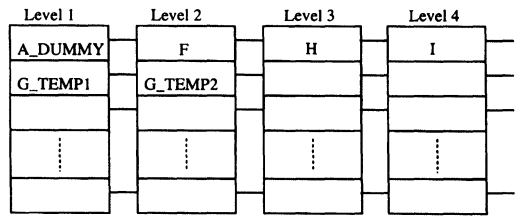
(b) Mapping onto PE Array

FIGURE 1 Example Circuit

the same, then the one-to-one mapping between them is straightforward. However, it is more often the case that the number of gates in a circuit to be simulated is larger than the number of PEs in one column of the PE array. In this case, the netlist has to be partitioned into groups using a partitioning algorithm which reduces the communication between groups. On the other hand, if the number of gates is much smaller than the number of PEs in one column, a PE array reconfiguration operation which is explained later, is executed.



(a) Expanded Example Circuit



(b) Mapping onto PE Array

FIGURE 2 Expanded Circuit of Figure 1

The new accelerator system consists of a Simulation Pattern Memory, a Netlist Memory, a Control Unit, a Fault List Memory, and a Processing Element (PE) array with local memory and signal interface. Figure 3 shows a block diagram of the accelerator. The parallel execution capability of the accelerator is provided by an $M \times N$ array of PEs, which communicate by signal interface circuits. The signals at the edges of the PEs wrap around to the edges of the other side and this forms a doubly connected mesh type of structure. The PE array is managed by the Control Unit, which is synchronized by an external clock. Every PE in a column is executed concurrently.

The description of each component of the accelerator is as follows. The Simulation Pattern Memory receives the simulation patterns for each circuit to be simulated from the host computer, and provides the necessary input pattern to the first column of the PE array whenever simulation (a good simulation or a fault simulation) is requested. The Netlist Memory contains information which is required for fault simulation, including: a gate function for boolean evaluation, input/output switch data for fanin/fanout information, and space for storing the simulation results. The preprocessor has a switch configuration algorithm which configures the connections between the gates and stores them in the Netlist Memory. The

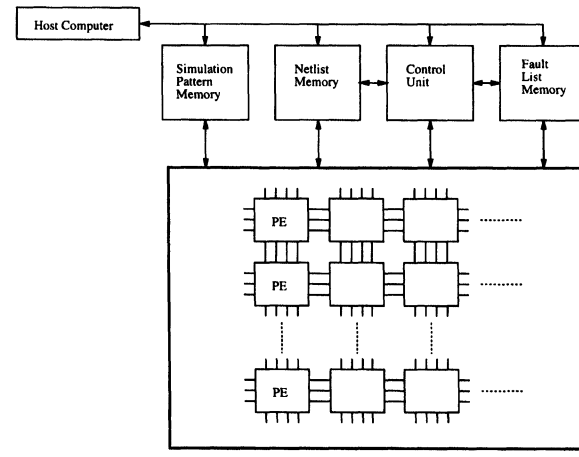


FIGURE 3 Configuration of the Accelerator

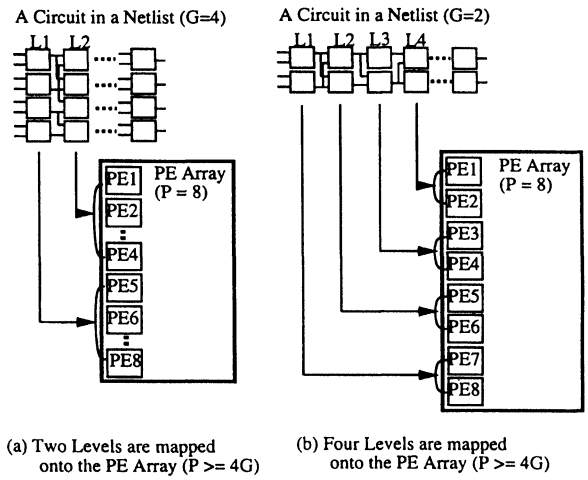
fanin/fanout relationship is represented in terms of the node number and a signal direction (fanin or fanout). The Netlist Memory keeps the PE array Address Translation Tables, which are controlled by the Control Unit, for determining the PE location. The Fault List Memory holds information about detected faults with a fault identification (ID) and value, and it receives fault information which is generated at every fault simulation cycle. This information is used for the generation of test sets and fault coverage. Fault dropping is also executed by using data from the Fault List Memory.

The Control Unit supplies the necessary data retrieved from the Netlist Memory for each PE array element. For fault simulation, each level of the circuit in the netlist is down-loaded into each column of the PE array and all PE elements in the same column are executed concurrently under control of the Control Unit. In addition to the initial loading procedure, the Control Unit checks the simulation states for each column and manages the PE array allocation.

The Control Unit monitors the Fault Flags of each PE array element and finds the PE columns where no faults exist, because they have already been detected during a previous simulation cycle. If a PE column which has no faults is encountered and all faults in all lower levels than the present level are detected, then the next fault simulation skips those levels to reduce the simulation time.

Initially, the fault simulation methodology performs a good circuit simulation. When the good circuit simulation is completed, then each PE cell contains the value of the correct output for each signal. This information is used for comparison during the fault simulation cycle. In the faulty circuit simulation cycle, the hardware accelerator performs a PE array reconfiguration in order to improve the simulation time. It divides each column of the PE array into several groups and maps the multiple levels of the netlist onto these groups, so that each group can be executed simultaneously. The PE array reconfiguration is performed according to the circuit topology in the netlist. This reconfiguration is possible when the number of gates in one level of a netlist is far less than the number of cells in one PE column, for example; when the condition $fG \leq P$ is satisfied where f is an integer number (possibly 2^n where n is an integer), G is the number of gates to be simulated at one level in a netlist, and P is the number of PE cells in one PE column. Note that the number of levels of the netlist to be simulated in one simulation cycle is increased by partitioning the PE column into several groups. Therefore, the number of down-loads from the netlist to the PE array is reduced. As a result, the simulation time is decreased significantly. Figures 4 (a) and (b) illustrate how multiple levels in a netlist are mapped onto each group in the PE array by the reconfiguration operation. Note that the circuit's multiple levels in a netlist can be simulated simultaneously in one simulation cycle.

Assume that $\dot{L}$ is the number of circuit levels in the netlist and D is the number of the down-loads from the netlist to the PE array without reconfiguration, and D_r is the number of the down-loads with reconfiguration. Then, $D = \dot{L}/C$ where C is the number of PE columns in the PE array. Therefore, if only one PE column is used, $C = 1$ and $D = \dot{L}$. Meanwhile, D_r is always less than or equal to D with reconfiguration, and is equal to D without reconfiguration. The conditions, $D_r \leq D \leq \dot{L}$ and $D = \dot{L}/C$ show the relationship between the number of circuit levels and the number of PE columns, including the reconfiguration. Figure 5 shows examples of a reconfigurable PE array. Firstly, (a) shows the original PE array which has



(a) Two Levels are mapped onto the PE Array (P >= 4G)

(b) Four Levels are mapped onto the PE Array (P >= 4G)

FIGURE 4 PE Array Reconfiguration

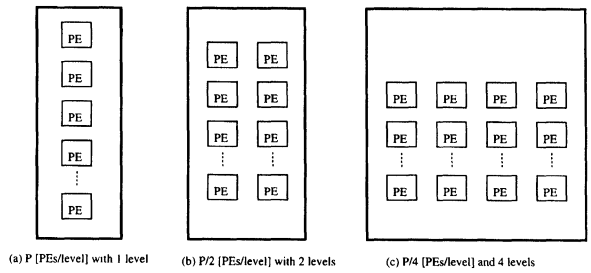


FIGURE 5 Example of PE array Reconfiguration

P PE elements in one level. Secondly, (b) gives an example of reconfiguration by a factor of two, hence (b) has half the number of PE elements in one column and two levels of elements. Lastly, (c) provides reconfiguration by a factor of four, hence it has $1/4 \times P$ PE cells per one column and 4 levels. Note that with the same total number of PE elements, only one circuit level can be executed in (a), whereas two and four circuit levels can be executed in (b) and (c), respectively. Therefore, due to the PE array reconfiguration operation, the speed-up of examples (b) and (c) compared to (a) can be as much as two and four times faster than (a), respectively.

The main components of the accelerator which have not yet been presented, are the PE array with it's local memory and the signal interface. Before generating the PE array's description, the signals (fanin and fanout) between two gates are specified in terms of their node number and a signal direction (fanin or fanout) at the preprocessing stage. The Array Address Table determines the array location so that elements in a netlist level are mapped to a PE array based on an offset from the base address for that PE block.

The signal interface located in each PE element has to provide an efficient interconnect structure. This interconnection topology maps arbitrary fanout patterns between successive logic levels, including feedback paths. Each element in a PE array routes the fanin/fanout signals, utilizing a non-directional, four-way redundant, nearest neighbor, interconnect structure.

If the total number of PE columns is less than the total number of circuit levels in the netlist, or if the total number of PE cells in one PE column is less than the number of gates in one level of the netlist, then stacking of the memory is required (the former is a horizontal stacking and the latter is a vertical stacking). In the horizontal stacking, the depth of the memory stacking is based on the predicted maximum number of levels of the net (It is reported that the maximum number of levels for commercial combinational nets is 64 levels [8]). For example, if the accelerator has only one column, then the stacking capacity of 64 (or 2^6) nodes will be allocated for the accelerator PE memory. It is assumed that this architecture has two PE columns, with a 32 stacking capacity for each column on each chip. In the vertical stacking, the simulation is executed repeatedly by a load partitioning algorithm, which is carried out at the preprocessing stage in the host computer.

The required memory space for one PE cell is 41 bits, as shown in Figure 6. This information includes: East-West (E-W) I/O switches (4 bits), Boolean Equation Unit (BEU) input switch (2 bits), North-South (N-S) I/O wire breaks (4 bits), Gate Function (3 bits), Output Saved Flag (1 bit), Output Value (1 bit), Fanin values of a Good Circuit (2 bits), First Fault ID (9 bits), First Fault Flag (1 bit), Fanin value of the First Faulty Circuit (2 bits), Second Fault ID (9 bits), Second Fault Flag (1 bit), and Fanin value of the Second Faulty Circuit (2 bits). Therefore, the local memory requirement for each PE cell will be 2^5 (for stacking the memory in two columns) times 2^6 (currently, only 41 bits are used for a Node Descriptor) or a total of 2^{11} bits.

The block diagram of a PE cell is shown in Figure 7. The PE cell consists of the Node Descriptor Memory, Input/Output Crossbar Switch, Boolean Evaluation Unit (BEU), Output Latch/Comparator, and Fault Handler.

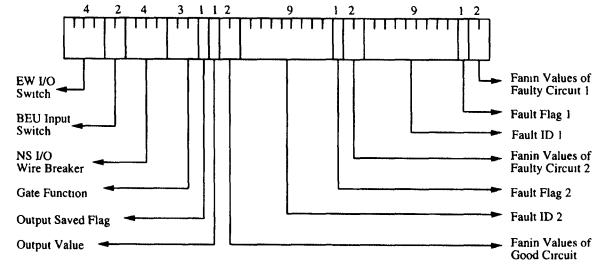


FIGURE 6 The Contents of the Node Descriptor Memory

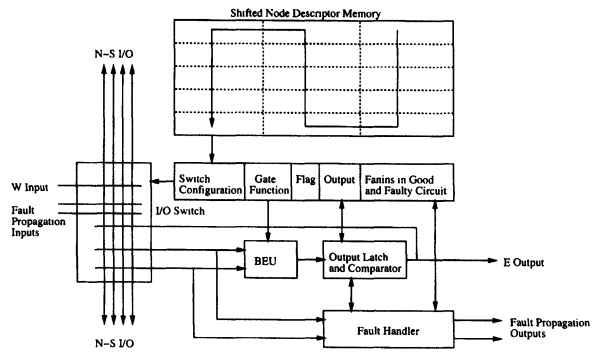


FIGURE 7 PE Cell Block Diagram

The description of each component of the PE cell is provided according to the sequence of the two steps of fault simulation which this architecture executes, namely, a good circuit simulation and a faulty circuit simulation.

The first step is the good circuit simulation. In this case, input signals are mapped onto output signals by the I/O Crossbar Switch. The I/O Crossbar Switch can be implemented with MOS pass transistors acting as simple crosspoint switches. The Gate function (one of three bit operand in the Node Descriptor Memory word) selects the available boolean evaluation function of the BEU. The BEU can implement any boolean function of two inputs.

In this architecture, the BEU can perform eight boolean operations. These eight operations can be customized according to a user's application. The basic operation of the BEU is to evaluate the specified function as specified by one of the eight boolean operations of two input values arriving from the crossbar switch routing.

Input signals from the previous PE cell (west bound) can be propagated in two directions, horizontally (E-W) or vertically (N-S). Along the horizontal axis (E-W), the input signals can either pass through the node (current PE cell) or switch onto the two gate inputs. On the other hand, along the vertical axis (N-S), input signals can pass through or switch to another node in the same level under control of the value in the netlist.

This algorithm allows a maximum of two inputs to one gate. More or less inputs are handled in the preprocessing phase, as will be explained. Functions of fewer than two input variables can be implemented by the process by tying-off unspecified inputs such that the original function is not altered. This situation could be encountered when the gate (or node) has only one input, such as: dummy gates for fanout branches, one input inverters, or one input pass transistors. The combining capability can be implemented by hardware in the PE cell according to the configuration specified by the preprocessor.

As a last step in the good circuit simulation, if the Output Saved Flag (F) is cleared, the output value of the gate and the two input values of the fanin signals of each node are stored in the Node Descriptor Memory by the Output Latch/Comparator, and the Fault Handler, respectively. The Output Saved Flag (F) is set after the storing of the output and fanin values is complete.

The second step in the fault simulation methodology is the faulty circuit simulation. In this case, the F flag is set to one and the Fault Handler generates the faults based on the value of the fanins of the good circuit simulation. An inserted fault is fed into the two inputs of the BEU and the output of the BEU is compared to the good circuit value. If the result is different from the original output value, the fault is propagated to the next PE cell. All propagated faults from each PE are shifted toward the primary outputs (POs) and the POs store the faults in the Fault List Memory. The host computer manages the Fault List Memory and prepares for fault dropping and PE array reconfiguration, if they are needed, and updates the detected fault list with the simulation patterns. When the host computer determines that the time for fault dropping or PE array reconfiguration has been reached, it transmits a signal to the Control Unit.

As we have shown, the simulation cycle for this architecture consists of two stages; the good circuit simulation cycle and the faulty circuit simulation cycle. After the Node Descriptor Memory has been read, the cycle time of the good circuit simulation (t_{good}) is the summation of the time of the following four operations; 1) memory load time (t_l), 2) the transmission time (t_t) of the fanout signal between PEs, 3) the evaluation of the BEU and latching time (t_e) of the gate output, and 4) the write back time (t_w) for restoring the result of the output value into the Node Descriptor memory. Therefore, the simulation

time for a good circuit simulation in one PE is given by,

$$t_{good} = t_l + t_t + t_e + t_w$$

Obviously, the total simulation cycle time is proportional to the number of levels in the circuit. The evaluation of the BEU and latching time (t_e) are generated in two phases. In the first phase, the gate function field of the Node Description Memory word and the two inputs are fed into the BEU. In the second phase, the results from the BEU are stored in the Output Latch. The signal transmission time (t_t) is generated by the wire delays between the chips and/or boards.

The simulation time of the faulty circuit (t_{faulty}) for one pattern is calculated by adding the values of the following operations: 1) memory load time (t_l), 2) fault insertion time (t_i), 3) the evaluation of the BEU and latching time (t_e), 4) comparison time (t_c) of the output to the good circuit value, and 5) fault ID and cell address set up time (t_p). Since two faults are handled at all times, operation 2), 3), 4), and 5) are executed twice. Therefore, the simulation time for one simulation pattern for the faulty circuit is given by,

$$t_{faulty} = t_l + 2 \times (t_i + t_e + t_c + t_p)$$

In addition to these two main simulation times, when the Control Unit performs fault dropping, the times for checking the fault status of each PE and for sending the result to the host computer would be required. However, it is assumed that these times are included in preprocessing and are not of importance for the simulation timing analysis.

The main goal of this architecture is to achieve high integrated chip capability with lower cost. In order to generate the number of PE cells which can be stored on one chip, the data for the maximum number of gates/chip is in the range of 2^{20} for current technology [8]. There are many possible ways to design the memory cell; including a static six-transistor cell, a dynamic four-transistor cell, three-transistor cells, or a one transistor cell [13]. In this paper, the three-transistor cell is used. It has been reported that the hardware control circuit part occupies far less chip space than the memory [8], therefore, only the memory space need be considered. Since there are 41 bits of memory space, the total number of cells per node descriptor is 41 × 3 transistors. Therefore, if we use 32 descriptors in each of two columns as a stacking memory, the total number of transistors is $(41 \times 3 \times (32 + 1)) = 2^{12}$ in each column. Compared to this, the area requirements for switch selection, evaluation, output comparison, and fault handling, are negligible.

Based on previous data [8], the total number of PE cells comes out to be $2^{20-12} = 2^8$ PEs per chip. 32 netlist Node Descriptors per PE cell column block are used, so the total number of netlist nodes per chip is $2^8 \times 2^5 = 2^{13}$. If these chips are packed at a 16 chip/board rate, 16 boards per chassis, and two chassis per system, then the total netlist nodes per one system is $2^{13} \times 2^4 \times 2^4 \times 2^1 = 2^{22}$, or four million netlist nodes.

ALGORITHM

In order to achieve high performance, using the new architecture, a new simulation algorithm was developed. To maximize the capability of the new algorithm, preprocessing is required. During preprocessing, two stages of circuit expansion are performed. This is done in order to maximally use the parallelism by expanding a circuit into a new circuit with each element having at most two faults. Also, after expansion, the preprocessing for the PE array reconfiguration is performed.

In the first stage of circuit expansion, any gate which has more than two fanins is expanded to gates having only two fanins. Consider the circuit shown in Figure 8. After expanding the circuit, the new circuit is shown in Figure 9. After expansion, if we apply fault collapsing, the number of faults in the new circuit is 14 and the number of faults in the example circuit is 13. The difference is that there is an uncollapsed fault, i.e., the signal from G_TEMP1 to G_TEMP2 stuck-at-0. This is because we only consider fault equivalence relationships. If we considered

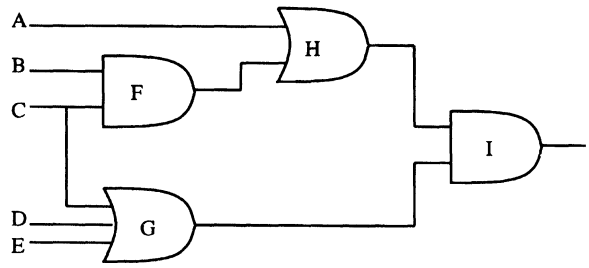


FIGURE 8 Example Circuit

fault dominance relationships, the two circuits would have the same number of faults. However, in fault collapsing, we can force the new faults to be ignored. This is like treating a gate as a functional model during fault collapsing. In the expanded model, only pin output signal faults and pin input signal faults are considered, and the internal faults are discarded. The object of this preprocessing is to equally distribute 2 faults to each element.

Consider the XOR gate expansion shown in Figure 10 (a). Since all internal faults are not considered, there are only 6 faults. Therefore, after the expansion, the number of faults is the same and the difference is only in the method of internal expression, since the propagation and excitation conditions of all faults are the same. An XNOR gate is expanded in a similar way, as shown in Figure 10 (b).

There are several ways to expand a multi-input gate to a set of 2-input gates. Since the number of levels in the circuit is an important factor for the overall performance of the new algorithm, the expansion is performed by choosing a set of gates so as to minimize the number of levels. For example, the 4 input AND gate of Figure 11 (a) is expanded to two levels of gates as shown in (c), rather than to three levels of gates as shown in (b).

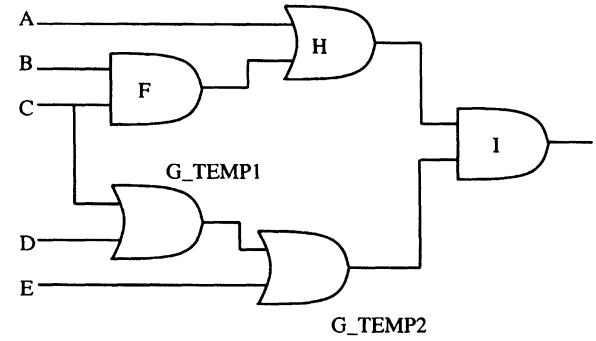


FIGURE 9 Example Circuit after First Expansion

In the second stage of circuit expansion, all fanout branches are regarded as dummy gates. The object of this preprocessing is to force the number of faults in a gate to always be less than or equal to 2, and to have all faults inserted on the input side of the gates. After considering all fanout branches, the example circuit is shown in Figure 12.

Notice that after expansion, the number of faults in the expanded circuit is the same as in the original circuit, and the number of faults inserted on the gates is distributed as uniformly as possible. The minimum insertion is 0 and the maximum insertion is 2. Since the number of fanins of any gate is 2, the number of propagated faults towards the next level gate is at most 2. Furthermore, the maximum number of faults on any gate is known and the required memory space can be optimized.

Let x be the number of XOR or XNOR gates in the circuit and let a be the number of AND, NAND, OR or NOR gates in the circuit, respectively. Also, let f be the number of fanout branches in the circuit. Then, there are $(x + a)$ gates in the circuit before expansion. Without loss of generality, let $g_1, g_2, \ldots, g_x$ be XOR or XNOR gates and $g_{x+1}, g_{x+2}, \ldots, g_{x+a}$ be AND, NAND, OR or NOR gates. Let $n(g_c)$ be the number of fanins of a gate c. Since all fanout branches are regarded as dummy gates, there are f more gates after the expansion. After expansion, the number of XOR or XNOR gates after expansion is given by $\Sigma_{c=1}^{x}(n(g_c) - 1) \times 5$. The number of AND, NAND, OR or NOR gates is given by $\Sigma_{c=x+1}^{x+a}(n(g_c) - 1)$. Therefore, the total number of gates in the circuit after expansion is given by $f + \Sigma_{c=1}^{x}((n(g_c) - 1) \times 5) + \Sigma_{c=x+1}^{x+a}(n(g_c) - 1)$.

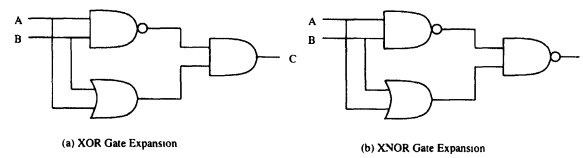


FIGURE 10 Expansion of an XOR Gate and an XNOR Gate

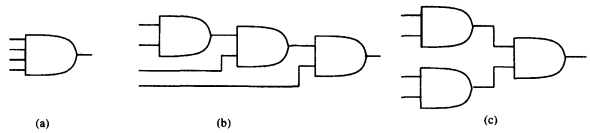


FIGURE 11 Possible Expansions of 4 Input AND Gate

After two stages of expansion, preprocessing for the PE array reconfiguration is performed according to the number of gates in a level. PE array reconfiguration is critical, since the number of levels to be handled at a given time can increase. Since this makes most of the PEs busy, PE array reconfiguration directly affects the performance. In this preprocessing, the optimal PE array reconfiguration is derived according to the circuit topology. The time of preprocessing for the PE array reconfiguration is of the complexity of $\mathcal{O}(L)$.

The algorithm for the new fault simulation is shown in Figure 13. Initially, a circuit is simulated without any faults and the results are stored. Then after inserting the first fault, fault evaluation is performed. If the value is different from that of the good simulation, it is propagated. For the second fault, the same processing is performed. Since there are only 2 faults in the fault list, the faults in the fault list of a gate are dropped or propagated to the next level, after this procedure. Since the faults in the first level are propagated to the next level, or dropped after the first stage, the gates in the first level are not evaluated any further. The above procedure is continued until all the given simulation patterns are considered.

The main bottleneck of conventional concurrent fault simulation is that if there is a large number of faults in the circuit, the number of faults attached to a gate during the simulation becomes large. If this is applied to a parallel architecture, it degrades the overall performance, since some processing elements are very busy but others are idle. Since this algorithm distributes the number of faults over the circuit, in each gate evaluation, there are always at most 2 faults in a gate's fault list. Therefore, this algorithm takes better advantage of the parallelism of the architecture.

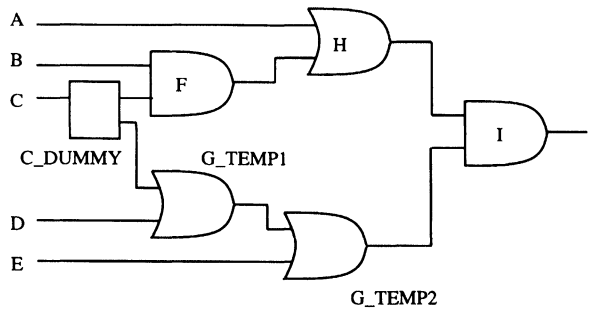


FIGURE 12 Example Circuit after Second Expansion

```
make a fault list attached to each element
for all patterns
    perform good simulation
    store the results of good simulation
    for all levels
        insert the first fault in the fault list
        simulate the fault
        compare the value
        if the value is different from that of good simulation
            propagate the fault to the next level
        else
            drop the fault
        insert the second fault in the fault list
        simulate the fault
        compare the value
        if the value is different from that of good simulation
            propagate the fault to the next level
        else
            drop the fault
    end
end
```

FIGURE 13 New Fault Simulation Algorithm using New Architecture

The algorithm is highly dependent on the number of levels in the circuit and the number of columns in the array architecture. If the architecture has L columns and a given circuit has less than L levels, then simulation can be done in less than $(L - 1)$ steps. These procedures can be efficiently performed in a parallel array architecture. Since the good simulation is performed before fault evaluation and single fault propagation is executed at each PE, this algorithm takes advantage of the parallelism using parallel patterns. Also, the handling of fault lists is another advantage over the handling of one fault at a time since it reduces the unnecessary overhead.

Consider the circuit shown in Figure 14. There are two gates (A_DUMMY and B_DUMMY) in level 1, two gates (C and D) in level 2, and two gates (C_PO

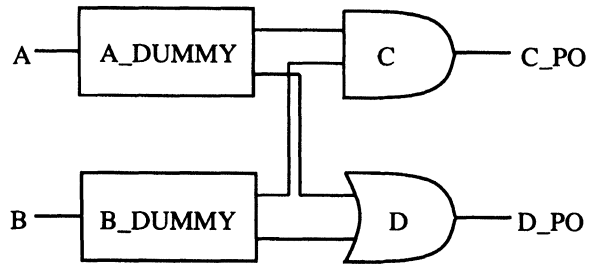


FIGURE 14 Simple Example Circuit

and D_PO) in level 3. During simulation, the contents of each processing unit is shown in Figure 15.

P(1,1)	P(2,1)	P(3,1)
A_DUMMY	C	C_PO
1	0	0
A-s-0	A_DUMMY-s-1	C-s-1
A-s-1	B_DUMMY-s-1	C-s-0

P(1,2)	P(2,2)	P(3,2)
B_DUMMY	D	D_PO
0	1	1
B-s-0	A_DUMMY-s-0	D-s-1
B-s-1	B_DUMMY-s-0	D-s-0

(a) After Good Simulation

P(1,1)	P(2,1)	P(3,1)
	C	C_PO
	0	0
	A-s-0	B_DUMMY-s-1
	B-s-1	

P(1,2)	P(2,2)	P(3,2)
	D	D_PO
	1	1
	A-s-0	A_DUMMY-s-0
	B-s-1	

(b) After One Level Consideration

P(1,1)	P(2,1)	P(3,1)
		C_PO
		0
		B-s-1

P(1,2)	P(2,2)	P(3,2)
		D_PO
		1
		A-s-0

(c) After Two Level Consideration

FIGURE 15 Processing Elements in Each Step

For simplicity, only the element name, the good simulation value, and the contents of the fault list are represented. Let (1,0) be the input pattern for simulation, and let the architecture have 2×3 processing elements. After preprocessing and good simulation of the pattern (1,0), the contents of each processing element is shown in Figure 15 (a). Consider P(2,1) as an example. The processing element P(2,1) represents the level 2 gate C. After good simulation, the result is stored as 0. Then, 2 faults (A_DUMMY-s-1 and B_DUMMY-s-1) related to gate C are inserted. After evaluation for the fault A_DUMMY-s-1, the effect is not propagated, since the value is the same as the good simulation value. However, since the evaluation result for the fault B_DUMMY-s-1 is different from the good simulation result, it is propagated to the next level. Therefore, B_DUMMY-s-1 is shown in the list of P(3,1), as shown in Figure 15 (b). After one level is considered, the contents of each processing element is shown in Figure 15 (b). Since all faults in the first level are considered and the next level processing elements have the fanin values of the good simulation, the first level elements are not considered any further. Therefore, when the second step is considered, the first column of processing elements is idle. The number of these idle processing units becomes larger as the process continues. However, these idle processing elements can be used to perform simulation of the next patterns, if available. Therefore, the processing units are busy most of the time during simulation. After the consideration of two levels, the contents of each processing element is shown in Figure 15 (c).

PERFORMANCE EVALUATION

It is very difficult to provide precise timing for this new algorithm, since this is highly dependent on the circuit's topology. However, a rough comparison between the new algorithm and one fault simulation algorithm using the same architecture can be made as follows. The algorithm which will be compared to the new algorithm, is based on logic simulation. In other

words, after insertion of each fault, the evaluation is performed as in logic simulation. This does not require much additional overhead. We will call this a 'conventional algorithm'. Also, an accelerator using this conventional algorithm will be called a 'conventional accelerator', for notational convenience.

Let L be the number of levels in the circuit and $\dot{L}$ be the number of levels after the expansion. For simplicity, only two cases are considered. One is that there is only one column in the architecture and the other is that there are $\dot{L}$ columns. Also for simplicity, fault dropping is not considered. Let P be the number of given simulation patterns and F be the number of faults in the circuit. If a conventional fault simulation algorithm is used, using one column, the total simulation time is given by

$$L \times (F \times S + G) \times P \tag{1}$$

where S represents the time including fault evaluation and comparison, and G represents the good simulation time. If fault simulation is executed using the method of logic simulation, using $\dot{L}$ columns, the total simulation time is given by

$$(F \times S + G) \times P \tag{2}$$

Consider the new simulation algorithm. If there is only one column in the array architecture, the simulation time for a pattern is given by

$$2 \times \frac{(\dot{L} - 1) \times (\dot{L} - 1 + 1)}{2} \times S + G = \dot{L} \times (\dot{L} - 1) \times S + G$$

In this case, the total simulation time (worst case) is

$$(\dot{L} \times (\dot{L} - 1) \times S + G) \times P \tag{3}$$

However, since there are $\dot{L}$ columns and this algorithm works in a pipelined fashion, a simulation time for a pattern is given by $(\dot{L} - 1) \times S + G$. Therefore, the total simulation time is given by

$$((\dot{L} - 1) \times S + G) \times P \tag{4}$$

This is the worst case since fault dropping is not considered. Since the new algorithm deals with many faults simultaneously (in parallel), the fault dropping speed of the new algorithm is faster than that of the conventional one.

Comparing equations (1) and (3) shows that for the case where there is only one column; if the following condition is satisfied, the new algorithm is more efficient.

$$\dot{L} \times (\dot{L} - 1) < L \times F \tag{5}$$

When comparing the two equations (2) and (4) in the case of $\dot{L}$ columns, it can be easily shown that the new algorithm is more efficient, if the following condition is satisfied.

$$\dot{L} - 1 < F \tag{6}$$

The conditions, shown in equations (5) and (6), are true for most practical circuits.

As shown in previous sections, the fault simulation cycle consists of two timing parameters; the timing value for a good circuit simulation and the timing value for a faulty circuit simulation. Assume that the times t_l, t_t, and t_e are the same as the system cycle, t_{sys}. Also, let the time t_w, which is also not more than a system cycle in modern microprocessor architectures, especially in a RISC architecture, be the same as t_{sys}. Therefore, the simulation cycle of a good circuit, t_{good} is defined as $t_l + t_t + t_e + t_w = 4 \times t_{sys}$. On the other hand, the simulation cycle of a faulty circuit t_{faulty} is $t_l + 2 \times (t_i + t_e + t_c + t_p)$. Ideally, if every time parameter is equivalent to one system cycle, then t_{faulty} is given by

$$t_{faulty} = 9 \times t_{sys}$$

In order to compare the simulation cycles of the new accelerator to a conventional one, assume that there exists enough PE array elements for fault simulation (i.e., enough PE columns). Thus, the total simulation time for one pattern in the new architecture (t_{NEW}) is

$$t_{NEW} = t_{good} + (\dot{L} - 1) \times t_{faulty}$$
$$= (9\dot{L} - 5) \times t_{sys}$$

The PE array reconfiguration capability provides a speed-up of

$$S_{pr} = \frac{D}{D_r}$$

where D and D_r are the numbers for down-loading to the PE array, without and with reconfiguration, respectively. Since the conditions, $D_r \leq D \leq \dot{L}$and $D = \dot{L}/C$, where C is the number of PE columns, exist, the total simulation time of the new algorithm, after adding the speed-up factor of the reconfiguration is

$$t_{NEW} = (9D_r - 5) \times t_{sys}$$

On the other hand, consider a conventional accelerator, where every single fault is inserted in a simulation circuit and a simulation method similar to a good circuit simulation is executed. For this case, the total simulation time per pattern is

$$t_{HARD} = (t_{good} + t_i) \times F = 5t_{sys} \times F$$

Hence, the speed up factor of the new algorithm over a conventional one is given by

$$S_{NEW} = \frac{t_{HARD}}{t_{NEW}} = \frac{5t_{sys} \times F}{(9D_r - 5) \times t_{sys}} = \frac{5F}{(9D_r - 5)}$$

The complexity of the new algorithm is $\mathcal{O}(D_r^2)$, if only one level of circuit elements is considered at a time. However, when all levels are considered simultaneously, the complexity of the new algorithm is $\mathcal{O}(D_r)$.

RESULTS

Fault simulation results for the ISCAS 85 combinational benchmark circuits [14] and the ISCAS 89 sequential benchmark circuit [15] are shown in Table I. In this table, in order to recognize the additional overhead of the expansion, GOR, LOR, and the speed up factor of the PE array reconfiguration S_{pr}, are represented. Also, the speed up ratio is represented in the last column. These numbers have been derived by dividing the software simulation time for the given examples by the calculated hardware simulation time for these examples.

GOR (Gate Overhead Ratio) is represented as

$$\frac{\text{the number of gates in the expanded circuit}}{\text{the number of gates in the original circuit}}$$

LOR (Level Overhead Ratio) is defined as

$$\frac{\text{the maximum level in the expanded circuit}}{\text{the maximum level in the original circuit}}$$

For example, circuit c499 uses many XOR gates. Therefore, it has a high GOR and LOR. The main factor affecting the overall performance is the LOR, rather than the GOR, since this architecture depends highly on the number of levels in the circuit. After circuit expansion, according to the number of gates in a level, the PE array is rearranged in the processing scheme and not in the actual hardware. This PE array reconfiguration makes the architecture achieve more efficient fault simulation, since it increases the number of levels to be handled at one down-loading time. In these results, the PE reconfiguration is done in 4 depths, i.e. from 256 × 1, 128 × 2, 64 × 4, and 32 × 8, in a chip. The average S_{pr} is 4.44 for these circuits. To compare the performance of the new approach, a software approach was developed using the same environment. For software fault simulation, a parallel pattern algorithm [16][17], with fanout free region analysis [18][19], was used. For fault simulation, 1024 random patterns were used. The new approach averages about 30,000 times faster than the software fault simulation, with a 25 *nsec* clock cycle, as shown in Table I.

The configuration of the hardware accelerator used for these results, is as follows. Since the architecture is based on an array, there is no additional overhead

TABLE I Fault Simulation Results for Benchmark Circuits

Circuit	GOR	LOR	S_{pr}	Speed-up Ratio (40 *MHz*)
c432	2.07	2.47	5.88	23800.00
c499	2.89	2.46	4.57	26960.78
c880	1.97	1.58	5.86	47254.90
c1355	1.49	1.65	4.78	51470.59
c1908	1.60	1.57	4.40	27938.46
c2670	1.40	1.77	3.53	68967.00
c3540	1.51	1.53	3.75	38650.09
c5315	1.56	1.39	2.45	50195.38
c6288	1.59	1.74	5.76	22676.38
c7552	1.48	1.42	1.83	55138.54
s208	1.35	1.35	4.60	10882.35
s298	1.59	1.46	5.33	13235.29
s344	1.25	1.36	6.00	17450.98
s349	1.25	1.41	6.00	17156.86
s386	1.56	1.21	4.25	8980.39
s420	1.36	1.52	5.88	25294.12
s444	1.55	1.69	4.40	22500.00
s510	1.44	1.29	3.60	16764.71
s526	1.74	1.46	4.00	21372.55
s641	1.24	1.26	6.93	8888.89
s713	1.30	1.38	7.00	10660.66
s820	1.76	1.42	2.43	15784.31
s832	1.79	1.50	2.25	15980.39
s838	1.38	1.56	7.08	17807.81
s953	1.42	1.56	4.67	77083.33
s1196	1.43	1.33	5.14	37254.90
s1238	1.49	1.36	4.75	37549.02
s1423	1.26	1.61	6.13	27567.57
s1488	1.38	1.15	2.88	20490.20
s1494	1.39	1.20	3.00	21078.43
s5378	1.42	1.39	1.50	33098.24
s9234	1.23	1.22	1.63	61328.13
Average				29805.28

due to the architecture expansion. If the accelerator has 2^8 PEs in one chip, 16 chips/board, 16 boards/chassis, and two chassis per system, the total number of PEs is 2^{17}. For larger circuits, the system would provide superior performance. Notice that 256 PEs per chip is based on current design technology (3 transistors for a memory cell). As the design technology progresses, it will be possible to design a memory cell using less than 3 transistors. Therefore, the number of PEs per chip can be much higher.

The cost of the hardware accelerator is estimated based on a manufacturing cost of expected small quantity chips plus the cost of a general purpose workstation system as a host computer [8]. The software simulation cost is estimated as a sum of software costs and the workstation cost. Let W be the cost of the host computer, C be the cost of a chip, and S be the cost of the software program, respectively. If 64 chips (each chip having 2^8 PE elements) are used, the total cost of the hardware accelerator is $64 \times C + W$. On the other hand, the cost of a software simulator is given by $W + S$.

The Cost Performance Ratio is defined as the obtainable simulation performance using a hardware accelerator, when the obtainable software simulation performance is assumed to be 1. In other words, the Cost Performance Ratio is defined as the software cost divided by the hardware cost, multiplied by the speed up ratio,

$$\text{Cost Performance Ratio} = \frac{\text{Software Simulation}}{\text{Hardware Accelerator}} \times \text{Speed Up Ratio} = \frac{W + S}{64 \times C + W} \times \text{Speed Up Ratio}$$

Currently, the cost of a hardware accelerator is higher than the cost of a software simulation system, i.e., $(64 \times C + W) > (W + S)$, hence the Cost Performance Ratio is less than the Speed Up Ratio. When the average speedup is considered, the total cost performance is about 19,000 times better than a conventional software simulator (In this computation, W, C, and S are assumed to be \$30,000, \$1,000, and \$30,000, respectively.).

CONCLUSION

The demand for a high speed, low cost, fault simulator has led to the need for a new computer architecture and a new simulation algorithm. Our objective for developing this hardware accelerator for fault simulation was to satisfy this demand.

The architecture of the new hardware accelerator was designed based on a reconfigurable mesh type PE array. The netlist is directly mapped onto a massively parallel PE array, which is composed of a tightly connected array of simple processing ele-

ments. Circuit elements to be simulated at the same level in the netlist are executed concurrently as in a pipelined process.

The new parallel simulation algorithm expands the gates to two input gates, which permits us to limit the fault number to two at each input signal to the gates, so that the faults can be spread out uniformly throughout the PE array. The PE array reconfiguration operation provides a great advantage for simulation speed, since it utilizes each PE cell most of the time. Another advantage of this algorithm is that it is possible to predict the total memory size used during the simulation, which is a big drawback of conventional concurrent simulation programs.

Simulation results, based on benchmark circuits, show that the hardware accelerator, in a multi-chip mode, would be orders of magnitude faster than a software simulation program on a general purpose computer. The performance of this architecture can be improved further by new technologies, such as; adopting a one-transistor in a memory cell model and using a fast system clock. In addition to that, the size of the proposed hardware accelerator can be reconfigurable for maximum cost performance; from the chip level to the system level, based on the size of the circuit to be simulated.

References

[1] N. Ishiura, H. Yasuura, and S. Yajima, "High Speed Logic Simulation on Vector Processors," *IEEE Trans. Computer-Aided Design*, pp. 305–321, May 1987

[2] R. Barto, and S. A. Szygenda, "A Computer Architecture for Digital Logic Simulation," *Electronic Engineering*, pp. 35–66, Sep. 1980

[3] G. Pfister, "The Yorktown Simulation Engine: Introduction," *DAC*, pp. 51–54, 1982

[4] Y. Kitamura, T. Hoshino, T. Kondo, T. Nakashima, and T. Sudo, "ch. 3 Hardware Engines for Logic Simulation," *Logic Design and Simulation, Advances in CAD for VLSI vol. 2*, 1986

[5] T. Blank, "A Survey of Hardware Accelerators used in Computer-Aided Design," *IEEE Design and Test of Computers*, pp. 21–39, Aug. 1984

[6] M. Abramovici, "A Logic Simulation Machine," *DAC*, pp. 65–73, 1982

[7] P. Agrawal, and W. Dally, "A Hardware Logic Simulation System," *IEEE Trans. on CAD*, pp. 19–29, 1990

[8] S. Fehr, "An Array-Based Hardware Accelerator for Digital Logic Simulation," *Dissertation, The University of Texas at Austin*, 1992

[9] Y. Levendel, P. Menon, and S. Patel, "Parallel Fault Simulation Using Distributed Processing," *The Bell System Technical Journal*, pp. 3107–3137, 1983

[10] A. Stein, D. Saab, and I. Hajj, "A Special Architecture for Concurrent Fault Simulation," *ICCD*, pp. 243–246, 1986

[11] P. Agrawal, V. Agrawal, K. Cheng, and R. Tutundjian, "Fault Simulation in a Pipelined Multiprocessor System," *ITC*, pp. 727–734, 1989

[12] S. Bose, and P. Agrawal, "Concurrent Fault Simulation for Logic Gates and Memory Blocks on Message Passing Multicomputers," *DAC*, pp. 332–335, 1992

[13] N. Weste and K. Eshraghian, "Principles of CMOS VLSI Design: A Systems Perspective," *Addison Wesley*, pp. 353–354, 1985

[14] F. Brglez, and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," *ISCAS*, pp. 695–698, 1985

[15] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *ISCAS*, pp. 1929–1934, 1989

[16] J. Waicukauski, E. Eichelberger and D. Folenza, "A Statistical Calculation of Fault Detection Probabilities by Fast Fault Simulation," *ITC*, pp. 779–784, 1985

[17] N. Gouders and R. Kaibel, "PARIS: A Parallel Pattern Fault Simulator for Synchronous Sequential Circuits," *ICCAD*, pp. 542–545, 1991

[18] S. Hong, "Fault Simulation Strategy for Combinational Logic Networks," *FTCS*, pp. 96–99, 1978

[19] O. Song and P. Menon, "Parallel Pattern Fault Simulation based on Stem Faults in Combinational Circuits," *ITC*, pp. 706–711, 1990

Authors' Biographies

Sungho Kang is an assistant professor of the Electrical Engineering Department at Yonsei University, Seoul, Korea. He was a senior staff engineer at the Semiconductor Systems Design Technology, Motorola Inc., a research scientist at Schlumberger Laboratory for Computer Science and a post doctoral fellow at the University of Texas at Austin. He received his B.S. from Seoul National University at Seoul, Korea, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Texas at Austin, respectively. Dr. Kang's research interests include design verification, fault simulation, testing and design for testability.

Youngmin Hur is a post doctoral fellow in electrical and computer engineering at the University of Texas at Austin. He received the B.S. degree in electronics engineering from Hanyang University, Seoul,

Korea, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, in 1991 and 1995, respectively. From 1978 to 1987, he was with the Agency for Defense Development, Korea. Mr. Hur's research interests are in the area of computer architecture and digital simulation.

Stephen A. Szygenda is a professor and Chairman of the Electrical and Computer Engineering Department at the University of Texas at Austin. He also hold the Clint Murchison Sr. Chair of Free Enterprise. He is a pioneer in the areas of Simulation, CAD, Fault Tolerant Computing, and Software Engineering; with 30 years of experience, dating back to the early 1960's at Bell Telephone Laboratories. During that period he was involved in the development of the first generation CAD tools for the #1 ESS. After leaving Bell Labs. and joining the faculty at the University of Missouri, he began the development of the TEGAS system. This work was later carried on at SMU and the University of Texas, where he joined the faculty at 1972. In 1979, he left the university environment to form a company, CCSS, which was the first multi-product simulation and test company, producing very large software and hardware systems. In 1980 this company was sold to COMSAT, where Dr. Szygenda remained as president until its subsequent sale to G.E., in 1983. At that time he founded the Rubicon Group; a unique high technology incubator. He rejoined the faculty of the University of Texas at Austin in 1986. He received his M.S. and Ph.D. degrees from Northwestern University, and has been active in numerous areas of the IEEE.