Research Article

An Approach for Implementing State Machines with Online Testability

P. K. Lala,¹ A. Mathews,² and J. P. Parkerson²

¹Department of Electrical Engineering, Texas A&M University, Texarkana, TX 75505, USA

²Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA

Correspondence should be addressed to P. K. Lala, parag.lala@tamut.edu

Received 3 June 2009; Revised 24 December 2009; Accepted 9 February 2010

Academic Editor: Rubin Parekhji

Copyright © 2010 P. K. Lala et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

During the last two decades, significant amount of research has been performed to simplify the detection of transient or soft errors in VLSI-based digital systems. This paper proposes an approach for implementing state machines that uses 2-hot code for state encoding. State machines designed using this approach allow online detection of soft errors in registers and output logic. The 2-hot code considerably reduces the number of required flip-flops and leads to relatively straightforward implementation of next state and output logic. A new way of designing output logic for online fault detection has also been presented.

1. Introduction

It is now widely accepted that transient errors are the major contributor of digital system failure [1–6]. Several techniques have been proposed over the years for designing self-checking combinational circuits and state machines to cope with transient errors [7–14]. An architecture for designing online/offline testable state machines was proposed in [11] that has an advantage over the traditional approaches to self-checking design. This method uses a 1-hot code to encode states, making the next state logic and output logic design straightforward. The disadvantage of the 1-hot code is that it requires n flip-flops to encode n states, which considerably increases the size of the circuit.

This paper proposes a technique for designing self-checking Mealy-type state machines whose registers and output logic are testable online. In a Mealy-type state machine the output is a function of the present state and current inputs [15]. The testable state machine uses a 2-hot code for state encoding instead of a 1-hot code used in [6]. The advantage of the 2-hot code over the 1-hot code is that it considerably reduces the number of flip-flops needed, while the next state and output logic are still relatively straightforward. Although it is possible to use an m -out-of- n code to encode the states of a state machine, an encoded state will have more 1's than the 2-hot code in cases where a circuit

has more than ten states. As a result the next state logic will require more circuitry, thereby increasing the unreliability of the circuitry. A new method for designing the output logic is also proposed in this paper. This proposed design generates the output and the complement of the output using only transistors and buffers; this approach makes the resulting circuitry more hardware efficient compared to using multiple input logic gates and incurs less overhead than a conventional approach, for example, duplicate and compare [1].

2. Architecture for Online Testable State Machine

A top view of the architecture that can be utilized to implement on-line testable state machines is shown in Figure 1; this is a modified version of that presented in [11]. The Output Logic takes in the present state and inputs, and generates the primary output F and its complement F' simultaneously. F and its complement can be compared externally to verify that they are the complement of each other. If they are not the complement of each other, there is a fault in the Output Logic.

As stated earlier in the proposed approach of state machine design, a 2-hot code is used to represent the states.

FIGURE 1: Top view of the proposed architecture.

FIGURE 2: Register Q implementation.

FIGURE 3: Example of Next State Logic.

A 2-hot code requires more flip-flops but simplifies the next state and output logic. Register R represents the present state, and Register Q , which feeds back into Register R , represents the next state. Register R and Register Q have scan out features that allow them to be tested offline. The output of the next state logic is loaded into register Q . The outputs of Registers R and Q are XORed and then compared by the Even Parity Checker. This determines if there is a transient error in the Next State Logic (details are discussed in the next

section). The content of register Q can also be scanned out to check for permanent fault(s) in the next state logic; this can be done offline.

3. Next State Logic Design

Using a 2-hot code, there are three possible relations between the present state and the next state. If two states have no 1's in common, they differ by four bits:

State 1: 110000

State 2: 001010

If two states have one 1 in common, they differ by two bits:

State 1: 011000

State 2: 010100

If two states have two 1's in common, they must be the same state.

Therefore, when using a 2-hot code, any two given states must have an even number of differing bits. In Figure 1, the XOR Module gives the number of differing bits between the present and next state, and the Even Parity Checker determines whether or not the XOR has an even number of 1's. If there is an even number of 1's in the XOR, then there should not be a single fault in either register nor in the Next State Logic. If there is a single fault in the Next State Logic or in one of the registers, the result may be a faulty 1 or a faulty 0 in the present state or next state. A faulty 1 or a faulty 0 in one of the registers would change the parity of the XOR function from even to odd. If there is an odd number of 1's in the XOR, then there is a fault in either the Next State Logic or in one of the registers, and it can be observed in the output "Next State Check."

FIGURE 4: Example of encoding scheme and realization using transistors.

FIGURE 5: Example of the path given by input combinations.

Register Q 's values are set by the Next State Logic, using reset/preset on the flip-flops. The implementation of Register Q is shown in Figure 2. All the flip-flops are reset to 0 on the rising edge of the system clock. After some delay, the Next State Logic sets the appropriate two flip-flops to 1 using preset (denoted by P in Figure 2). The Next State Logic should provide enough delay to ensure that the flip-flops have all been reset before the appropriate present state flip-flops are preset.

The Next State Logic may be implemented with only p-type transistors, as seen in Figure 3. One possible present state is when the R_1 and R_4 outputs of Register R are 1. In the example in Figure 3, there are two inputs (X_0 and X_1). If

$X_0 = 0$ and $X_1 = 0$, and both R_1 and R_4 are asserted, then a 1 is propagated to the outputs P_0 and P_1 , which denotes some particular state. This circuit may be repeated for all present states ($\{R_1, R_2\}$, $\{R_1, R_3\}$, etc.).

4. Output Logic Design

The Output Logic must take in the present state and the inputs, and must generate the output and the bitwise inverse of the output, as seen in Figure 1. The goal here is to minimize the overhead required to generate F and F' . Figure 4 shows the encoding scheme for four possible numbers of states and the transistor design that selects

FIGURE 6: Implementation of circuit to generate F_1 and F_1' .

FIGURE 7: Buffer designs.

the appropriate state. A shared transistor is used for states that have a 1 in common. Since only one state can be active at a time, only one path can be active at a time, and this path gives the present state.

A similar approach, as shown in Figure 5, is used to design the portion of the circuit that gives the correct combination of inputs that are active.

In Figure 5, every possible input combination points to the corresponding path that is activated. Note that with three inputs there are eight possible input combinations and eight possible paths to be activated, in which one and only one path must be active at any given time. When Figures 4 and 5 are put together, the result gives us all possible combinations of present state and input combinations. Since only one path can be active in Figure 4 and only one path can be active in Figure 5, one and only one path will be active at any time when the two are put together. However, a difficulty arises in trying to connect the present state transistors with the input transistors (or connect Figures 4 and 5). For example, if state 1 is directly connected to inputs 1 and 2, and state 2 is directly connected to inputs 2 and 3 then state 1 will be connected to input 3, thus giving an incorrect value for F when state 1 and input 3 are activated. This can be avoided by using buffers to control flow of signals when multiple present state lines

fan-in to a single input line. This technique will be illustrated using the following example.

Table 1 shows a transition table for a state machine and the resulting output equations. This state machine has six states—using all possible 2-hot combinations of the four state bits, two inputs (X_1 and X_0), and two outputs (F_1 and F_0).

The output logic circuit for generating F_1 and F_1' from the example in Table 1 is shown in Figure 6; the inverters necessary to generate the complements of the inputs are not shown. This design combines the techniques shown in Figures 4 and 5 to give a path for any possible present state/input combinations that make $F_1 = 1$. For example, the transition table in Table 1 shows if $Q_3 \cdot Q_2 \cdot X_0 \cdot X_1 = 1$, then $F_1 = 1$. It can be seen in Figure 6, if $Q_3 \cdot Q_2 \cdot X_0 \cdot X_1 = 1$, then a 1 is propagated to F_1 . At the same time, a 0 is propagated to not (F_1) in the bottom circuit through an equivalent path.

Notice that the right circuit is a replica of the left circuit, except (1) the p-type transistors are replaced by n-type transistors (2) all the input and state signals are inverted, and (3) the output has a pull-up mechanism instead of a pull-down mechanism. Because the circuits are alike, if a path is activated in the left circuit, then the same path must be

FIGURE 8: Sharing of transistors in the Output Logic.

activated in the right circuit. Since the left circuit propagates a 1 to the output or otherwise is pulled to 0, and the right circuit propagates a 0 to the output or otherwise is pulled to 1, the outputs will always be the complement of each other in the absence of a fault.

The buffers control the direction of signal flow in the circuit. They also act as amplifiers between multiple transistors connected in series. Since there is a resistance associated with all transistors, it is not desirable to have more than three transistors connected directly in series [16]. The more transistors in series the slower the circuit will be, although the transistors could be made larger than minimum to reduce the delay. In addition fewer transistors in series result in less resistance between the output and the power rails. A general rule of thumb as proposed in [17] is “No more than 4 n-channel or 3 p-channel devices in series in any gate.”

It should be noted that the number of buffers in Figure 6 may be reduced by configuring them in a different form. The buffers used in the pull-up circuit (Figure 6(a)) and the buffers used in the pull-down circuit (Figure 6(b)) are different. Figure 7 shows an implementation of these buffers.

The proposed technique has been applied to design a subset of MCNC of circuits selected as suggested in [18]. The results are listed in Table 2. The first column lists the number of transistors in the next state logic (NSL) of the benchmark circuits; similarly the second, third, and the fourth columns show the number of transistors required in the output logic

(OL), XOR module (XOR), and parity checker (PAR) in each circuit. The total number of transistors is shown in the fifth column. The number of flip-flops includes Registers, those in register R , Register Q , and the Next State Checking logic of Figure 1.

It should be indicated that although different outputs in a multioutput circuit can share logic, sharing of logic between an output (F) and its complement (F') is not allowed in the proposed scheme so that a fault does not affect both output circuits at the same time thus preventing its detection. If there is a fault in one of these shared p-type transistors, the fault will be propagated to F_0 and F_1 , but not to F'_0 and F'_1 , therefore the fault will be detected. Similarly, a fault that appears in a shared n-type transistor will be propagated only to F'_0 and F'_1 but not to F_0 or F_1 and therefore will also be detected.

The transistor counts in the OL column in Table 2 were derived assuming no sharing of logic across different outputs. Thus the transistor counts may be reduced by possible sharing of logic. For example in Figure 5, output circuits F_0 and F_1 can share the p-type transistors that identify the present state by transmitting a 1. Similarly in Figure 6, the circuits that generate (F'_0) and (F'_1) can share the n-type transistors that identify the present state by transmitting a 0. However sharing could increase fan-outs in the circuits thereby requiring more buffers. Figure 8 shows a general layout of shared of transistors in a testable state machine designed using the proposed scheme.

TABLE 1: Example FSM transition table and resulting output equations.

Present state				Inputs		Next state				Outputs	
R_3	R_2	R_1	R_0	X_1	X_0	Q_3	Q_2	Q_1	Q_0	F_1	F_0
1	1	0	0	0	0	1	1	0	0	0	1
				0	1	1	0	1	0	0	
				1	0	0	0	1	1	0	
				1	1	0	1	0	1	1	
1	0	1	0	0	0	0	1	1	0	1	
				0	1	0	1	0	1	0	
				1	0	0	0	1	1	1	
				1	1	1	0	0	1	0	
1	0	0	1	0	0	1	1	0	0	1	
				0	1	1	1	0	0	0	
				1	0	1	0	0	1	0	
				1	1	1	0	0	1	0	
0	1	1	0	0	0	0	1	0	1	0	
				0	1	0	1	1	0	1	
				1	0	0	1	1	0	1	
				1	1	1	1	0	0	1	
0	1	0	1	0	0	1	0	0	1	0	
				0	1	0	0	1	1	0	
				1	0	0	1	0	1	1	
				1	1	1	0	0	1	1	
0	0	1	1	0	0	1	1	0	0	1	
				0	1	1	0	1	0	0	
				1	0	0	1	1	0	0	
				1	1	0	1	0	1	0	
$F_0 = R_3R_2(X_1 + \overline{X_1}\overline{X_0}) + R_3R_1(\overline{X_0}) + R_3R_0(\overline{X_1}\overline{X_0}) + R_2R_1(\overline{X_1} + X_1\overline{X_0}) + R_2R_0(X_1) + R_1R_0(X_1\overline{X_0} + \overline{X_1}X_0)$											
$F_1 = R_3R_2(X_1X_0) + R_3R_1(\overline{X_1} + X_1\overline{X_0}) + R_3R_0(X_1) + R_2R_1(X_1 + \overline{X_1}X_0) + R_2R_0(X_1 + \overline{X_1}\overline{X_0}) + R_1R_0(\overline{X_1}\overline{X_0})$											

TABLE 2: Number of transistors and flip-flops needed for implementing MCNC benchmark circuits.

	NSL	OL	XOR	PAR	TOTAL	Flip-Flops
cse	225	314	56	48	643	15
ex1	248	762	56	48	1114	15
dk16	280	286	64	56	686	17
dk17	192	240	40	32	504	11
donfile	240	0	64	56	360	17

5. Conclusion

This paper has proposed a method for designing state machines with online testability. The XOR Module and Parity Checker allow for online testing of Registers R and Q , and the Next State Logic. This design will detect any single stuck-at fault in the Next State Logic or in the Registers. The proposed technique uses a design approach that is different from that used for designing conventional CMOS-based design; it uses transistors and buffers rather than fully complementary transistors.

Acknowledgment

The first author was supported in part by the National Science Foundation USA under Grant 0925080.

References

- [1] P. K. Lala, *Self-Checking and Fault Tolerant Digital Design*, Morgan-Kaufmann, Aberdeen, Scotland, 2001.
- [2] C. Zhao and S. Dey, "Evaluating and improving transient error tolerance of CMOS digital VLSI circuits," in *Proceedings of*

- the International Test Conference*, pp. 1–10, Santa Clara, Calif, USA, 2006.
- [3] S. Mitra, N. Kee, and S. Kim, “Robust system design with built-in soft-error resilience,” *IEEE Design and Test Computers*, vol. 38, no. 2, pp. 43–52, 2005.
 - [4] M. Omana, G. Papasso, D. Rossi, and C. Metra, “A model for transient fault propagation in combinatorial logic,” in *Proceedings of the 9th IEEE On-Line Testing Symposium (IOLTS ’03)*, pp. 111–115, July 2003.
 - [5] K. Mohanram and N. A. Touba, “Cost-effective approach for reducing soft error failure rate in logic circuits,” in *Proceedings of the International Test Conference*, pp. 893–901, 2003.
 - [6] F. Wang and V. D. Agrawal, “Soft error rate determination for nanometer CMOS VLSI logic,” in *Proceedings of the 40th Annual Southeastern Symposium on System Theory*, pp. 324–328, New Orleans, La, USA, 2008.
 - [7] M. K. Stojčev, G. Lj. Djordjevič, and T. R. Stanković, “Implementation of self-checking two-level combinational logic on FPGA and CPLD circuits,” *Microelectronics Reliability*, vol. 44, no. 1, pp. 173–178, 2004.
 - [8] G. L. Djordjevič, M. K. Stojčev, and T. R. Stanković, “Approach to partially self-checking combinational circuits design,” *Microelectronics Journal*, vol. 35, no. 12, pp. 945–952, 2004.
 - [9] A. Matrosova and S. Ostanin, “Self-checking FSM design with observing only FSM outputs,” in *Proceedings of the 6th IEEE On-Line Testing Symposium (IOLTS ’00)*, pp. 153–154, Palma de Mallorca, Spain, 2000.
 - [10] R. A. Parekhji, G. Venkatesh, and S. D. Sherlekar, “A methodology for designing optimal self-checking sequential circuits,” in *Proceedings of the International Test Conference*, pp. 283–291, Nashville, Tenn, USA, October 1991.
 - [11] P. K. Lala and A. Walker, “A unified scheme for designing testable state machines,” in *Proceedings of the IEEE Asia Test Symposium*, pp. 273–278, Kyoto, Japan, 2001.
 - [12] K. Mohanram, E. S. Sogomonyan, M. Gossel, and N. A. Touba, “Synthesis of low-cost parity-based partially self-checking circuits,” in *Proceedings of the 9th IEEE On-Line Testing Symposium (IOLTS ’03)*, pp. 35–40, July 2003.
 - [13] I. Levin, V. Ostrovsky, S. Ostanin, and M. Karpovsky, “Self-checking sequential circuits with self-healing ability,” in *Proceedings of the 12th ACM Great Lake Symposium on VLSI*, New York, NY, USA, April 2002.
 - [14] B. Abramov, V. Ostrovsky, and I. Levin, “Designing self-checking circuits with smooth power dissipation,” in *Proceedings of the 24th IEEE Convention of Electrical and Electronics Engineers*, pp. 1–5, Eilat, Israel, 2006.
 - [15] P. K. Lala, *Principles of Modern Digital Design*, John Wiley & Sons, New York, NY, USA, 2008.
 - [16] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, Pearson Education, Upper Saddle River, NJ, USA, 2003.
 - [17] M. Hoffman and J. K. Kim, “Delay optimization of combinational static CMOS logic,” in *Proceedings of the 24th ACM Design Automation Conference*, pp. 125–131, 1987.
 - [18] S. Yang, “Logic synthesis and optimization benchmarks, user guide version 3.0,” in *Proceedings of the International Workshop on Logic Synthesis*, MCNC, Research Triangle Park, NC, USA, May 1991.