Proc. Natl. Sci. Counc. ROC(A)
Vol. 21, No. 3, 1997. pp. 189-199

VLSI Design of a Cellular-Automata Based Logic and Fault Emulator

YIH-LANG LI, YING-CHAO LAI, AND CHENG-WEN WU†

Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.

(Received November 23, 1995; Accepted December 23, 1996)

ABSTRACT

With advances in VLSI technology and the decline in hardware costs, special-purpose machines have become more and more popular for a wide range of applications. We have proposed a unilateral 2-D cellular automata (CA) model which can be treated as an SIMD parallel architecture. In this paper, we present a CA chip which realizes our previously proposed CA model for highly parallel logic and fault simulation. By using pipelining, our CA simulation engine achieves a simulation rate of more than one billion gate evaluations per second using a 20 MHz clock and 8-bit words. The CA architecture allows easy scaling; i.e., each CA chip can directly communicate with its four neighboring CA chips to construct a larger cellular array. Our CA chip operates in either initialization or simulation mode and can perform logic and fault simulation. It produces one output in every six clock cycles after the pipeline has been filled. Compared with previously reported parallel fault simulators, its performance is superior by about three orders of magnitude.

Key Words: cellular automata, fault simulation, graph embedding, layered network, logic simulation, SIMD

I. Introduction

Fault simulation is important in logic circuit design and testing. The quality of a test sequence depends on its fault coverage as well as length, which determine the test effectiveness and efficiency for the circuit under test (CUT). Fault simulation is used in automatic test pattern generation (ATPG) to evaluate the fault coverage of generated test patterns, and is also used to analyze the faults of a CUT detected by these test patterns; i.e., fault dictionaries can be constructed with information for locating faults.

The complexity of logic and fault simulation (per pattern) is $O(n)$ and $O(n^2)$, respectively, where n is the gate count (and roughly the test set size) of the CUT. With the rapid increase in the complexity of VLSI circuits, short simulation time is urgently needed. Acceleration of logic and fault simulation has been under investigation for many years (Armstrong, 1972; Antreich and Schulz, 1987; Maamari and Rajski, 1988; Becker *et al.*, 1992); however, it seems that we can hardly anticipate a guaranteed linear-time fault simulation algorithm (Harel and Krishnamurthy, 1987). Instead of looking for efficient simulation algorithms to run on a sequential machine, some researchers have tried to take advantage of the computation power of parallel processors to boost the simulation rate (Agrawal, 1986; Kitamura, 1986; Ishiura *et al.*, 1987; Agrawal *et al.*, 1989; Huisman *et al.*, 1990; Agrawal and Chakradhar, 1992; Ishiura and Yajima, 1992; Narayanan and Pitchumani, 1992). There are two important issues associated with parallel processing: correctness and parallelism. As many gate evaluations are performed simultaneously, causality must be preserved to ensure the correctness of the results. Event-driven simulation is widely used due to its capability of preserving causality as well as correctly dealing with element delays for both combinational and sequential circuits. From the viewpoint of the grain size of parallel processing, event-driven simulation is much more suitable for coarse-grain computers. Fine-grain parallel processing can not handle event queues easily, especially for large circuits with huge event queues (Gloria and Faraboschi, 1992). Scheduling is an approach to maintain causality in a massively parallel processing (MPP) environment. In Gloria and Faraboschi (1992),

†To whom all correspondence should be addressed.

the preprocessing phase, which consists of partitioning and scheduling, partitions the circuit into several parts to be embedded in different processors and then decides the evaluation order of all the gates and the signal transfer among processors to guarantee the causality property before simulation. Another important issue is circuit parallelism, which directly affects the performance of a parallel processing system based on the circuit partitioning technique. The relation between circuit parallelism and circuit size is unpredictable and nonlinear in general (Bailey, 1992). It is inappropriate to extrapolate the parallelism of larger circuits using the measured parallelism of small circuits (Li and Wu, 1995). In addition to unpredictable circuit parallelism, interprocessor communication overhead is another obstacle for high speed circuit-partitioning based logic and fault simulation. Research results have indicated that it is difficult to reach the theoretical maximum speedup in a multiprocessor system due to the cost of interprocessor communication, which often follows high concurrency (Agrawal, 1986). Pipelining is another efficient technique used to parallelize simulation for both software and hardware implementation (Agrawal *et al.*, 1989; Li and Wu, 1995). We have shown that we can divide a simulation algorithm into several stages and use pipelining, which results in higher simulation rates for larger circuits (Li and Wu, 1995).

Cellular automata (CA) is similar to an SIMD parallel architecture. We have presented a unilateral 2-D CA model to parallelize logic and fault simulation, and shown that a larger circuit has a higher simulation rate. Our CA is better than previous works reported for pure logic simulation (Li and Wu, 1995). CA consists of identical cells whose interconnection topology is a mesh structure, and where each cell can only receive data directly from its neighbors. Each cell is in one state of its state set at each time instant for a synchronized CA. Depending on the current states of its neighbors, the cell follows a set of predefined state transition rules, which simulate the dynamic electrical signal flow through the components of the circuit, to change its current state to a proper state at the next time instant. With the characteristics of simple cell structure, homogeneity, and topological regularity, CA is ideal for VLSI implementation. In this paper, we propose a CA chip architecture and present its VLSI circuit design, which realizes the previously proposed CA model. The CA chip architecture is scalable in both dimensions. It produces one output in every six CA clock cycles after the pipeline has been filled in the initial simulation stage, and outperforms previously reported parallel simulators by three orders of magnitude.

II. Cellular Automata

The details of the CA model on which our chip is based can be found in our previous paper (Li and Wu, 1995). We give a brief summary here. Our CA has a unilateral 2-D structure with eight states. It is synchronous and pipelinable. To demonstrate logic and fault simulation on our CA, we will use the ISCAS-85 benchmark circuit, c17, as an example (see Fig. 1). The preprocess includes transformation of the circuit into one with fanin and fanout equal to two, transformation of the resulting circuit into a layered network (LN), mapping of the LN onto the 2-D CA, and generation of initial cell information, which contains the initial state, gate type, offset value, etc. (Li and Wu, 1995). We show the LN of c17 in Fig. 1(a), where all the buffers are inserted during the transformation process from a leveled circuit to the corresponding LN so as to satisfy the spatial-locality requirement of CA. The mapping of an LN to the 2-D CA is very simple. We make each net, except primary-input nets and primary-output nets, occupy two neighboring cells, one at a fanout column and the other at the corresponding (neighboring) fanin column. Only fanout branches need to be considered; stems can be neglected. In c17, e.g., Net 11 has two fanout branches (Net 14 and Net 15) as shown in Fig. 1(a). In this case, we shall only consider Net 14 and Net 15 in the mapping process. The mapping result is shown in Fig. 1(b), where the

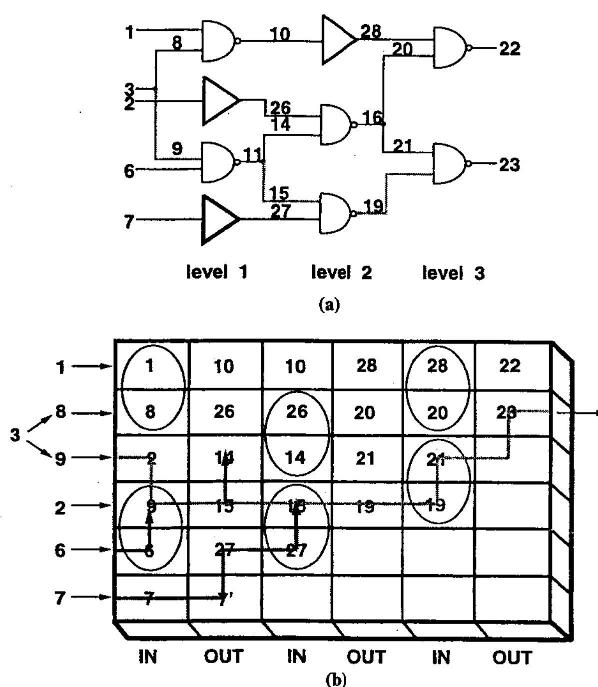

Fig. 1. The mapping of circuit c17.

six ellipse regions correspond to the input cells of the six NAND gates.

As shown in Fig. 1(b), CA cells are classified into two types: fanin cells and fanout cells. Fanin cells receive their input signals from a previous level and pass these signals upwards or downwards to their target cells. Fanin cells are either in the Fanin state or in the BotFanin state. In Fig. 1(b), Cell 1 and Cell 9 are in the Fanin state, and other cells at the same column are in the BotFanin state. Fanout cells are either in the FanoutRecv state or in the Fanout state initially. Those in the FanoutRecv state receive signals from its left column whenever all the gate evaluation results in that column are available. Therefore, the cell whose left neighbor is the top cell of a gate will be in the FanoutRecv state at the beginning. In Fig. 1(b), Cells 10, 14, 15, and 7' are all in the FanoutRecv state, and others in the same column are in the Fanout state. Upon receiving the gate evaluation results, fanout cells distribute them to the target cells according to the fanout number of each gate. When the distribution of all the output signals in a column is finished, those cells in the Fanout (FanoutRecv) state will change their state to NewPipe (NewPipeRecv) in order to inform the right fanin column of the arrival of new input signals. A cell in state NewPipe (NewPipeRecv) will change to the Fanout (FanoutRecv) state at the next time instance to wait for the new result from the next input pattern. The shaded line in Fig. 1(b) illustrates the data flow of simulation inside CA, which corresponds to the shaded signal path in Fig. 1(a). In the figure, Cell 2 receives Signal 9, whose target cell is Cell 9; hence, it sends Signal 9 downwards to Cell 9. After the arrival of Signal 9, Cell 9 receives Signal 6 from Cell 6 and performs the NAND operation of Signal 6 and Signal 9 if Signal 6 is available in Cell 6. In the fanout column, Cell 15, which is in the FanoutRecv state, receives the gate evaluation result from Cell 9. Because Net 11 has two fanout branches, Cell 15 distributes the gate evaluation result to Cell 14 and itself. On the other hand, Cell 7' receives another gate evaluation result from Cell 7 and distributes it to Cell 27 as an input signal of the NAND gate at the next level.

In fault simulation, we drop the faults located at fanout nets during fault collapsing such that only fanin cells will be faulty. We mask the fault register of a specific cell which simulates the current fault to inject the fault, then apply $w/2$ patterns and simulate in parallel for both good and faulty circuits, where w is the word length of our CA. Finally, the results are compared in the primary output cells to determine whether the current fault under simulation is detected. In this manner, CA can perform pipelined logic and fault simulation.

III. Cell Architecture

The main operation of the CA cells is data transfer between two neighboring cells in both vertical and horizontal directions. There are two data paths in a column: one for downward transfer and the other for upward transfer. To judge whether a datum has reached its destination, we use an offset value to record the distance between the source and destination. The offset value accompanying the data being transferred is incremented or decremented according to its direction whenever it moves one position vertically. Therefore, each CA cell has a simple arithmetic unit (AU) to perform the arithmetic operation. Because there is no need for logic operation in the fanout column, fanout cells have no logic unit (LU). We show the cell structure in Fig. 2. The status transition information generator (STIG) of a cell receives the current status of its neighbors and generates all the status transition information needed by all transition control units (TCUs), which receive the next register values and information from STIG to determine their new values. STIG also generates the gate type of each fanin cell to determine the type of logic operation to be performed and controls the signals for AU to determine the offset of upward and downward data flows. The upward negative offset will be incremented, and the downward positive offset will be decremented. The new offset value will be loaded into register *UpOffReg* (*DnOffReg*) as its new value. Meanwhile, a signal from below (above) is loaded into register *UpSigReg* (*DnSigReg*). All the *DnOffReg* and *DnSigReg* registers of the cells in a column form a downward data path. Similarly,

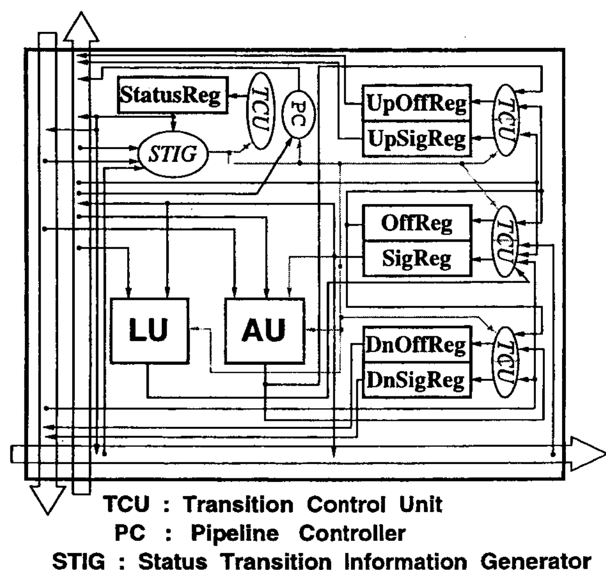

Fig. 2. The CA cell structure.

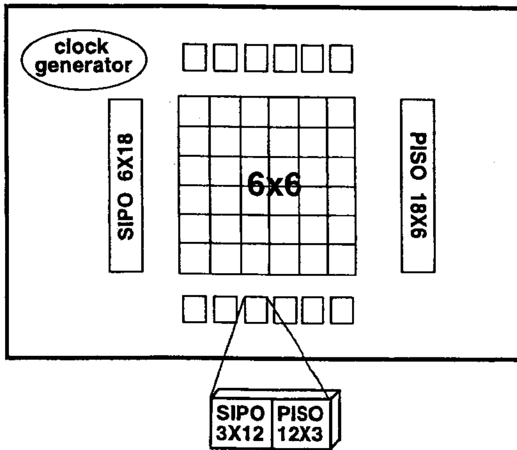

Fig. 3. The CA chip architecture.

all the *UpOffReg* and *UpSigReg* registers of the cells in a column form an upward data path. For each cell, the downward (upward) data path is free if there is no data to move downward (upward) in the cell above (below). Each cell can put the received signal from the left and the associated offset into the desirable data path only when that data path is free; i.e., the data already in the data path has a higher priority to flow into the registers of a cell than does the data in the cell.

OffReg keeps its content unchanged during simulation. *OffReg* stores the offest value by which each cell decides whether the signal received from the left needs to move toward its target cell and the direction of the move. If the offset value of *OffReg* is positive (negative), the offset as well as the signal received from the left is sent to *DnOffReg* (*UpOffReg*) and *DnSigReg* (*UpSigReg*), respectively, as the downward (upward) data path is free. If the value of *OffReg* is zero, it is the target cell of the signal from the left. If the value of *DnOffReg* (*UpOffReg*) of the upper (lower) cell is 1 (-1), STIG will generate certain information to indicate that this cell is the target cell of the signal in the upper (lower) cell, and the TCU of register *SigReg* will directly load the value of *DnSigReg* (*UpSigReg*) of the upper (lower) cell into *SigReg*. Once the *SigReg* of a fanin cell receives its own signal, the fanin cell will either immediately send its signal to LU to perform a logic operation in case the gate type of this cell is INVERTER or BUFFER or periodically check the status of its lower cell to wait for the other input signal if the logic operation of this cell requires two input signals.

In a fanout cell, the directions of all the signals passing through it are the same, which indicates that there is at most one signal passing through it at each time instance, so the fanout cell needs to perform at most one addition or subtraction at a time. On the other hand, a fanin cell has to perform one addition and one subtraction in a clock cycle in the worst case.

Our CA performs pipelined logic and fault simulation. We can consider each column of the 2-D CA architecture as a pipeline stage for the simulation process. The pipeline controller (PC) in Fig. 2 generates the status of itself and those cells below, and then sends the status to the upper cell. Each cell's PC accumulates the status in such a way that the top cell of a column will produce the information when all the cells of that column have finished their operations.

IV. Chip Architecture and Chip Implementation

We show the CA chip architecture in Fig. 3, in which there are 36 CA cells. The kernel is a 6x6 cellular array which consists of three fanin columns and three fanout columns. Due to the pin-count limitation, we use *serial-in-parallel-out* (SIPO) and *parallel-in-serial-out* (PISO) modules to process input and output signals, respectively. We have designed the CA chip so that it is extensible in both x and y dimensions; i.e., we can put together more CA chips to form a larger CA for larger circuits.

The chip design flow is shown in Fig. 4. The first phase includes circuit design, simulation, verification,

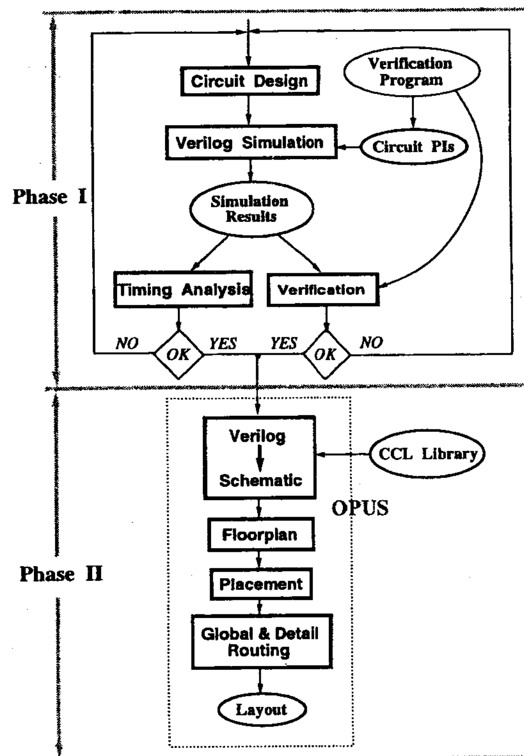

Fig. 4. The chip design flow.

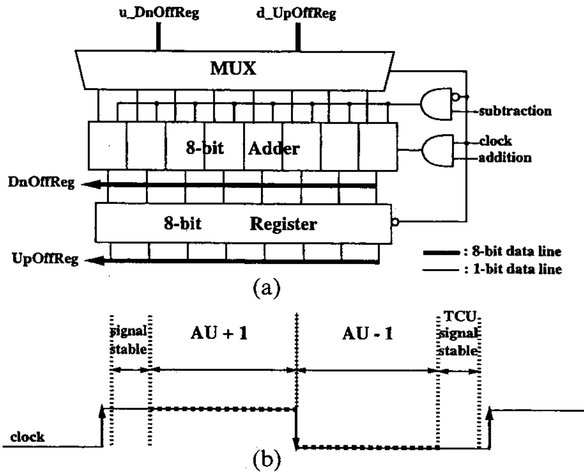

Fig. 5. Initial arithmetic unit design.

and timing analysis. In this phase, we use VERILOG to model the chip and VERILOG-XL to perform circuit simulation at the gate level. In Li and Wu (1995), we have implemented a CA software simulator for logic and fault simulation performed by CA. We use the CA simulator as the gold unit in the design process of all the primitive modules, CA cell, cellular array, and the whole CA chip, so design verification is greatly simplified. The verification program shown in Fig. 4 is obtained by slight modification of the CA simulator, which can generate necessary primary-input patterns at different hierarchical levels for VERILOG-XL simulation and the corresponding primary-output results for automatic comparison and verification. Simulation results contain numbers and waveforms of the input and output signals. Phase one is repeated from the primitive module design to the whole chip design. Phase two is the physical design process, which includes transforming the VERILOG chip design to schematic, floor-planning, placement, global routing, and detail routing. OPUS is used to accomplish the physical design.

1. Cellular Array

In Fig. 2, all the modules are combinational circuits except for the registers. STIG is the source of all control signals of all the TCUs. It provides the identification information of variable conditions, such as the existence of available signals on the upward and downward data paths, the receipt of a target signal, its status, the direction of the data path to forward a signal received from the left, etc. Each register has its own TCU; i.e., we design each TCU according to the corresponding register's state transition function. LU performs one of seven different logic operations in accordance with the gate type signaled by STIG. Since fanout cells do not need to perform logic operations, there is no LU in these cells. As mentioned before, fanin cells have to complete one addition and one subtraction in one clock cycle in the worst case while fanout cells have to perform one addition or subtraction. We show the initial AU design of fanin cells in Fig. 5, where the main body of AU is an 8-bit ripple-carry adder. The design principle is to perform addition during the positive pulse and to perform subtraction during the negative pulse in the same clock cycle. The binary data inside CA are in 2's complement form; consequently, the operation of subtraction by one is equivalent to that of addition by the 2's complement of 1. The output of AU is connected to the inputs of eight negative-edge-triggered flip-flops. If there one addition and one subtraction are to be performed, AU carries out addition during the positive pulse and loads the result into the eight flip-flops in the negative edge. After addition, AU carries out subtraction during the negative pulse, and then the results of addition and subtraction are loaded into *UpOffReg* and *DnOffReg*, respectively, in the positive edge of the next clock cycle. All the flip-flops of the cellular array except those in AU are positive-edge triggered. However, this design has one drawback; the clock cycle will be bounded by the computation time of addition and subtraction. If we want to raise the operating clock rate, we have to distribute the computation time of addition and subtraction over positive and negative pulses. Two 8-bit ripple-carry adders can be used. Experimental results show that the area of one 8-bit ripple-carry adder is approximately equal to that of eight flip-flops and one two-to-one multiplexer. Hence, using two 8-bit ripple-carry adders, the operating clock rate can be increased without increasing the cell area. Therefore,

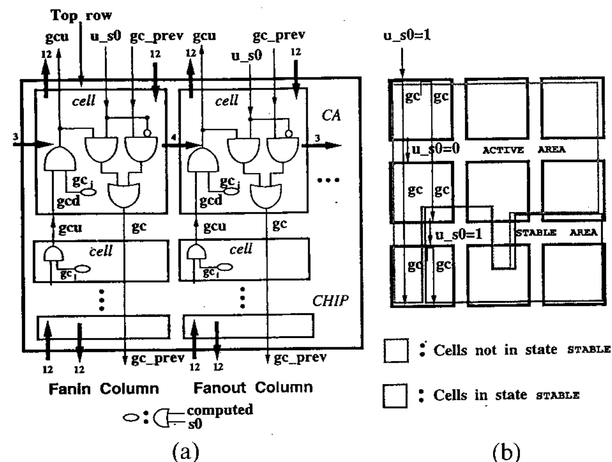

Fig. 6. The pipeline circuit. (a) Pipeline circuit in a chip. (b) Pipeline control signal in a chip array.

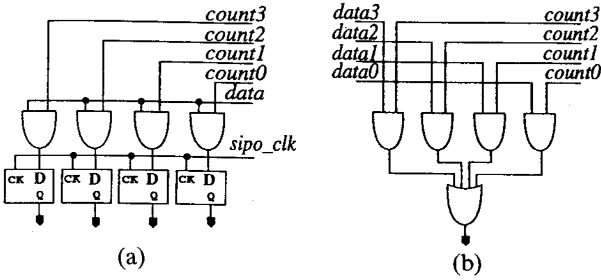

Fig. 7. (a) The 1×4 SIPO module. (b) The 4×1 PISO module.

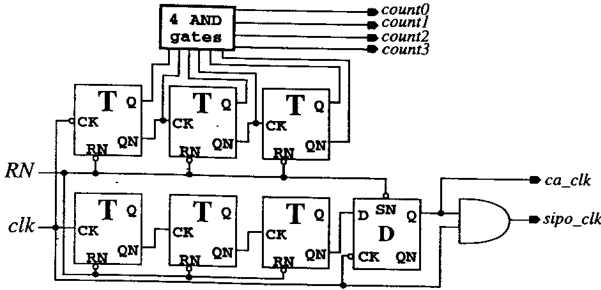

Fig. 8. The clock generator module.

two AUs are available in a fanin cell for concurrent addition and subtraction operations while there is only one in a fanout cell.

Pipelining is an important technique in our CA design, which is realized by adding an AND gate and an OR gate in each cell, as shown in Fig. 6(a). The signals *computed* and *s*0 are generated by STIG, where the former indicates whether this cell has completed its operation for the current data stream, and where the latter indicates whether this cell is in state STABLE. The signal gc_i is set to 1 when this cell has finished its work or this cell is in state STABLE, where it does nothing. The status of a column is accumulated by ANDing each cell's gc_i. The signal *gcd* is the status of those cells below itself, and the signal *gcu* is the status *gcd* plus its own status. The top cell's *gcu* is the status of this column. This special *gcu*, called *gc*, has to pass through all the cells of this column to inform them of the completion of all operations. For a multi-chip CA, the signal *gc* of a column can be either the top cell's *gcu* or the signal *gc* from its upper chip before we fix its location in a CA chip array. Figure 6(a) shows this selection, where *gc_prev* is the signal *gc* from its upper chip, and where signal *u_s*0 indicates whether its upper cell is in state STABLE. For the top cell of each column, if *u_s*0=1, this means that either this chip is the top chip of a column in the CA, or that the cells in this column are all in state STABLE during simulation. Under these circumstances, *u_s*0 selects *gcu* as the pipeline control signal; otherwise, *gc* comes from above, i.e., *gc_prev*. We show these conditions in Fig. 6(b). If all the cells of a column are in state STABLE, it does not matter what *u_s*0 is.

2. IO Modules and Clock Generator

In Fig. 6(a), we also show the width of signals which pass through the boundary of the 6×6 cellular array. There are 324 bits flowing to and from the 6×6 cellular array during each clock cycle. It is clear that several bits have to share the same pin if we use a 120-pin package. Twenty pins are reserved for power/ground and clocks. The input and output data stream for a top or bottom cell of a fanin or fanout column are both 12 bits. The following relations shows the pin count limitation and the clock cycles necessary for processing input and output signals:

$$4a \times 6 + 2b \leq 100; \quad (1)$$

$$\left\lceil \frac{12}{a} \right\rceil = \left\lceil \frac{18}{b} \right\rceil; \quad (2)$$

where a is the number of pins reserved for the 12-bit data stream at the top and bottom, and b is the number of pins reserved for the 18-bit data stream at the left and right. We obtain the result, a=3 and b=5, from the above relations. Twelve bits share three pins, and the processing time for input and output signals is 4 clock cycles. We construct a one-to-four SIPO circuit and a four-to-one PISO circuit as primitive modules to process input and output signals, respectively. A three-to-twelve (twelve-to-three) SIPO (PISO) module is composed of 3 one-to-four (four-to-one) SIPO (PISO) modules. We also combine 4 one-to-four (four-to-one) SIPO (PISO) modules to process 16 bits out of 18 horizontal input (output) signals and assign two more pins to the remaining two bits. Hence, a six-to-eighteen (eighteen-to-six) SIPO (PISO) module is composed of 4 one-to-four (four-to-one) SIPO (PISO) modules and two extra pins. Figure 7 shows the primitive SIPO and PISO circuits. *Count*0, *count1*, *count2*, and *count3* are the counting status of a negative-edge-triggered counter which counts from 0 to 7 periodically, as shown in Fig. 8. There is another simple SIPO design. A one-to-four SIPO module can be realized by connecting 4 D flip-flops one by one such that data can reach the target flip-flop after passing through several flip-flops. However, this design may cause an unexpected 1 to appear in the output of those flip-flops where the unexpected 1 has passed through. These unexpected 1s will cause some flip-flops to be unable to keep their contents unchanged. The design in Fig. 7(a) provides each data line with a stable value during the period of input signal scanning.

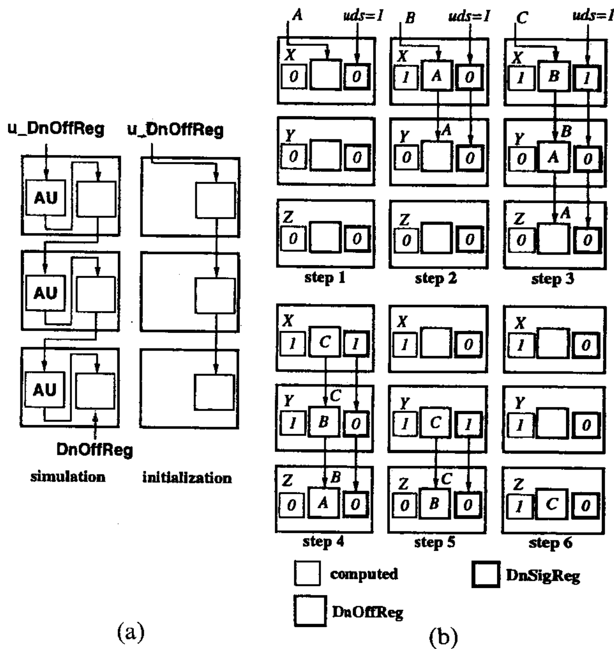

Fig. 9. (a) Configurable data path. (b) Coordination of *uds* and *computed*.

The 3-bit up counters of the clock generator module play an important role in the generation of various clocks (see Fig. 8). One CA chip requires three types of clocks: one for the clock generator module, another for the SIPO modules, and a third for the cellular array. The system clock is connected to the counter of the clock generator. The positive-edge-triggered counter, *counter 1* (the lower part of Fig. 8), produces clocks for the cellular array and the SIPO modules. By connecting the output, *QN*, of the most significant flip-flop of *counter 1* to a negative-edge-triggered D flip-flop, we obtain the cellular array clock, *ca_clk*, whose clock cycle is eight times the system clock cycle. The negative-edge-triggered D flip-flop delays the cellular array clock for half a system clock cycle, which ensures generation of a complete clock cycle for the cellular array at the beginning of the system clock. By ANDing the system clock and the output *Q* of the negative-edge-triggered D flip-flop, we produce a new clock to be used by SIPO modules, *sipo_clk*. This clock behaves in the same as does the system clock with a slight delay when *counter 1* counts from 1 to 4, and is idle in the other states of *counter 1*. The SIPO and PISO modules process input and output signals when *counter 2* (the upper part of Fig. 8) counts from 0 to 3, and the cellular array performs simulation from the counting period of 4 to 7. The relationship among the various clocks and the counting status is as follows. In the positive edge of *ca_clk*, each flip-flop in the cellular array loads its new value and changes into a new state. Immediately after that, *count0* is activated to select the part of the data to send out and read in, and then the first positive pulse of *sipo_clk* occurs to load the data selected by *count0*. This operation of counting status and clock *sipo_clk* is repeated three times to send out and read in all the output and input signals. These operations occur during the positive pulse period of clock *ca_clk*. Before the rise of the next positive edge of clock *ca_clk*, CA cells perform simulation, and each signal reaches a stable value.

3. Initialization Circuit

By means of the predefined transition rules and circuit characteristics, CA can perform logic and fault simulation. The CA chip has two operating modes: an initialization mode and a simulation mode. Circuit characteristics are loaded into each cell in the initialization mode. Registers, which need to be initialized, include *FanoutNo* (2 bits), *state* (4 bits), *OffReg* (8 bits), and *Gtype* (3 bits), i.e., a total of 17 bits. We use *DnOffReg* as the data register during initialization. Three passes are needed to initialize all the necessary registers for a fanin cell.

In the simulation mode, *DnOffReg* is connected to the AU input of the cell below, and the AU output is then connected to the input of *DnOffReg*. However, *DnOffReg* registers in the same column link directly to form a data path in the initialization mode, as shown in Fig. 9(a). Initialization data flows downward. In order to judge the data availability and initialization status of a cell, we modify the TCU circuit for two one-bit registers, *computed* and *DnSigReg*. All cells' *computed* registers are set to 0 before proceeding to a new pass. When a cell has initialized the associated register(s) in this pass, it sets *computed* to 1. The initialization data can only be used once even though it has to pass through all the cells of a column. *DnSigReg*, which follows initialization data, i.e., *DnOffReg*, tells the cell below if it can load the initialization data coming from above into associated register(s). The condition that *uds*=1 represents that the upper initialization data has been used, where *uds* is the *DnSigReg* of the cell above. Therefore, one cell can load the value of *DnOffReg* of its cell above into an associated register(s) only when *uds*=1 and *computed*=0. Figure 9(b) shows the coordination of *uds* and *computed*. Initially, *uds* of the top cell *X* is equal to 1, so cell *X* loads data *A* into a related register(s), sets *computed* to 1, and sets its *DnSigReg* to 0 to tell the cells below it that data *A* has been used. In step 2, owing to the detection of a zero value of *uds*, cell *Y* just lets data *A* pass by and preserves a value of zero in *DnSigReg*. Although the *uds* of cell *X* is 1, cell *X* also lets data

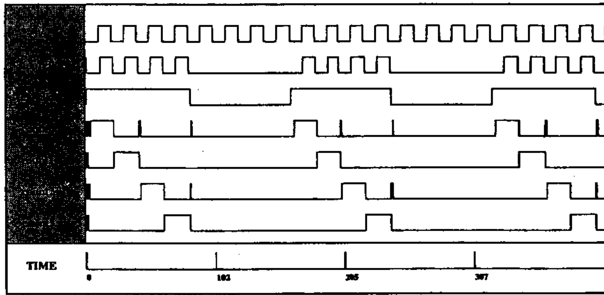

Fig. 10. VERILOG simulation result of the clock generator.

B pass by and preserves a value of one in *DnSigReg* because its *computed* is 1. In step 3, cell *X* lets the available data *C* pass by and preserves its availability; cell *Y* loads data *B* into corresponding register(s), sets *computed* to 1, and clears the availability of data *B*; cell *Z* lets the used data *A* pass by. In general, *n* cells require 2*n* clock cycles to complete the initialization process for a specific register(s) of all the cells in a column.

Besides the 17-bit data, we need two more bits to mark a specific register(s) to be initialized at each pass. Two bits from *UpOffReg*, *uo*1 and *uo*0, act as these two control bits. Initialization passes can be classified into three types in accordance with the distribution of initialization data over passes:

Type A: *uo*1, *uo*0=0,0. *DnOffReg* consists of *uo*1, *uo*0, *FanoutNo*, and *state*.
Type B: *uo*1, *uo*0=0,1. *DnOffReg* contains the value of *OffReg*.
Type C: *uo*1, *uo*0=1,0. *DnOffReg* contains the value of *Gtype*.

It seems that the initialization process for fanin columns requires only three passes from the above initialization data arrangement. As a matter of fact, four passes are required to complete the initialization process: two type-A passes, one type-B pass, and one type-C pass. In addition to initializing *FanoutNo* and *state*, a type-A pass is also used as the preparation pass for the type-B and type-C passes. For example, a type-A pass sets *uo*1 and *uo*0 to be 0 and 1, respectively, before a type-B pass proceeds. Two type-A passes mean that *uo*1 and *uo*0 have to be reset twice. The first reset operation can be achieved by the reset signal *RN*. The second reset operation is induced by a global signal *IRN*. An existing signal *u_s*2 which shows whether the upper cell is in state BotFanin is available. We rename the top cell's *u_s*2 of each fanin column as *IRN* and broadcast it to all the cells of this column so as to control the second reset operation. The new reset signal of *uo*1 and *uo*0 is the product of *RN* and the output of a NAND gate whose two inputs are *IRN* and the inverse of *start*, where *start* is a control signal to start the simulation. Two initialization control bits will be reset either when *RN*=0 or when *u_s*2=1 and *start*=0. Register *computed* also needs to be cleared before any type of pass occurs. We use each column's *gc* to achieve the reset operation. Fanout cells only have to initialize *OffReg* and *state*; therefore, there is no type-C pass for fanout cell initialization. The data of the *FanoutNo* field in type-A pass can be arbitrary. Fanout cells only require two passes to initialize their registers. Reset signal *IRN* is not needed for fanout cells. The reset control signal of *uo*1 and *uo*0 is *RN*. Because fanin cells and fanout cells must enter the simulation mode simultaneously, the initialization process of fanout columns is the same as that of fanin columns. Fanout columns repeat the first two passes when fanin columns proceed to the last two passes.

V. Simulation Results

We use VERILOG-XL to perform gate-level circuit simulation in the first phase of the chip design process. Simulation for all the primitive modules, the CA cell, and the whole chip completed successfully. The simulation results show that the clock cycle was 12 *us* (about 83 MHz), and that the operating frequency of the cellular array was approximately 83/8=10 MHz. Figure 10 shows the VERILOG simulation results of the clock generator module. Clock *clk* was the system clock and was connected to the clock generator module, *sipo_clk* was the clock for the registers of the SIPO modules, and *ca_clk* was the clock for the registers of the cellular array. As stated before, the internal registers of the SIPO modules and cellular array were positive-edge triggered. *Count*0, *count*1, *count*2, and *count*3 were the control signals used to select data to

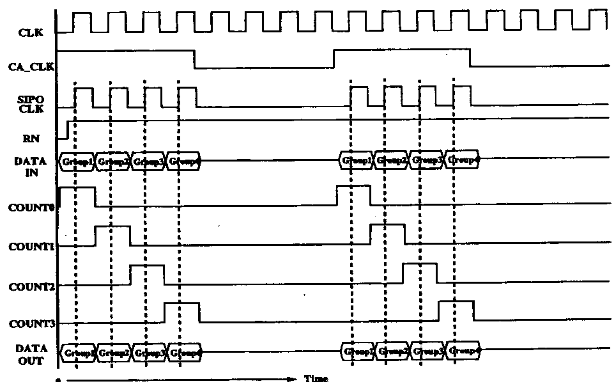

Fig. 11. A timing diagram of our chip.

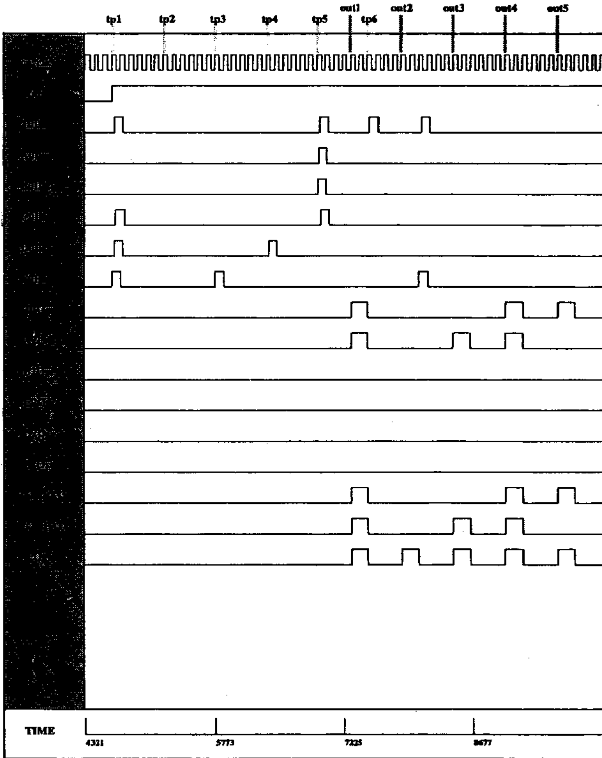

Fig. 12. VERILOG simulation results of the CA chip.

be sent out and scanned in for PISO and SIPO modules. Because the output signals from the PISO modules of one chip were the input signals of its neighboring chips' SIPO modules, *count*0 had to be activated to keep the first group of data ready before the first positive edge of *sipo_clk* rose, and the same relationship existed between *count*1 and the second positive rising edge of *sipo_clk*, etc. There were hazards in *count*0 and *count*2; however, these hazards did not affect the circuit behavior since the periods of these hazards were not located in the positive rising edge of *sipo_clk*. Moreover, the PISO modules were used to output the current registers' state of the cellular array; therefore, the first group of output data from the PISO modules were produced after the registers of the cellular array had changed to a new state. In fact, the positive rising edge of *ca_clk* could occur either before or after *count*0 was activated, but this had to happen before the first positive rising edge of *sipo_clk* so that the input signals for the SIPO modules were ready. In a previous section, we adopted two AUs for fanin cells to reduce the clock cycle of CA. There is another simple way to increase the operating frequency. We can activate the critical path signals, such as *OffReg*, before the four PISO and SIPO operations.

A timing diagram of our chip is shown in Fig. 11. The control signal *RN*, which is used to reset all registers to 0, must return to 1 before the first rising edge of external clock such that the I/O control circuits of our chip can work correctly. Moreover, input data should be ready before the rising edge of *SIPO_CLK*, and the simulation output of each group will be controlled by *count*0, *count*1, *count*2, and *count*3. The timing of the input and output data should be identical because the output data of the current chip will be the input data of its neighboring chips. Figure 12 shows the chip simulation result when c17 was loaded. The signals displayed from top to bottom are for the CA clock, initialization, T, A, B, C, D, E, and output signals. *T0I1ss* is the *lss* signal of the fanin cell in the first row and first level, where *lss* is the stable signal from the left, which stores the simulation result. The signals from *T0I1ss* to *E0I1ss* are primary inputs of c17. The corresponding sequence from *T0I1ss* to *E0I1ss* to the primary input of c17 in Fig. 1 is 1, 3, 3, 6, 2, and 7. *T2Oss* is the stable signal of the fanout cell in the first row and the third level. Circuit c17 has two primary outputs, i.e., 22 and 23, which correspond to *T2Oss* and *A2Oss*, respectively. *TOssPAD* is the output of *T2Oss*'s pad, whose value has to be equal to that of *T2Oss*. *POgc* is the pipeline control signal of the primary-output column. When *start* becomes 1, we apply one input pattern every six CA clock cycles. Figure 12 shows that the CA chip produced one simulation result in every six CA clock cycles after the first simulation result was generated. This simulation performance is the same as that estimated by our CA simulator in Li and Wu (1995).

The CA chip contains 10,138 primitive gates, 432

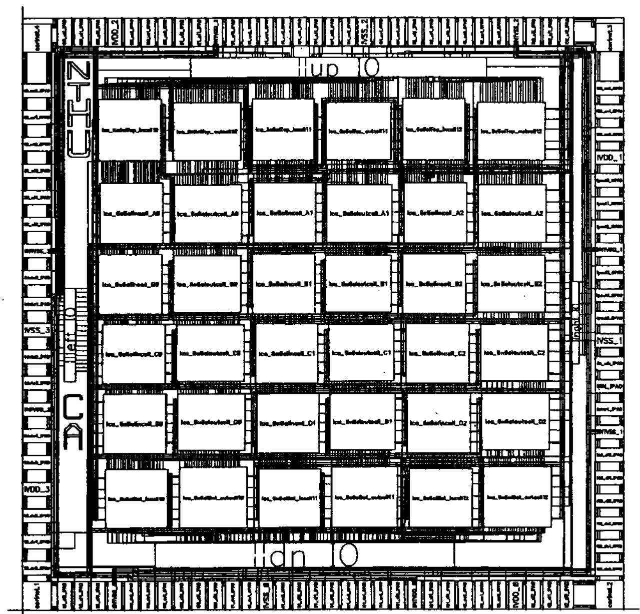

Fig. 13. CA chip layout.

full adders and 1,573 flip-flops, where the cellular array consists of 9,450 gates, 432 full adders, and 1,404 flip-flops. For other modules, there are 688 gates and 169 flip-flops. According to estimation bassed on the library data-book (one full adder contains 9 gates and one flip-flop contains 8 gates), the CA chip is composed of 26,610 gates. Figure 13 shows the chip layout. In the layout, the clock generator module has been merged into the left IO module.

VI. Conclusion

In our previous work (Li and Wu, 1995), we showed that a 2-D CA architecture with proper circuit mapping is suitable for logic and fault simulation, with better simulation performance compared to that of previously published works. In this paper, we have described the VLSI architecture and circuit design of the CA simulation engine. The CA architecture is scalable in both dimensions. It produces one output in every six CA clock cycles after the pipeline has been filled in the initial simulation stage, and outperforms previously reported parallel simulators by three orders of magnitude.

Acknowledgment

This work was supported in part by the National Science Council, R.O.C., under contracts NSC 83-0404-E007-063 and NSC 84-2215-E007-027.

References

Agrawal, P. (1986) Concurrency and communication in hardware simulators. *IEEE Trans. on Computer-Aided Design*, **5**, 617-623.

Agrawal, P., V. Agrawal, K. T. Cheng, and R. Tutundjian (1989) Fault simulation in a pipelined multiprocessor system. *Proc. International Test Conference*, pp. 727-734. Washington, D.C., U.S.A.

Agrawal, V. D. and S. T. Chakradhar (1992) Performance analysis of synchronized iterative algorithms on multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, **3**, 739-746.

Antreich, K. J. and M. H. Schulz (1987) Accelerated fault simulation and fault grading in combinational circuits. *IEEE Transactions on Computer-Aided Design*, **6**, 704-712.

Armstrong, D. B. (1972) A deductive method for simulating faults in logic circuits. *IEEE Transactions on Computers*, **21**, 464-471.

Bailey, M. L. (1992) How circuit size affects parallelism. *IEEE Transactions on Computer-Aided Design*, **11**, 208-215.

Becker, B., R. Hahn, and R. Krieger (1992) Fast fault simulation in combinational circuits: an efficient data structure, dynamic dominators and refined check-up. *Proc. European Design Automation Conference*, pp. 436-441. Paris, France.

Gloria, A. D. and P. Faraboschi (1992) Massive parallelism in multi-level simulation of VLSI circuits. *INTEGRATION, the VLSI journal*, 145-171.

Harel, D. and B. Krishnamurthy (1987) Is there hope for linear time fault simulation. *Proc. 17th Int. Symp. Fault-Tolerant Computing*, pp. 28-33. Pittsburgh, PA, U.S.A.

Huisman, L. M., I. Nair, and R. Daoud (1990) Fault simulation of logic designs on parallel processors with distributed memory. *Proc. IEEE International Test Conference*, pp. 690-696. Washington, D.C., U.S.A.

Ishiura, N. and S. Yajima (1992) Linear time fault simulation algorithm using a content addressable memory. *Proc. European Design Automation Conference*, pp. 442-445. Paris, France.

Ishiura, N., H. Yasuura, and S. Yajima (1987) High-speed logic simulation on vector processors. *IEEE Transactions on Computer-Aided Design*, **6**, 305-321.

Kitamura, Y. (1986) Hardware engines for logic simulation. In: Logic Design and Simulation, pp. 165-192. *Advances in CAD for VLSI*, Vol. 2. E. Horbst Ed. North-Holland, Amsterdam, Netherlands.

Li. Y. L. and C. W. Wu (1995) Cellular automata for efficient parallel logic and fault simulation. *IEEE Transactions on Computer-Aided Design*, **14**, 740-749.

Maamari, F. and J. Rajski (1988) A fault simulation method based on stem regions. *Proc. Int. Conf. Computer-Aided Design*, pp. 170-173. Santa Clara, CA, U.S.A.

Narayanan, V. and V. Pitchumani (1992) Fault simulation on massively parallel simd machines: algorithms, implementations and results. *Journal of Electronic Testing: Theory and Applications*, **3**, 79-92.

架構於細胞式自動機之邏輯與偵錯模擬硬體模擬器之超大型積體電路設計

李毅郎　賴盈照　吳誠文

國立清華大學電機工程學系

摘　要

隨著超大型積體電路技術不斷的進步與硬體價格的降低，特殊用途機器由於具有平行處理的能力，越來越廣泛地被用在加速偵錯模擬上。在我們以前提出的二維細胞式自動機(CA)模型中，CA被當作單一指令流多重資料流(SIMD)的平行架構來加速邏輯與偵錯模擬。

在本論文中，我們提出一細胞式自動機晶片來實體化先前我們所提出的CA模型，此CA晶片具有快速平行處理邏輯與偵錯模擬的能力。由於使用導管式技術(pipeline)，我們的模擬機器在機器字組長度爲8位元與時鐘頻率爲20MHz的情況下，可以達到每秒模擬十億個邏輯閘的模擬速度。CA架構可以很簡單的往四個方向（上、下、左、右）擴充，也就是每個CA晶片能夠直接地與它四個相鄰的CA晶片互相溝通連接而形成一個較大的細胞式陣列。CA晶片有兩種工作模式：初使化模式與模擬模式；在模擬工作模式下，CA晶片能執行邏輯與偵錯模擬。當導管被填滿之後，每隔6個時鐘週期即會產生一組結果。和先前已發表的平行偵錯模擬器比較，我們的結果大約快了1000倍。