DESIGN DIVERSITY FOR CONCURRENT ERROR DETECTION IN SEQUENTIAL LOGIC CIRCUITS

Subhasish Mitra and Edward J. McCluskey
Center for Reliable Computing
Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, California
http://crc.stanford.edu

Abstract

We present a technique using diverse duplication to implement concurrent error detection (CED) in sequential logic circuits. We examine three different approaches for this purpose: (1) Identical state encoding of the two sequential logic implementations, duplication of flip-flops, diverse implementation of the combinational logic part (output logic and next-state logic) and comparators on flip-flop outputs and primary outputs; (2) Diverse state encoding of the two implementations, duplication of flip-flops, diverse combinational logic implementation and comparators on primary outputs only; and (3) Identical state encoding, parity prediction for the flip-flops, diverse combinational logic implementation, comparators on primary outputs and parity checkers on flip-flop outputs. Our results for the simulated sequential benchmark circuits demonstrate that the third approach is most efficient in protecting sequential logic circuits against multiple and common-mode failures. The computational complexity of the data integrity analysis of the third approach is of the same order as that of the first approach and is at least an order of magnitude less than that of the second approach.

1. Introduction

Concurrent Error Detection (CED) techniques are widely used for designing systems with high data integrity. By data integrity, we mean that the system either produces correct outputs or generates an error signal when incorrect outputs are produced. A duplex system in the form of a self-checking pair is a classical example of a CED scheme which has been used for guaranteeing data integrity in many applications like the IBM G5 and G6 processors [Spainhower 99]. Figure 1.1 shows the basic principle of operation of a duplex system. As long as only one module fails, a duplex system provides guaranteed data integrity.

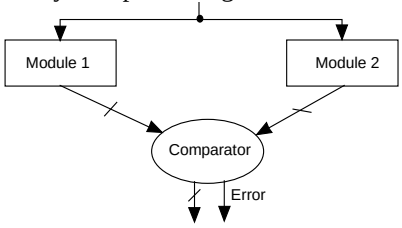

Figure 1.1. A duplex redundant system

It is generally assumed that module failures are independent events; hence, in a duplex system, the probability that both modules fail is very low for realistic failure rates. However, this assumption is not always true. In a duplex system, *common-mode failures* (CMFs) result from failures that affect both modules at the same time, generally due to a common cause [Lala 94]. These include operational failures due to external (such as EMI, power-supply disturbances, radiation) or internal causes and design mistakes. CMFs are surveyed in [Mitra 00a].

Design diversity was proposed and used in the past to protect redundant systems against common-mode failures [Avizienis 84, Briere 93, Riter 95]. In [Avizienis 84], *design diversity* was defined as the independent generation of two or more software or hardware elements (e.g., program modules, VLSI circuit masks, etc.) to satisfy a given requirement. The basic idea is that, with different implementations, common failure modes will cause different error effects.

The conventional notion of diversity is qualitative and does not provide any quantitative insight into design of diverse duplex systems. In [Mitra 99a], a metric was developed to quantify design diversity and analyze the reliability, availability and data integrity of duplex systems using this metric. In [Mitra 00b], this metric was used as a cost function to synthesize diverse implementations of combinational logic functions. However, the efforts on characterization of diverse duplex systems were focused on combinational logic circuits. In this paper, we extend our ideas to sequential logic circuits.

This work was done as part of the ROAR (Reliability Obtained by Adaptive Reconfiguration) project [Saxena 00]. In the project, the system under consideration is reconfigurable and contains user-programmable logic elements (e.g., FPGAs). For such systems, faults can be detected during system operation, the faulty part can be located, and the system can be reconfigured to operate without using the defective part. The Field Replaceable Unit (FRU) is a programmable logic block or a routing resource, instead of a chip or a board used in any conventional fault-tolerant system. Hence, it is reasonable to design combinational or sequential logic with concurrent error detection such as duplication.

In Sec. 2, we describe three approaches to designing sequential logic circuits with CED based on diverse

duplication and present simulation results comparing these three schemes. Section 3 describes a technique to analyze the data integrity of sequential logic circuits with CED. We conclude in Sec. 4.

2. Diverse Duplication for Sequential Logic Circuits

We consider the Finite State Machine (FSM) model of sequential circuits [McCluskey 86] as shown in Fig. 2.1. In addition, we assume that faults do not affect the clock signal (not shown in Fig. 2.1) in the FSM implementations. While our technique can be extended for faults on clock signal lines, this assumption is reasonable when fault-tolerant clocks [Siewiorek 92] are used.

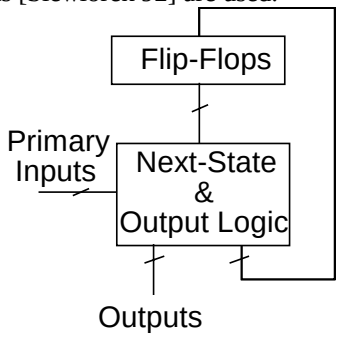

Figure 2.1. FSM model of a sequential circuit

Various techniques have been proposed in the past to implement concurrent error detection in sequential circuits. These include techniques based on parity prediction, Berger and Bose-Lin codes. [Zeng 99] presents a comprehensive description of these previously reported CED techniques for sequential logic circuits. Results presented in [Mitra 00c] demonstrate that, for general combinational logic circuits, CED techniques based on diverse duplication provide better protection against multiple failures and CMFs compared to simple duplication and parity prediction; moreover, the area overhead of diverse duplication is comparable to (or marginally more than) that of parity prediction. Hence, in this paper we study CED techniques based on diverse duplication for sequential logic circuits.

2.1. Identical State Encoding and Diverse Logic (ISEDL)

In Fig. 2.2 both implementations have identical encoding of the FSM internal states; however, we have diverse implementations of the next-state and the output logic. The primary outputs and the state-bits (flip-flop outputs) of the two implementations are compared and an error is indicated when a mismatch occurs.

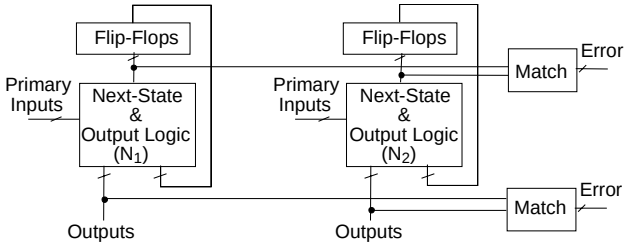

Figure 2.2. Identical state encoding and diverse logic (ISEDL)

For synthesizing diverse implementations of the next-state and the output logic the technique in [Mitra 00b] can be used. This CED technique suffers from the problem that there is no diversity in the state encoding (i.e., the flip-flop contents). In the worst-case, for a fault f affecting a flip-flop in the first implementation, a fault g affecting the corresponding flip-flop in the second implementation can be identified, such that the fault pair (f, g) can never be detected by the comparator; this situation is not desirable.

2.2. Diverse State Encoding and Diverse Logic (DSEDL)

Diversity can be created by encoding the internal states of the given FSM in "different" ways in the two implementations. This provides another degree of freedom in the synthesis of FSMs with CED based on diverse duplication and can possibly help in providing enhanced protection against CMFs compared to the scheme in Fig. 2.2. This scheme is shown in Figure 2.3. Since the encoding of the internal states of the FSM are not identical in the two implementations, simple self-checking comparator designs cannot be used to check the flip-flop outputs – the comparator design can be very complex. This can degrade the capability of this technique to detect multiple failures and CMFs.

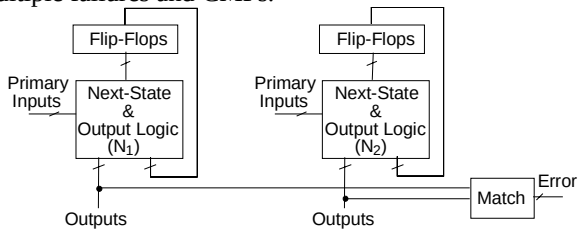

Figure 2.3. Diverse state encoding and logic implementation (DSEDL)

The encoding of the internal states of the second implementation can be looked upon as a transformation of the encoding of the internal states of the first implementation. Formally, if $E_1(s)$ represents the encoding of state s in the first implementation, and $E_2(s)$ represents the encoding of state s in the second implementation, then $E_2(s) = T(E_1(s))$. If T is a "simple" transformation (e.g., linear transformation consisting of xor gates only), then we can design inexpensive checkers (e.g., parity trees) to check the flip-flop outputs.

2.3. Diverse Duplication for Output Logic; Parity Prediction for Next-State Logic (PPNSLDOL & PPDL)

The CED technique ISEDL (Sec. 2.1) has the following advantages over the technique DSEDL (Sec. 2.2): (1) The flip-flop outputs in the two implementations can be compared; hence, if a fault-pair produces non-identical next state outputs, it will be detected; (2) As will be illustrated in Sec. 3, the computational complexity of the analysis of the ISEDL technique is much less than that of the DSEDL technique. However, the ISEDL technique suffers from the problem of having no diversity in the flip-flop contents.

The CED scheme of this section combines the advantages of the ISEDL and DSEDL techniques. We use diverse duplication for the output logic and parity prediction for the next-state logic of the FSM implementation. Figures 2.4a and 2.4b show two implementations of this CED scheme.

In Fig. 2.4a we use simple parity prediction for the next-state logic (with the appropriate constraints on logic sharing). The technique in [Mitra 00b] can be used for synthesizing the diverse implementations of the output logic; the technique in [Touba 97] can be used for synthesizing the next-state logic with parity prediction. This technique is called PPNSDOL (Parity Prediction for Next State Logic and Diverse Output Logic).

In Fig. 2.4b, we use diverse duplication for the next state logic also and check the outputs of the two implementations using a comparator. Then, we add one or more parity trees at the outputs of one of these implementations to generate parity bits. This technique is called PPDL (Parity Prediction and Diverse Logic). The PPDL technique provides more protection from multiple failures and CMFs affecting the next-state logic compared to the PPNSDOL technique (Fig. 2.4a) [Mitra 00a].

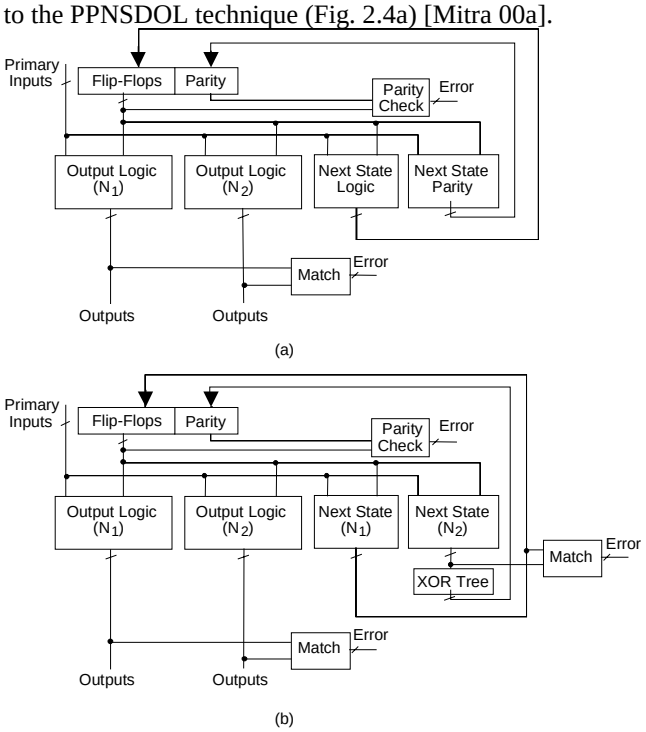

Figure 2.4. Diverse Duplication of sequential logic circuits with parity prediction on flip-flops. (a) PPNSDOL (b) PPDL.

2.4. Simulation Results

In Table 2.1 we report the area overhead of the CED schemes for some MCNC FSM benchmark circuits and the IEEE 1149.1 Boundary-Scan TAP controller [Parker 92] (named TAP).

The circuits were synthesized using the *Sis* tool [Sentovich 92]. For synthesizing diverse implementations of the FSM next-state and output logic we synthesized truth tables with true and complemented outputs using the *Sis* tool. We used *espresso* for two-level minimization, *rugged.script* for multi-level optimization and the LSI Logic G10p library [LSI 96] for technology mapping. For synthesizing FSMs with diverse state encoding, we used two different state encoding algorithms *nova* [Villa 90] and *jedi* [Lin 89]. For synthesizing parity prediction for next-state logic we used the technique described in [Touba 97]. For most cases, the PPDL technique (Fig. 2.4b) generates circuits with less area overhead compared to PPNSDOL (Fig. 2.4a); hence, area results for the PPNSDOL technique are not shown in Table 2.1.

Table 2.1. Area Results (LSI G10p Units)

Circuit Name	Comb. Logic area, # Flip-Flops		
	ISEDL	DSEDL	PPDL
TAP	**371**, 8	406, 8	390, **5**
bbsse	801, 8	**780**, 8	841, **5**
cse	1159, 8	**1127**, 8	1195, **5**
beecount	**197**, 6	208, 6	222, **4**
dk14	**506**, 6	559, 6	526, **4**
ex1	1654, 10	**1639**, 10	1704, **6**

Next, we present simulation results on the vulnerability of CED techniques to multiple failures and CMFs. In dependable systems, it is realistic to assume that a corrective action is initiated after the system generates an error signal. Thus, for any system with CED, data integrity is guaranteed as long as the system does not produce an undetected corrupt output before indicating an error.

For each fault pair (f_i, f_j) affecting the FSM, for each primary input sequence, the FSM produces outputs that belong to the following categories: (1) correct outputs; (2) produces an error signal before producing an undetected erroneous output; (3) produces an undetected erroneous output before producing an error signal. Let $y_{i,j}$ *be the fraction of input sequences for which the system produces only correct outputs*; let $z_{i,j}$ *be the fraction of input sequences for which the system produces an error signal before producing an undetected erroneous output.* We define the term $w_{i,j} = \dfrac{z_{i.j}}{1 - y_{i,j}}$ for the fault pair (f_i, f_j) as the *detected fraction* or *incorrect output detectability*, *which is the fraction of primary input sequences producing erroneous outputs for which the system data integrity is maintained*. If the value of this term is 1 the system either produces correct outputs or indicates erroneous situations when incorrect outputs are produced. If the value is 0 the system never produces any error signal when incorrect outputs are produced. Note that, if a CED-based system produces correct outputs for all input combinations even in the presence of a fault, then the fault is redundant. Similarly, for each fault pair (f_i, f_j), we define the *probability of undetected error* as $x_{i,j} = 1 - y_{i,j} - z_{i,j}$.

We used the following simulation procedure. For each single-stuck-at fault f_i, we simulated exhaustively all fault

pairs to identify another single-stuck-at fault f_j in the same circuit that had the minimum value of $w_{i,j}$ or $x_{i,j}$. Hence, the fault pair (f_i, f_j) can be regarded as a *worst-case fault pair*. Finally, we averaged the $w_{i,j}$'s (or $x_{i,j}$'s) over all the worst-case fault pairs. The primary input sequences during simulation were applied in the following way. For each state *s* of the implemented FSM, we initialized the FSM to state *s* and applied 500-1000 pseudo-random primary input sequences generated by an LFSR; each primary input sequence was of length of 100-200. The results for fault pairs in the combinational logic parts are shown in Table 2.2. The benchmark circuits are small enough so that the simulation procedure can be completed. For Table 2.2, the results for the CED techniques PPNSDOL and PPDL (Fig. 2.4a and 2.4b) are not shown separately because the results for PPDL (Fig. 2.4b) are the same as that for ISEDL (Fig. 2.2). Moreover, as discussed in [Mitra 00c], the results for PPNSDOL are worse than that for PPDL.

Table 2.2. Worst-case analysis of faults in comb. logic

Circuit Name	Incorr. O/p Detectability		Prob. Undet. Error	
	ISEDL, PPDL	DSEDL	ISEDL, PPDL	DSEDL
TAP	**0.4**	0.06	**0.6**	0.94
beecount	0.33	**0.37**	0.38	**0.33**
cse	**0.46**	**0.46**	**0.11**	0.12
dk14	**0.54**	**0.54**	**0.34**	0.38

The results of Table 2.2 indicate that, for the simulated designs, the protection provided by the ISEDL or PPDL techniques against multiple failures or CMFs in the combinational logic is better than or comparable to that of the DSEDL technique (diverse state encoding). The DSEDL technique has very low incorrect output detectability for the TAP controller FSM. This is mainly due to the fact that, for the DSEDL technique, the combinational logic can produce non-identical errors on the flip-flop inputs; however, since there is no "easy way" to check the flip-flop contents, these errors cannot be detected and eventually the faults eventually produce identical errors. Table 2.3 shows simulation results for faults affecting only the flip-flop outputs.

Table 2.3. Worst-case analysis of faults on flip-flops

Circuit Name	Incorr. O/p Detectability			Prob. Undet. Error	
	ISEDL	DSEDL	PPNSDOL, PPDL	DSEDL	PPNSDOL, PPDL
TAP	0	**0.74**	0.54	**0.26**	0.46
cse	0	0.30	**0.43**	**0.14**	0.34
dk14	0	0.40	**0.49**	0.6	**0.51**
dk16	0	0.48	**0.54**	0.52	**0.46**
ex1	0	**0.58**	0.48	**0.42**	0.52

The results of Table 2.2 and 2.3 demonstrate the effectiveness of the PPDL technique of Fig. 2.4b (diverse combinational logic implementation, parity prediction for flip-flops and generation of parity bit through an XOR-tree from a next-state logic implementation) for implementing CED in the simulated designs.

It may be noted here that, if transient faults create bit-flips (rather than bit-stucks) in the flip-flops of a sequential circuit, then the CED technique based on diverse state encoding technique based on linear transformations, which is an extension of the idea of parity prediction as described at the end of Sec. 2.2, is expected to outperform the other techniques (ISEDL, PPNSDOL or PPDL) so far as data integrity is concerned.

In the next section we describe a formal technique for analyzing each of the CED schemes; the discussion also shows that the computational complexity of analyzing the DSEDL technique is at least an order of magnitude higher than that of the ISEDL, PPNSDOL or PPDL techniques.

3. Analysis of CED schemes

Suppose that we are given two implementations N_1 and N_2 of an FSM M. The FSM M can be characterized by a state table [McCluskey 86] which can be formally represented by the following set $\{I, O, S, T, L\}$. Here, I is the *set of primary input combinations*, O is the *set of primary output combinations* and S is the *set of internal states*. T is the *transition logic* which can be looked upon as a mapping from $S \times I$ to S. L is the *output logic* which can be represented as a mapping from $S \times I$. An *input distribution* of an FSM is given by the conditional probability distribution $P(i|s)$ for all $i \in I$ and $s \in S$. $P(i|s)$ is the conditional probability that a primary input combination $i \in I$ is applied to the FSM when it is in state s. For the current paper, we assume that all primary input combinations are equally likely for all states. However, for specific systems, the input distribution can be approximated using trace simulations.

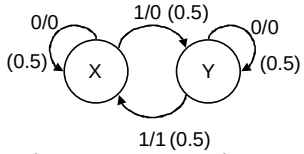

Figure 3.1. Example FSM

For example, consider the example FSM in Fig. 3.1. For this FSM, S = {X, Y}; I = {0, 1}, O = {0, 1}. The next-state logic T is given by: T(X, 0) = X, T(X, 1) = Y, T(Y, 0) = Y and T(Y, 1) = X. The output logic L is: L(X, 0) = 0, L(X, 1) = 0, L(Y, 0) = 0 and L(Y, 1) = 1. For any state, the probability that the primary input has value 0 (or 1) is 0.5. Figures 3.2a and 3.2b show two implementations N_1 and N_2 of the FSM in Fig. 3.1. If we use these two implementations for CED we have a DSEDL CED technique.

Let us suppose that faults f and g affect implementations N_1 and N_2, respectively. We can construct faulty FSMs M_f = {I, O, S_f, T_f, L_f} and M_g = {I, O, S_g, T_g, L_g} in the presence of f and g, respectively. The two faulty FSMs are shown in Fig. 3.2c and 3.2d, respectively. Next, we can construct the product machine $K = M \times M_f \times M_g$, as follows. The set of states of K is given by $K_S = S \times S_f \times S_g$; i.e., each state of K can be represented as a tuple (a, b, c), where $a \in S$, $b \in S_f$ and $c \in S_g$. The transition logic K_T of K is given by the following mapping: K_T[(a, b,

c), i] = [$T(a, i)$, $T_f(b, i)$, $T_g(c, i)$] where (a, b, c) $\in S \times S_f \times S_g$ and $i \in I$. The output logic K_L of the product FSM K is a mapping from $K_S \times I$ to $O \times O \times O$ and is defined by K_L[(a, b, c), i] = [$L(a, i)$, $L_f(b, i)$, $L_g(c, i)$] where (a, b, c) $\in S \times S_f \times S_g$ and $i \in I$. The input distribution of the product FSM K is defined as $P[i|(a, b, c)] = P(i|a)$ in FSM M, where (a, b, c) $\in S \times S_f \times S_g$ and $i \in I$. Figure 3.3 shows the product FSM K for the example in Fig. 3.2.

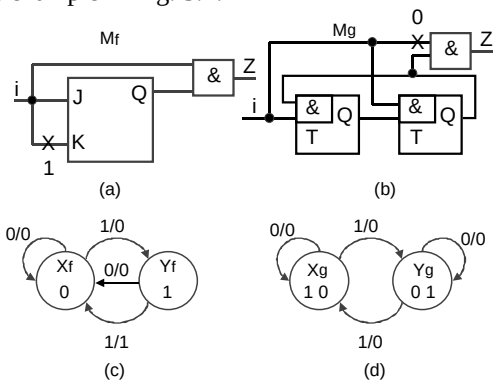

Figure 3.2. FSM implementations with faults. (a) Implementation with fault f. (b) Implementation with fault g. (c)-(d) State diagram of implementation with fault f and g.

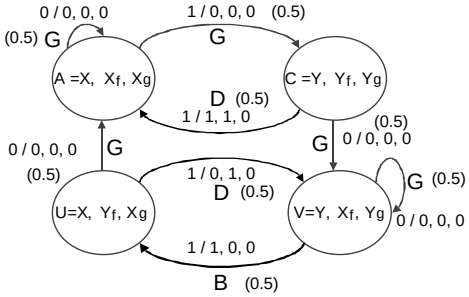

Figure 3.3. Product FSM: Good, Bad, Detecting transitions

The state transitions of the product FSM K can be classified into following categories: (1) *good* (G) transition, (2) *detecting* (D) transition, and (3) *bad* (B) transition.

A transition from state (a, b, c) under input combination i is a *good transition* if the output produced by the FSM M is the same as the outputs produced by M_f and M_g. Formally, a transition K_T[(a, b, c), i] = [$T(a, i)$, $T_f(b, i)$, $T_g(c, i)$] is a good transition if and only if $L(a, i) = L_f(b, i) = L_g(c, i)$.

A transition from state (a, b, c) under input combination i is a *detecting transition* if the outputs produced by M_f and M_g are different. Formally, a transition K_T[(a, b, c), i] = [$T(a, i)$, $T_f(b, i)$, $T_g(c, i)$] is a detecting transition if and only if $L_f(b, i) \neq L_g(c, i)$.

A transition from state (a, b, c) under input combination i is a *bad transition* if M_f and M_g produce identical erroneous outputs (different from the output produced by M). Formally, a transition K_T[(a, b, c), i] = [$T(a, i)$, $T_f(b, i)$, $T_g(c, i)$] is a bad transition if and only if $L(a, i) \neq L_f(b, i)$ and $L_f(b, i) = L_g(c, i)$.

Figure 3.3 shows the labels of the transitions of the product FSM K. For a CED technique based on diverse logic implementation but identical state encoding, the outputs of the corresponding flip-flops in the two implementations can be compared. This means that any state (a, b, c) in the product machine K detects the presence of a fault if $b \neq c$. All such states can be merged into a single state *Detected*. This reduction is not possible for a CED scheme with diverse state encoding unless there is an "easy" way to check that both the implementations are in the same state. All detecting transitions in the product machine K can be redirected to the *Detected* state; all edges starting from the states that are merged into the *Detected* state can be deleted. There is no outgoing edge from the *Detected* state. All bad transitions in the product FSM K can be redirected to a new state *Error*. There is no outgoing edge from the *Error* state. After these reductions, all unreachable states and edges starting from them in the final FSM can be deleted. Figure 3.4 illustrates these reduction techniques for the product FSM in Fig. 3.3 for the case when the internal states of the two FSMs are checked. The system never enters an *Error* state and the data integrity in the presence of the fault pair is 1.

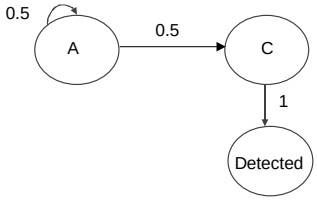

Figure 3.4. Reduced FSM with comparator states

The data integrity of the CED system at time t in the presence of faults can be defined in the following way. For each state s of the original fault-free FSM, we identify the state S = (a, b, c) in the product FSM such that $a = s$, and b and c are the corresponding the states in the two implementations with faults; next, we calculate the probability $E(S, t)$ of being in the *Error* state in the product FSM at time t starting from state S. This can be calculated using straightforward Markov analysis techniques and tools like SHARPE (http://www.ee.duke.edu/~kst). The data integrity of the CED system in the presence of a given fault pair is equal to $\sum_s P(s)[1-E(S,t)]$. Here, $P(s)$ is the stationary probability of state s in the original fault-free FSM. For very low failure rates, it is realistic to assume that the original FSM reaches a stationary probability state before a fault affects the FSM. Analysis of CED schemes based on diverse duplication of output logic and parity prediction of next-state logic is similar to the analysis technique described above and is not repeated.

3.1. Computational Complexity of the Analysis

Theoretically, the above analysis technique is computationally intensive because of the following problems. The analysis technique may run into memory problems due to possible state space explosion during the computation of the product FSM. For example, if the original FSM has 64 states, it is theoretically possible that the product FSM will have $64^3 = 262{,}144$ states if we use the DSEDL technique (without comparators comparing the flip-flop outputs). Moreover, if the original FSM has a large number of primary inputs, then the construction of the product FSM will be very time consuming if we have to compute the state transition of the product FSM from each state for each primary input combination. In Table 3.1, we show the characteristics of the 1149.1 Boundary-Scan TAP controller and the MCNC FSM benchmark circuits and the average and the maximum number of states in the product FSM over all single stuck-at fault pairs.

Most of the FSM benchmarks in the MCNC benchmark suite have the number of states not more than 32. The TAP FSM has 16 states and a single primary input. A similar observation can be made about the internal benchmark FSM specifications of CAD companies. This is perhaps because FSMs used in real designs are designed as interacting state machines.

Table 3.1. Characteristics of designs for which exact analysis of ISEDL, PPNSDOL and PPDL is feasible

Circuit Name	# PI, # PO, # States	Avg. # states in product FSM	Max. # states in product FSM
TAP	1, 7, 16	22	90
cse	7, 7, 16	23	140
dk16	2, 3, 27	45	342
ex1	9, 19, 18	25	160
sand	11, 9, 32	60	600

However, there are some FSM specifications with the number of states approximately 97 or 135; moreover, for FSMs with a large number of primary inputs, an exact analysis for each input combination can be very time consuming (FSMs *s420*, *s510*, *s820* and *scf* with 19, 19, 18 and 27 primary inputs, respectively). For these FSM benchmarks approximate techniques must be devised.

4. Conclusions

We studied the problem of implementing concurrent error detection (CED) based on diverse duplication in sequential logic circuits. We examined three different techniques for this purpose. Our simulation results demonstrate that the CED technique based on diverse duplication of combinational logic and parity prediction of flip-flop contents is most efficient in protecting sequential logic circuits against multiple and common-mode failures. We also described an exact technique to analyze the data integrity of sequential logic circuits with CED. Our results on MCNC benchmark circuits show that the exact analysis technique is feasible for many (80 %) benchmark circuits although theoretically it can suffer from state space explosion problems. Future research must focus on extending the idea of parity prediction for next-state logic to generate "simple" transformations for diverse state encoding and developing efficient analysis techniques that do not suffer from state explosion problems and can handle FSM specifications with a large number of primary inputs.

5. Acknowledgments

This work was supported by DARPA under Contract No. DABT63-97-C-0024 (ROAR project).

6. References

[Avizienis 84] Avizienis, A. and J. P. J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *IEEE Computer*, pp. 67-80, August 1984.

[Briere 93] Briere, D. and P. Traverse, "Airbus A320/A330/A340 Electrical Flight Controls: A family of fault-tolerant systems," *Proc. FTCS*, pp. 616-623, 1993.

[Lala 94] Lala, J. H. and R. E. Harper, "Architectural principles for safety-critical real-time applications," *Proc. of IEEE*, vol. 82, no. 1, pp. 25-40, January 1994.

[Lin 89] Lin, B., and A. R. Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description language," *International Conf. VLSI*, pp. 414-417, 1989.

[LSI 96] *G10-p Cell-Based ASIC Products*, LSI Logic.

[McCluskey 86] McCluskey, E. J., *Logic Design Principles*, Prentice-Hall, 1986.

[Mitra 99a] Mitra, S., N. R. Saxena and E. J. McCluskey, "A Design Diversity Metric and Reliability Analysis for Redundant Systems," *Intl. Test Conf.*, pp. 662-671, 1999.

[Mitra 00a] Mitra, S., N. R. Saxena and E. J. McCluskey, "Common-Mode Failures in Redundant VLSI Systems: A Survey," *IEEE Trans. Reliability*, Special Section on Fault-Tolerant Systems, 2000.

[Mitra 00b] Mitra, S. and E. J. McCluskey, "Combinational Logic Synthesis for Diversity in Duplex Systems," *Proc. Intl. Test Conf.*, pp. 179-188, 2000.

[Mitra 00c] Mitra, S. and E. J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?," *Proc. Intl. Test Conf.*, pp. 985-994, 2000.

[Parker 92] Parker, K. P., *Boundary Scan Handbook*, Kluwer Academic Publishers, 1992.

[Riter 95] Riter, R., "Modeling and Testing a Critical Fault-Tolerant Multi-Process System," *FTCS*, pp. 516-521, 1995.

[Saxena 00] Saxena, N. R., *et al.*, "Dependable Computing and On-Line Testing in Adaptive and Reconfigurable Systems," *IEEE Design & Test of Comp.*, Jan-Mar 2000.

[Sentovich 92] Sentovich, E. M., *et al.*, "SIS: A System for Sequential Circuit Synthesis," *ERL Memo. No. UCB/ERL M92/41*, EECS, UC Berkeley, CA 94720.

[Siewiorek 92] Siewiorek, D. P., R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 1992.

[Spainhower 99] Spainhower, L. and T. A. Gregg, "S/390 Parallel Enterprise Server G5 fault tolerance," *IBM Journal of Res. and Dev.*, pp. 863-873, Sept./Nov. 1999.

[Touba 97] Touba, N. A. and E. J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection," *IEEE Trans. CAD*, pp. 783-789, July 1997.

[Villa 90] Villa, T., and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-level Logic Implementation," *IEEE Trans. CAD*, Vol. 9, No. 9, pp. 905-924, Sept. 1990.

[Zeng 99] Zeng, C., N. R. Saxena and E. J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection," *Proc. Intl. Test Conf.*, pp. 672-680, 1999.