VLSI Design, 2002 VOL. 15 (3), pp. 629–635

Taylor & Francis
Taylor & Francis Group

Improving the Timing of Extended Finite State Machines Via Catalyst

SHI-YU HUANG

Electrical Engineering Department, National Tsing-Hua University, HsinChu, Taiwan, ROC

(Received 15 March 2001; Revised 30 January 2002)

We propose a timing optimization technique for a complex finite state machine that consists of not only random logic but also data operators. In such a design, the timing critical path often forms a cycle and thus cannot be cut down easily by popular techniques such as pipelining or retiming. The proposed technique, based on the concept of *catalyst*, adds a functionally redundant block—which includes a piece of combinational logic and several other registers—to the circuits under consideration so that the timing critical paths are divided into stages. During this transformation, the circuit's functionality is not affected, while the speed is improved significantly. This technique has been successfully applied to an industrial application—a Built-In Self-Test (BIST) circuit for static random access memories (SRAMs). The synthesis result indicates a 47% clock cycle time reduction.

Keywords: Finite state machine; Retiming; Catalyst; Timing optimization; Synthesis

INTRODUCTION

The logic part of a synchronous digital design is often viewed as the interconnection of a number of data-path components and control blocks. The data-path components include arithmetic circuits as well as other data-related components such as counters, shift-registers, stacks, queues, etc. On the other hand, the control block is often regarded as a finite state machine [11]. Since there are strong two-way interactions between the data and control components, plenty of efforts have been devoted to stitching together these two types of components as a single abstract model, e.g. the extended finite state machine (EFSM) model [1]. Through these models, designers will be able to capture complicated design intentions in a compact graphical form.

In many control-dominant circuits, the above hybrid model is necessary in order to characterize the two-way interactions between the data and the control. Figure 1 shows an example. A control's outputs regulate a counter's operation (e.g. reset, freeze, or increment), while the counter's value also feeds back to the control to decide which transition to take in the next clock cycle. It is very likely that the timing critical paths of this design pass through the random logic of the control as well as the counter.

In addition to the logic-domain and physical-domain timing optimization techniques provided in various design tools, significant circuit speed up can also be achieved through the restructuring of the RTL code by applying the architecture-level techniques such as pipelining and retiming [5,6,8,9]. However, these techniques, although effective for a data-path dominant component, tend to be less useful in a control-dominant one.

In Refs. [2–4], a new type of timing-optimization technique called "architectural retiming" was proposed. Through the concepts of "negative register" and "pre-computation", the micro-operations of the design are rescheduled so that the timing critical paths can be pipelined to reduce the clock latency. Significant improvement was reported on data-dominant designs such as greatest-common-divisor (GCD) generator.

In this paper, we propose an automatic procedure for optimizing a control-dominant circuit expressed as an extended finite state machine. This technique is analogy of adding catalyst to a chemical reaction system. The resulting outcome is not changed but the process is thereby accelerated. In a given design modeled as a complex finite state machine, the timing critical path is likely to be a cycle passing through the combinational portion of the control as well as the data-path. By catalyst, a circuit including a piece of combinational logic, referred to as *prediction logic*, and a number of registers is added to the design, so that in the resulting design, the original cyclic path is divided into segments. In order to maintain the circuit's functionality during the transformation, the prediction logic, which is related to the state transition graph of the given finite state machine, needs to be

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/1065514021000012246

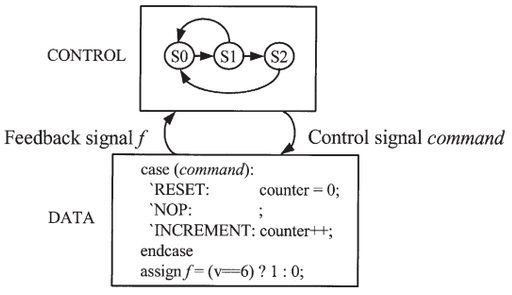

FIGURE 1 Two-way interactions between the data and the control.

constructed by a specific procedure that will be described in "Timing optimization via catalyst" Section.

To some extent, our technique is similar to the architectural retiming techniques [2] in terms of concepts. But there are two differentiating factors: (1) Our technique is based on a formal design model—extended finite state machine, while the work in Ref. [2] is based on a more design-specific micro-architecture. (2) The timing-optimization flow presented in this work is more general and systematic than the one in Ref. [2] and thus can be easily automated by a high-level synthesis tool.

We have applied this technique to a Built-In Self-Test circuit that produces a pre-defined test sequence for on-chip memory testing [7]. The speed of this design is an important design criterion because the speed of the BIST circuit needs to be as fast as the memory under test in order to provide the at-speed memory read/write patterns. Otherwise, certain timing defects such as those that prevent a cell from rising to high within a specified time period may not be captured. Our design uses a 0.35 μm CMOS standard cell library. The post-layout static timing analysis indicates that the added catalyst reduces the clock cycle time from 4.55 to 3.08 ns, achieving a 47% timing improvement at the cost of 23% area increase.

The rest of this paper is organized as follows. The Second section reviews the basics of extended finite state machine model. Third section details the proposed timing optimization technique and the construction procedure of the catalyst circuit. Also, we give a general high-speed architecture for realizing complex finite state machines. Fourth section discusses our design experience using the proposed technique. Fifth section gives the conclusion.

PRELIMINARIES

Extended Finite State Machine Model

An extended finite state machine [1] differs from the traditional finite state machine in their definitions of the transitions. In a conventional FSM, the transition is associated with a set of input Boolean conditions and a set of output Boolean operations. In an EFSM model, the transition can be expressed by an "*if statement*" consisting of a *trigger condition* and a set of *data operations*. When

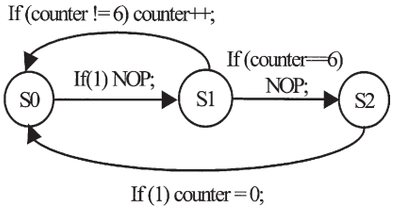

FIGURE 2 An example of extended finite state machine.

the trigger condition is satisfied, the transition is fired, bringing the machine from the current state to the next state and performing the specified data operations. Figure 2 shows an example. The transition from state $S1$ to $S0$ is denoted as: $T(S1 \rightarrow S0)$: If (counter! = 6) counter++.

In this transition, the trigger condition is "counter! = 6", while the associated operation is "counter++". Therefore, it is only executed when the current state is $S1$ and the data variable *counter* is not 6. When executed, it brings the machine to state $S0$, and increments the counter value by 1.

In general, the trigger conditions and the data operations may depend on the primary inputs as well as the data variables. However, for simplicity without the loss of generality, we assume in the sequel that trigger conditions and data operations only depend on the data variables.

A Primitive Architecture

Given a circuit modeled as an extended FSM, we first partition this model into a more structural diagram consisting of three blocks as shown in Fig. 3:

FSM-block: A conventional finite state machine that realizes the state transition graphs of the EFSM model.

A-block: An arithmetic block for performing the data operation (e.g. the increment of a counter) associated with each transition. The operation of this block is regulated by the output signal of the FSM-block.

E-block: A block for evaluating the trigger conditions associated with each transition (e.g. evaluating if a counter's value equals a threshold number). The input

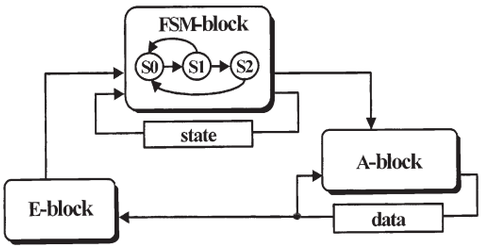

FIGURE 3 A primitive architecture for realizing EFSMs.

signals of this block are the data variables, while the output is a set of binary signals taken as the inputs by the FSM-block.

Consider the example in Fig. 2. The corresponding FSM-block is simply the random logic that realize finite state machine. In this example, the A-block is responsible for performing three kinds of operations on the data variable *counter*, reset, freeze, and increment. The E-block is mainly for realizing the trigger conditions of transitions. The outputs of the E-block serve as the inputs to the FSM-block for deciding the state transition. In this example, the E-block is a simple integer comparator that checks if the data variable *counter*'*s* value is six.

The timing critical paths can be revealed by examining the most complicated transition in the EFSM model, which is the aforementioned $T(S1 \rightarrow S0)$. This transition involves the computation of a trigger condition and a counter's increment in addition to its source and destination states. As a result, the implementation of this transition involves the logic in every one of the three blocks. Since this transition needs to be executed in a single clock cycle if triggered, its corresponding timing path could start from the output of the data variable *counter*, passing through the E-block, FSM-block, and A-block and finally reaches the input of the variable *counter*. Let the delays of the three blocks along the critical path denoted as D(FSM-block), D(A-block), and D(E-block). Then the clock cycle time is dominated by (D(E-block) + D(FSM-block) + D(A-block)).

TIMING OPTIMIZATION VIA CATALYST

In this section, we use a three-step transformation to illustrate how catalyst can be added to improve the speed. Also, we discuss the procedure of deriving the catalyst circuitry.

Step 1: Retiming Move Across A-block

Based on the primitive architecture, the first step we apply is a retiming move [9] that moves the data variable from the outputs of the A-block to its inputs, leading to a block diagram shown in Fig. 4. One the two new registers is named *command* in order to reflect its meaning of being the control signal from the FSM-block, while the other is name *pre-data* because it holds the *data value in the previous clock cycle*. Note that the input signal to the E-block is not affected by this retiming move. It still represents the *current data*'*s value*. The new critical path now starts from the output of the variable *command*, passing through the A-block, E-block, FSM-block, and then finally reaches the same variable's input. The delay remains to be (D(E-block) + D(FSM-block) + D(A-block)). Therefore, this retiming move does not reduce the clock cycle time. However, it creates a platform on which the catalyst can be added.

Step 2: Adding Catalyst

To the above-retimed architecture, we add a piece of logic (referred to as prediction logic) and two registers Q1 and Q2 between the A- and E-block. The result is a structure shown in Fig. 5. It can be seen that the timing critical path has now been divided into three segments:

Segment 1: The path across A-block. The delay is D(A-block).

Segment 2: The path across the catalyst circuitry. The delay is D(catalyst).

Segment 3: The path across E-block and then FSM-block. The delay is (D(E-block) + D(FSM-block))

The clock cycle time is now dominated by the largest delay of these three segments. It is trivial that the delays of the segment 1 and 3 are both less than the delay of the original timing path, which is (D(E-block) + D(FSM-block) + D(A-block)). The only possibility that the addition of the catalyst results in a slower design is when D(catalyst) is greater than (D(E-block) + D(FSM-block) + D(A-block)).

Deriving Prediction Logic

The above transformation is only valid if the added catalyst circuitry does not change the circuit's behavior. In the original architecture, the input to the E-block is the data signal. Hence, the catalyst circuitry should also

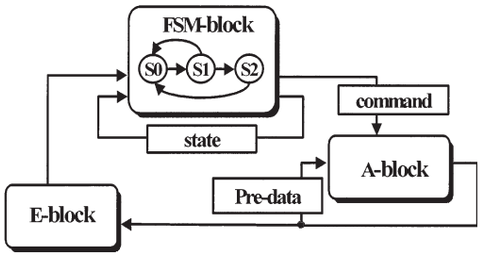

FIGURE 4 Architecture after the first retiming move.

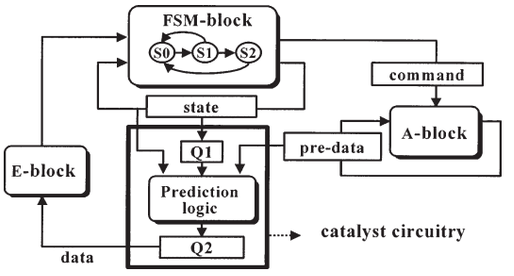

FIGURE 5 Architecture after inserting the catalyst circuitry.

produce the same signal at the output of the register Q2 in order to maintain the circuit's functionality. A close examination of the prediction logic reveals the requirements to achieve this objective. We use the following notations for the input/output signals of the prediction logic:

P-state: The output signal of register Q1, being the state signal delayed by one clock cycle. This signal represents the state signal in the previous clock cycle.

C-state: The output signal of the state register, representing the current state.

Pre-data: The output signal of A-block's register pre-data, being the data signal delayed by one clock cycle. The signal represents the data signal in the previous clock cycle.

In order to maintain the circuit's functionality, the prediction logic should produce the data signal of the next clock cycle, denoted as *N*-data, so that this signal become current data signal after being delayed by the register Q2. In some sense, the prediction logic can be viewed as a combinational sub-circuit that *looks ahead* for two clock cycle in the given design's state transition graph in order to predict next clock cycle's data signal (*N*-data) based on the information of only current state (*C*-state), previous state (*P*-state), and previous data (*P*-data). We first define a term called two-cycle transition before proving that there always exists such prediction logic for any given extended finite state machine.

DEFINITION 1 *Two-cycle transition*: A two-cycle transition, T^2, as shown in Fig. 6, is a set of three states $\{S1, S2, S3\}$ and two transitions $\{T1, T2\}$ so that $T1$ and $T2$ can happen sequentially, bringing the state machine from state $S1$ to state $S2$ and then to state $S3$ in two consecutive clock cycles. Note that the transition $T1$ and $T2$ can be characterized by their trigger conditions (denoted as t1 and t2) and associated operations (denoted as op1, op2).

DEFINITION 2 *Middle-state degree*: A state's middle-state degree is the number of two-cycle transitions that pass the state as the second state. It is equal to the product of the number of fanins and the number of fanouts of the particular state under consideration.

OBSERVATION 1 The total number of two-cycle transitions of a given state transition graph is simply the summation of each state's middle-state degree.

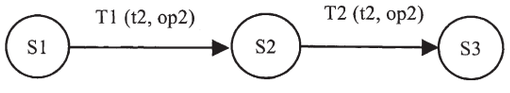

FIGURE 6 A two-cycle transition taking *S*2 as the middle state.

OBSERVATION 2 A current state (*C*-state), a previous cycle's state (*P*-state), and a previous cycle's data (*pre-data*) uniquely define a two-cycle transition. For simplicity without the loss of generality, we implicitly assume that the primary input signals do not involve in any transition's trigger condition or output operations.

Proof First of all, the transition from *P*-state to *C*-state is uniquely defined by the value of *pre-data*, denoted as *T*1(t1, op1). Otherwise, there are multiple transitions that may be triggered at the same time, violating the basic assumption that the given state transition graph is sound at any time. Secondly, when the machine is in *C*-state, the data value, *C-data*, is equal to op1(*P-data*). This data value in turn uniquely defines the next transition to take, denoted as *T*2(t2, op2), and the next state, denoted as *N*-state. Therefore, we have derived the three state sequence, {*P*-state, *C*-state, *N*-state}, and the two transitions, $\{T1, T2\}$, that form a two-cycle transition. ■

PROPOSITION 1 The input signals of the prediction logic, i.e. {*P*-state, *C*-state, *pre-data*} uniquely defines the data value of the next clock cycle, namely, *N*-data. In other words, the prediction logic exists for any given extended finite state machine.

Proof Based on the above observation, current cycle data value, *C*-data, and the second transition, *T*2(t2, op2), are both well defined. Therefore, the next clock cycle data value, *N*-data, is simply op2(*C*-data), where *C*-data is actually op1(*P*-data). Therefore, we conclude *N*-data is uniquely defined as op2(op1(*P*-data)). ■

OBSERVATION 3 Although the input signals of the prediction logic incorporated are sufficient to predict the next clock cycle's data value, they are not all necessary. It can be shown that the set of {*P*-state, *P*-data} is already enough to predict the next clock cycle data value. However, this alternative may lead to a more complicated implementation of the prediction logic. By including the current state *C*-state as an input signal, the hardware overhead of the prediction logic is thereby reduced. The prediction logic can be constructed by a nested if -then-else statements in which each two-cycle transition corresponds to an "*if*" or "*else-if*" statement. However, nested if-then-else statements as shown in Fig. 7. A close

```
begin
 case(P-state):
  `S0: case(C-state):
     `S1: if(pre-data==6)   N-data = pre-data;
        else          N-data = pre-data + 1;
  `S1: case(C-state):
     `S0: if((pre-data !=6) && 1)  N-data = pre-data + 1;
     `S2: if(1 && (pre-data==6))  N-data = pre-data;
                        `S2: case(C-state):
     `S0: if(1 && 1)   N-data = 0;
 endcase
end
```

FIGURE 7 The RTL code for an example EFSM.

examination of the above logic expression reveals that the implementation requires incriminators, comparators, and multiplexers.

Step 3: Retiming Move Again

The addition of the catalyst circuitry breaks down the originally cyclic timing path into stages. In our experience, the critical path now lies in the prediction logic. In order to further reduce the clock cycle time, we perform another retiming move that re-position the register Q2 from the output of the prediction logic to the output of the E-block. The resulting circuit is shown in Fig. 8. Retiming transformations are often performed in a way that the critical component could be divided. However, in our transformation, we further add the E-block to the critical component—prediction logic. Let the cascade of the prediction logic and the E-block be referred to as *predicted E-block*. The following derivation of the predicted E-block will show the motivation of this retiming move.

Notice that the E-block is for evaluating every trigger condition in the EFSM. In the example of Fig. 2, there is only one trigger condition, which is checking if (data = 6). After the above retiming move, the predicted E-block is responsible for creating the trigger signal of the next clock cycle, denoted as $N-e$, which is "1" when (N-data = 6), and "0" otherwise. This signal becomes the trigger signal feeding the FSM-block after it is delayed by the register Q2. The Verilog-like code for the predicted E-block thus can be expressed as below in Fig. 9. Unlike the implementation of the prediction logic, the predicted E-block does not require any incrementer. It only requires multiplexers and comparators to implement. Since the speed of a comparator is much faster that an incrementer, we have the following important observation that motivates the above retiming move.

OBSERVATION 4 If the data operations involved in the two-way interactions between the data component and the control component are all constant addition/subtraction, then the predicted E-block (i.e. the prediction logic plus the E-block) could be much faster than the pure prediction logic alone.

```
Begin
 case(P-state):
  `S0: case(C-state):
     `S1: if(pre-data==6)   N-e = (pre-data==6);
        else          N-e = (pre-data==5);
  `S1: case(C-state):
     `S0: if((pre-data !=6) && 1)  N-e = (pre-data==5);
     `S2: if(1 && (pre-data==6))  N-e = (pre-data==6);
  `S2: case(C-state):
     `S0: if(1 && 1)   N-e = 0;
 endcase
end
```

FIGURE 9 The RTL code for the predicted E-block.

A High-speed Architecture

Given an EFSM, a potential high-speed architecture can be constructed as the one indicated in Fig. 10, which consists of three major blocks—FSM-block, retimed A-block, and predicted E-block. In the primitive architecture, the clock cycle time, denoted as $\tau_{\text{cycle-primitive}}$, is roughly equal to the sum of the delays across the FSM-block, the A-block, and the E-block. In the new architecture, the clock cycle time, denoted as $\tau_{\text{cycle-catalyst}}$, is determined by the maximum delay of the three blocks. Therefore, the new architecture is faster than the primitive one unless the delay of the predicted E-block is larger than the sum of the delays of FSM-block, A-block, and E-block. In the next section, we demonstrate the effectiveness of this technique by an industrial design.

APPLICATION

The proposed technique has been successfully applied to a small industrial design for memory built-in self-test.

Built-in Self-test for Memories

Built-in self-test (BIST) is a design-for-test technique in which the test pattern generation and test application are both accomplished by one-chip hardware. For embedded memory modules, this technique is more indispensable due to the lack of accessibility to the input/output pins of the circuit under test. Furthermore, built-in self-test allows at-speed testing of high-performance designs with

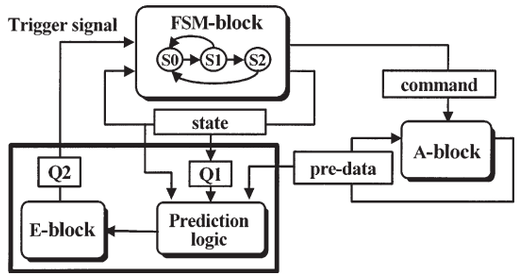

FIGURE 8 The structure after moving Q2 across the E-block.

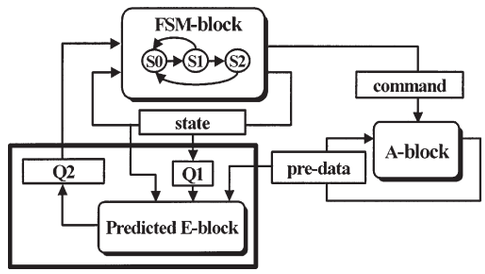

FIGURE 10 A higher-speed architecture for an EFSM.

low-end inexpensive testers, contributing to testing cost reduction.

The BIST circuit for testing a memory often consists of two major components: (1) test pattern generator and (2) response analyzer. Here the test pattern generator is a piece of hardware that is responsible for generating the pre-defined test patterns, each of which is composed of three parts: (1) command (i.e. read or write), (2) cell address, and (3) read/write data. The response analyzer is to determine if the memory outputs match with the expected responses during the testing. The main focus of memory testing is often on detecting the manufacturing defects associated with the memory cells, e.g. deciding if there exists a defect that causes a cell to be stuck-at "0", or fail to rise to high-level voltage within a specified time window. Based upon a number of fault models, the required test patterns to discover these faults can thereby defined.

The abstraction descriptions of these test patterns are often called test algorithms. Numerous algorithms have been proposed. In this BIST design, we implemented five most popular ones: (1) scan algorithm, (2) checkerboard algorithm, (3) horizontal march algorithm, (4) vertical march algorithm, and (5) multiple address selection algorithms. The details of the other algorithms are referred to the literature [10].

A primitive design reveals that the speed of a memory BIST circuit is limited by the pattern generator, which is concisely modeled as a 42-state extended finite state machine (part of this EFSM is shown in Fig. 11). Associated with the transition of this model are the trigger conditions and data operations dependent on two data variables These two multiple-bit data variables serves as the *row address* and *column address* of the memory cell to be accessed. The number of bits of these two data variables depends on the configuration of the memory. For example, if the address space is 1024×64, then the number of bits for row address and column address would be 10 and 6. The A-block realizing the data operations receives nine different kinds of operation commands from the FSM-block: NO-CHANGE, INIT-ADDRESS, LAST-ADDRESS, NEXT-CELL, PREVIOUS-ELL, Y-NEXT-CELL, Y-PREVISOU-CELL, CROSS-CELL, and RECALL. Each of these commands will force the A-block to reset, increment, complement, or just refresh the two data registers. The trigger signals generated by the E-block are simply deciding if any of the data variables equals zero or a threshold number. The example transition T_1 of the partial EFSM in Fig. 11 is expressed as:

$$\text{If}(\text{row!} = \text{ROW} - 1 \| \text{col!} = \text{COL} - 1)\,\text{then}\,\text{NEXT}-\text{CELL}.$$

It means that when the data variable *row* is not equal to the last row number, *ROW*-1, or the column address is not equal to the last column number, *COL-1*, then this transition is taken and a command, *NEXT-CELL*, is issued to the A-block, which updates the two address variables so that they point to the next cell to be accessed.

In this design, we found that the timing critical path forms a cycle in the primitive architecture. After the addition of the catalyst circuitry and the merge of the E-block with the prediction logic through retiming, the critical path lies in the A-block.

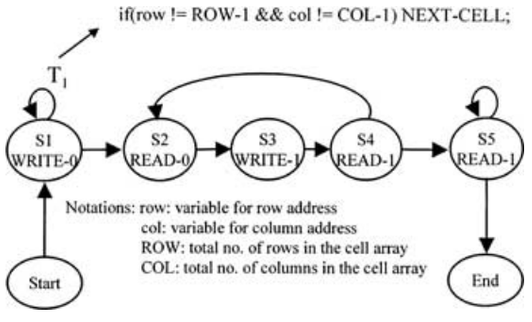

FIGURE 11 Partial EFSM model for the pattern generator.

Synthesis Results

The described memory BIST circuit has been implemented as a synthesizable RTL code using the proposed catalyst-based architecture. We simulate it along with the functional model of the memory device under test to establish the confidence of its functional correctness. Synthesis tool, *Design compile*, is used to convert this RTL code into a netlist and perform logic optimization using 0.355 μm CMOS technology library. We use the timing-driven feature of a place-and-route tool, *Apollo*, to generate the layout. Post-layout timing analysis using *Design Time* shows that the clock cycle time is 3.08 ns. On the other hand, the clock cycle time of the primitive version *without* the catalyst circuitry is 4.55 ns *after being optimized* by *Design Compiler*. Also, the primitive version has 2455 equivalent gates, while the high-speed version has 3040 equivalent gates. The area increase is about 23%. Based on these results, we thus conclude that the proposed timing optimization technique achieves a 47% speedup as applied to the memory BIST circuit. This timing improvement makes possible the at-speed testing for high-speed embedded memories that may run at fast as 300 MHz.

CONCLUSION

Existing architecture-level techniques for timing optimization such as pipelining and retiming, although highly effective in many data-path designs, often fail as applied to a complex control circuit where the random logic of the finite state machine and certain counter-typed operators jointly define the timing critical paths that form a cycle. Inspired from the design process of a real-life application—memory BIST—we propose a timing

optimization technique based on the concept of catalyst. This technique, mimicking the process of adding catalyst to a chemical reaction system, breaks down the cyclic timing paths into separate pieces by adding a functionally redundant block. We showed that the catalyst circuitry exists for any design given in extended finite state machine model. Furthermore, we summarize this technique as a general high-speed architecture for realizing a control-dominant circuit. The post-layout timing analysis of our memory BIST circuit indicates a 47% timing improvement, at the cost of 23% area increase.

References

[1] Cheng, K.-T. and Krishnakumar, A.S. (1993) "Automatic Functional Test Generation using the Extended Finite State Machine Mode", *Proc. 30th ACE/IEEE Design Automation Conf.*, 86–91.

[2] Hassoun, S. and Ebeling, C. (1996) "Architectural retiming: pipelining latency-constrained circuits", *IEEE Design Automation Conf.*, 708–713.

[3] Hassoun, S. and Ebeling, C. (1998) "Using Pre-computation in architecture and logic resynthesis", *Int. Conf. Computer-Aided Design*, 316–323.

[4] Hassoun, S. (1998) "Fine grain incremental rescheduling via architectural retiming", *Int. Sympos. System Synthesis*, 158–163.

[5] Leiserson, C.E. and Saxe, J.B. (1991) "Retiming synchronous circuits", *Algorithmica* **6**, 5–35.

[6] Malik, S., Sentovich, E.M., Brayton, R.K. and Sangiovanni-Vincentelli, A. (1991) "Retiming and resynthesis: opimizing sequential networks with combinational techniques", *IEEE Trans. Computer-Aided Designs* **10**(1), 74–84.

[7] Saluja, K.K., *et al.*, (1987) "Built-In Self-Test RAM: a practical alternative", *IEEE Design Test Computer*, 42–51.

[8] Shenoy, N. and Rudell, R. (1994) "Efficient implementation of retiming", *Proc. Int. Conf. Computer-Aided Design*, 226–233.

[9] Singhal, V., Pixley, C., Rudell, R. and Brayton, R.K. (1995) "The validity of retiming sequential circuits", *Proc. 32th Design Automation Conf.*, 316–321.

[10] Suk, S. and Reddy, S.M. (1981) "A march test for functional faults in semi-conductor random-access memories", *IEEE Trans. Computers* **C-30**(12), 982–985.

[11] Villa, T., Kam, T., Brayton, R.K. and Sangiovanni-Vincentelli, A. (1997) Synthesis of Finite State Machines: Logic Optimization (Kluwer Academic Publishers, Dordrecht).

Shi-Yu Huang received his B.S. and M.S. degrees in Electrical Engineering from National Taiwan University in 1988 and 1992 and Ph.D. degree in Electrical and Computer Engineering from the University of California, Santa Barbara in 1997, respectively. From 1997 to 1998 he was a software engineer at National Semiconductor Corporation, Santa Clara, investigating the system-on-a-chip design methodology. From 1998 to 1999, he was with Worldwide Semiconductor Manufacturing Corporation, designing the high-speed Built-in Self-Test circuits for memories. He joined the faculty of National Tsing-Hua University, Taiwan, ROC in 1999, where he is currently an Associate Professor. His research interests are in the area of computed-aided design for VLSI, with an emphasis on design verification. He co-authored a book entitled "Formal Equivalence Checking and Design Debugging" Published by Kluwer Academic Publishers in 1998.